# The Friedmann-Gleichung Machine: A Governed Enterprise Runtime for Generative Apps as a Service

**Author:** Marco van Hurne

**Affiliations:** Eigenvector Research | Inholland University of Applied Sciences

**Contact:** marco.vanhurne@eigenvector.eu | henk.dekoning@eigenvector.eu | marco.vanhurne@inholland.nl

**Date:** May 2026

## Abstract

The enterprise transition from deterministic automation to agentic artificial intelligence introduces profound governance challenges, particularly in high-stakes, exception-rich environments classified as Zone III. While current enterprise AI products optimize for interaction quality, they frequently under-specify process continuity, runtime governance, semantic admissibility, and auditability. This paper introduces the Friedmann-Gleichung Machine (FGM), a novel neuro-symbolic architecture designed to ensure that agentic AI systems operate in a manner that is both semantically coherent and compliant with enterprise policies. The FGM acts as a governed enterprise runtime that dynamically generates full applications—including user interfaces, logic, workflows, and data integrations—based on user intent. By integrating a Progressive Validation Gateway for dual-layer semantic and policy validation, LangGraph for Markov Decision Process (MDP) based orchestration, and OpenUI for streaming interface generation, the FGM provides a verifiable, pre-execution validation layer. We formalize the core mechanisms mathematically, introducing five assurance levels and formal theorems regarding semantic

soundness, policy compliance, and MDP convergence. Furthermore, we present a comprehensive latency analysis, a STRIDE threat model, and a critical failure modes analysis addressing ontology failure, SPARQL hallucination, latency stacking, and prompt injection. The proposed architecture provides a blueprint for bounded autonomous enterprise execution, enabling organizations to scale agentic systems while preserving accountability, semantic validity, economic rationality, and institutional control.

## 1. Introduction

The emergence of Large Language Models (LLMs) has catalyzed the development of sophisticated agentic AI systems—autonomous agents capable of reasoning, planning, and executing complex multi-step tasks by interacting with external tools and APIs [1]. These systems promise unprecedented efficiency gains by automating high-level business processes. However, the key problem is not that enterprises lack AI capability. The problem is that agentic AI can generate actions, interfaces, and workflows faster than existing governance models can validate them. Their black-box nature, susceptibility to hallucination, and lack of formal grounding in business context create a perfect storm of operational and compliance risks [2]. A comprehensive conceptual overview of the FGM architecture is provided in Appendix C, while the detailed technical architecture is documented in Appendix D.

The primary bottleneck in scaling agentic AI is no longer model capability; it is governance [3]. Analysis of 177 documented agentic AI deployments across 20 industry sectors reveals that technical failures account for only 16% of deployment failures, whereas governance failures account for 28% and data quality issues for 34% [3]. As organizations move beyond pilot projects into production deployments, they discover that the decisive question is not merely whether an agent can act, but whether the selected execution pattern creates defensible value under strict governance, intervention, and evidence constraints.

This challenge defines what we call Zone III: the operating regime between rigid workflow and unconstrained autonomy [4]. Zone III processes are structured enough to justify automation, but too semantically unstable, exception-sensitive, evidence-bearing, and governance-exposed for conventional BPM alone. At the same time, they are too consequential to entrust to loosely governed copilots or free-form agent stacks.

To address this, we introduce the Friedmann-Gleichung Machine (FGM), a governed enterprise runtime for Generative Apps as a Service. The FGM is not simply an agent framework, not simply a BPM extension, and not simply a UI generator. It is a controlled emergence runtime for enterprise applications, where processes and interfaces are not hardcoded in advance but are allowed to come into existence only inside a validated governance envelope. The system does not merely execute applications; it defines the conditions under which applications are allowed to come into existence. In biological terms, the FGM enforces a form of architectural *orthogenesis*—the theory that evolution follows a directed, inherent path. In the FGM, an initial user intent does not evolve randomly through stochastic LLM generation; rather, it is constrained by the ontology and policy engines to follow a predetermined, governed trajectory toward a compliant application state.

This paper makes the following contributions:

- - We define the FGM architecture, integrating generative UI, agent orchestration, and neuro-symbolic governance.
- - We introduce a Progressive Validation Gateway with dual-layer semantics and compiled semantic validation to address critical failure modes.
- - We formalize the architecture mathematically, introducing five assurance levels and formal theorems for the validation gateway.
- - We provide a comprehensive latency analysis and performance control model.

- - We present a STRIDE threat model and a formalization of the Evidence Bundle as a cryptographic protocol.

This paper is accompanied by a suite of supplementary appendices, which are available as separate downloadable documents via the paper's repository on ArXiv/TechRxiv. Appendix A presents 12 detailed architectural execution flows with sequence and swimlane diagrams. Appendix B provides a reference implementation and control specification. Appendices C and D offer the conceptual and detailed technical architecture, respectively. Appendix E provides a developer implementation guide with concrete technology stack specifications. Appendix F contains core component code examples. Appendix G presents the sprint-based implementation roadmap. Appendix H provides a patent analysis of the three core claim families. Readers are encouraged to consult the supplementary appendices alongside the main paper; the appendices are listed with their individual download links in the Supplementary Materials section at the end of this paper.

## 2. Related Work

The FGM architecture stands at the intersection of several research streams: process automation suitability, runtime governance, neuro-symbolic AI, and generative user interfaces.

### *2.1 Process Automation Suitability and Zone III*

The Process Automation Suitability Framework (PASF) establishes that not all enterprise work is equally suitable for automation [3] [4]. PASF categorizes enterprise processes into four zones based on structurability, risk profile, reversibility, and exception density. Zone III ("Automate with Caution") represents processes with high structurability but high risk and governance exposure, requiring mandatory Human-in-the-Loop (HITL) and strict runtime policy enforcement [3]. Recent systematic literature reviews confirm that agentic AI is increasingly

deployed in these complex enterprise operations, particularly in supply chain and ERP contexts [5].

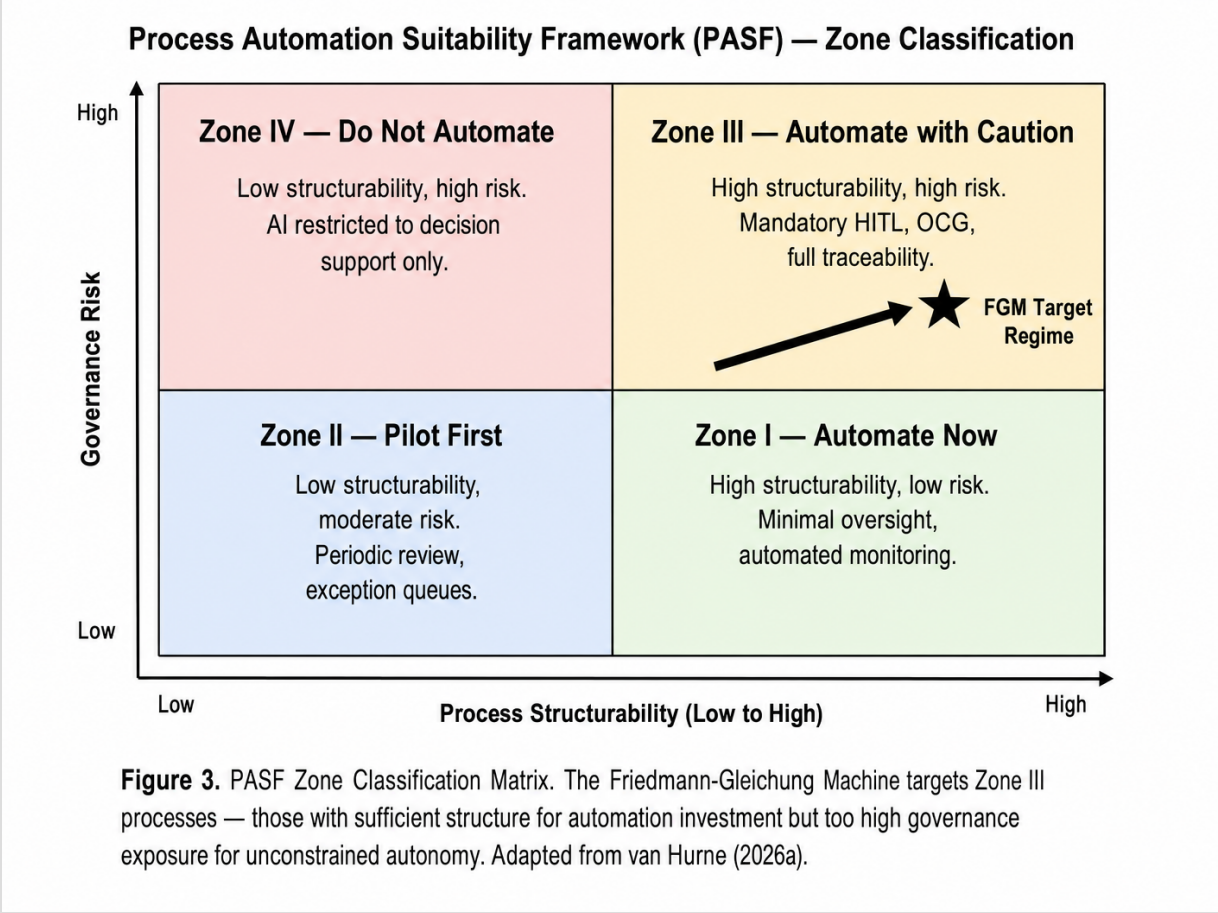


**Figure 3.** PASF Zone Classification Matrix. The Friedmann-Gleichung Machine targets Zone III processes — those with sufficient structure for automation investment but too high governance exposure for unconstrained autonomy. Adapted from van Hurne (2026a).

***Figure.*** *Figure 1: PASF Zone Classification Matrix. The Friedmann-Gleichung Machine targets Zone III processes. Adapted from van Hurne (2026a).*

### *2.2 Runtime Governance and Bounded Autonomy*

Current AI governance frameworks often provide static compliance guidelines but lack executable runtime enforcement [6]. The Governed Runtime Architecture Framework (GRAF) addresses this by conceptualizing agentic governance as a continuous runtime control system comprising five layers: Policy Enforcement, Observability & Traceability, Human Control, Assurance & Evidence, and Value Recognition [6]. Recent work by Kaptein et al. (2026)

formalizes runtime governance as "policies on paths," demonstrating that prompt-level instructions are insufficient for enterprise compliance [7]. Similarly, Gough (2026) frames this as "bounded autonomy," requiring behavioral specification languages and runtime enforcement architectures [8]. The FGM implements these principles to ensure that control remains active during execution.

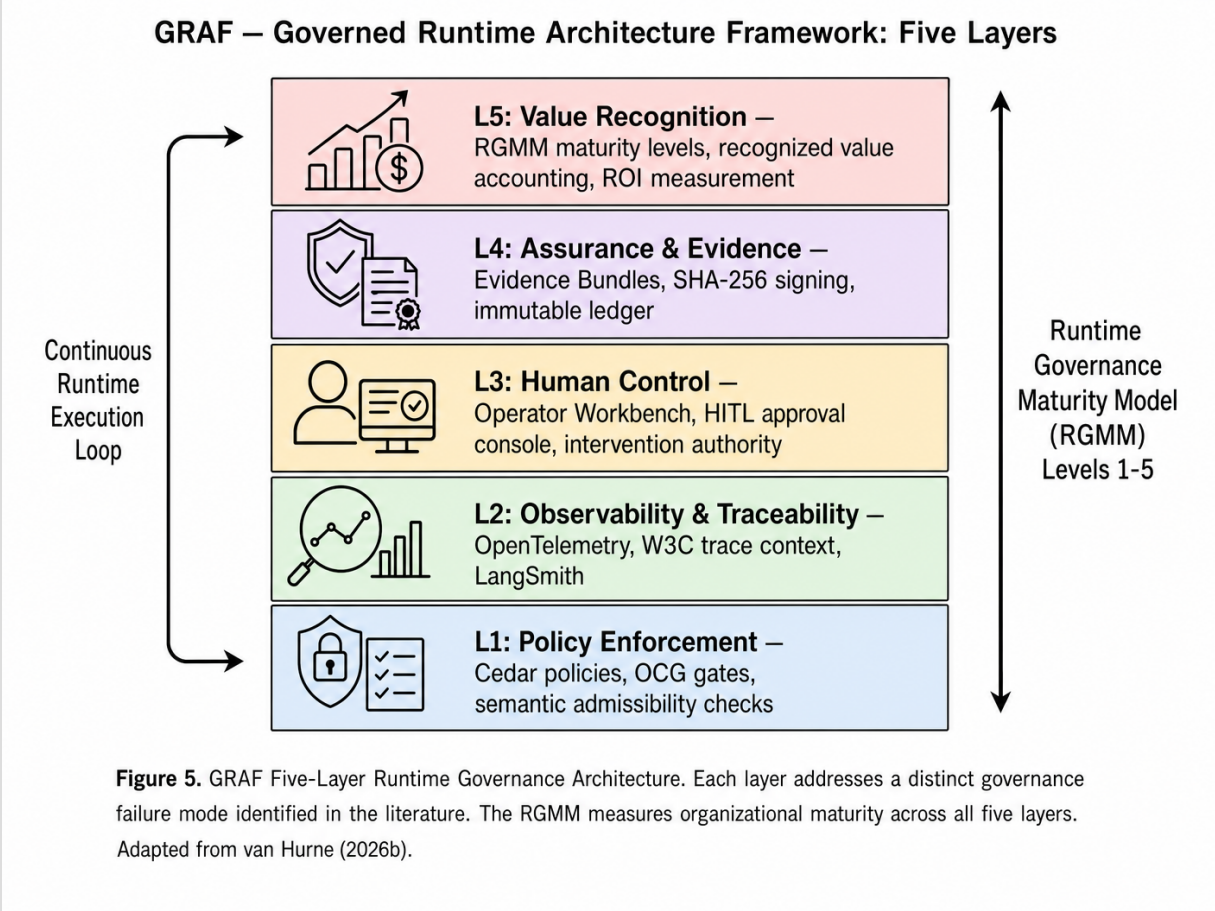


***Figure.*** *Figure 2: GRAF Five-Layer Runtime Governance Architecture.*

### *2.3 Neuro-Symbolic AI and the OCG*

Agentic frameworks like ReAct [9] rely on LLMs for reasoning, which lack intrinsic understanding of business semantics. Neuro-symbolic AI combines neural networks with symbolic reasoning (e.g., ontologies) to provide formal guarantees [2]. The Ontological

Compliance Gateway (OCG) implements a validation model separating semantic coherence validation from policy compliance checks [2]. The application of neuro-symbolic AI for knowledge graph construction and reasoning is gaining significant traction, particularly in complex domains like manufacturing [10] [11]. Furthermore, recent research demonstrates that LLMs can be effectively grounded in real-world semantics through symbolic ontological knowledge [12].

### *2.4 Generative UI and the Generative Web*

The transition from static interfaces to on-the-fly generated experiences is driven by agentic browsing and open UI protocols [13]. Generative UI produces ephemeral, bespoke applications tailored to user intent. Open-source frameworks like OpenUI provide streaming-first rendering languages that significantly reduce token overhead compared to JSON, enabling real-time generation of complex enterprise dashboards [13]. Leviathan et al. (2025) demonstrate that LLM-generated UIs are strongly preferred by human raters over standard LLM outputs, highlighting the viability of this paradigm [14].

### *2.5 Prompt Injection and Security*

OWASP identifies prompt injection as the primary threat (LLM01:2025) to large language model applications [15]. Gulyamov et al. (2026) provide a comprehensive review of these vulnerabilities, noting that even a small number of carefully crafted documents can manipulate AI responses through RAG poisoning [16]. To address these threats, frameworks like Zero Trust for AI agents [17] and the ASIDE (Architectural Separation of Instructions and Data for Embeddings) principles [18] have been proposed. The FGM incorporates these security paradigms deeply into its architecture.

## 3. The Friedmann Analogy and Controlled Application Expansion

The system's name derives from the Friedmann equations, which describe the expansion of the universe. By analogy, the FGM describes the "expansion" of a full application from a dense initial state (the user intent). A dense initial intent expands into an application space. Semantic density, generative capability, governance friction, constraints, policy, and runtime evidence determine how that expansion is allowed to happen. The analogy should support the architecture, not replace it: it provides a conceptual framework for understanding how a single natural language prompt can deterministically unfold into a complex, multi-component enterprise application under strict governance constraints.

## 4. Architecture Overview

The FGM is designed to dynamically expand a single user intent into a fully functional, governed application. The architecture consists of core modules spanning interaction, orchestration, governance, knowledge, and foundation layers, providing a deep, component-level integration.

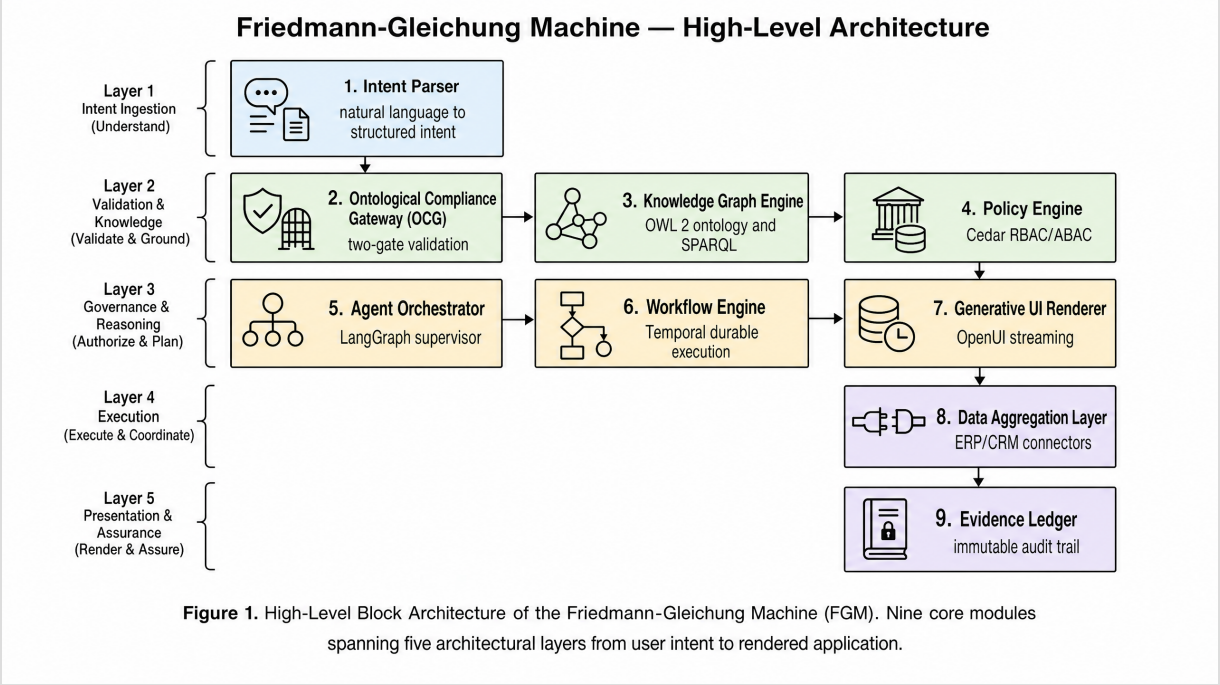


**Figure 1.** High-Level Block Architecture of the Friedmann-Gleichung Machine (FGM). Nine core modules spanning five architectural layers from user intent to rendered application.

***Figure.*** *Figure 3: High-Level Block Architecture of the Friedmann-Gleichung Machine (FGM).*

The revised architecture keeps LangGraph for agent orchestration, Temporal for durable execution and HITL pauses, OpenUI or AG-UI for streaming generative UI, Apache Jena or another graph/ontology engine for semantic validation, Cedar for policy-as-code, Keycloak for identity, Qdrant or another vector database for unstructured retrieval, Kong or another API gateway for traffic control, OpenTelemetry for observability, and an immutable ledger for evidence storage. This is not a random technology stack but a layered control system for bounded autonomous execution.

## 5. Progressive Validation Gateway

The core of the FGM's safety guarantees is the Progressive Validation Gateway (formerly the OCG). Before any agent action is executed, the proposed intent must pass through this gateway. It contains two major logical dimensions: semantic validation and policy validation. A complete reference implementation of this pipeline is provided in Appendix B, Section B.3.

Twelve concrete execution flows, including happy-path, blocking, and HITL scenarios, are documented in Appendix A.

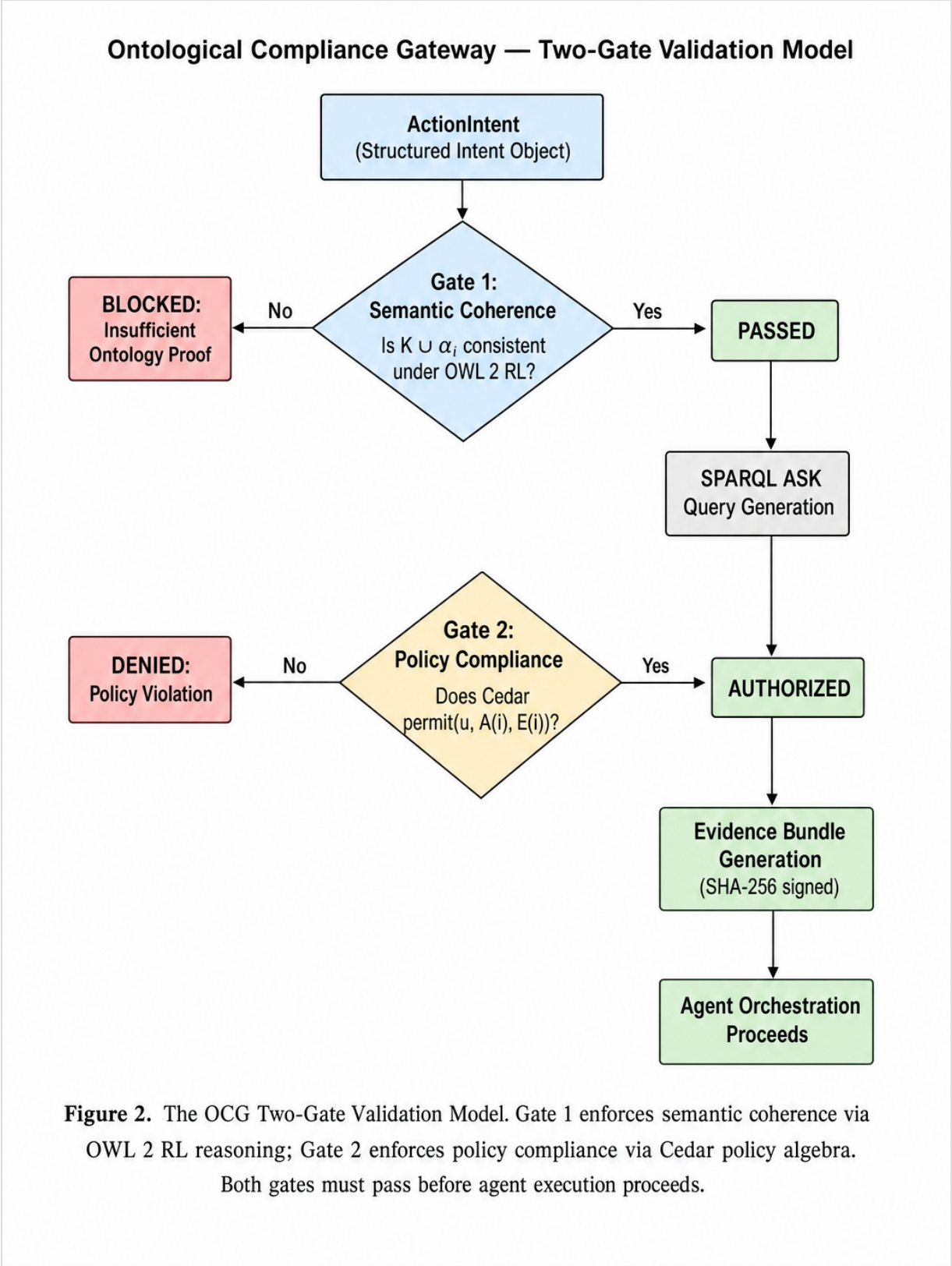


**Figure 2.** The OCG Two-Gate Validation Model. Gate 1 enforces semantic coherence via OWL 2 RL reasoning; Gate 2 enforces policy compliance via Cedar policy algebra. Both gates must pass before agent execution proceeds.

***Figure.*** *Figure 4: The Progressive Validation Gateway Model.*

Semantic validation now has two internal levels (Dual-Layer Semantics, discussed in Section 6). Policy validation remains deterministic and uses principal, action, resource, context, and environmental attributes evaluated against a Cedar policy engine to enforce Role-Based Access Control (RBAC), Attribute-Based Access Control (ABAC), and Segregation of Duties (SoD) [2]. See Appendix A, Flow 4 for a detailed illustration of a policy violation blocking scenario.

## 6. Dual-Layer Semantics

The original architecture assumed that the enterprise ontology is the semantic authority of the system. This is powerful but dangerous, because the ontology can become both the single source of truth and a single point of failure. The revised architecture adopts a middle-way model: progressive semantic validation. The dual-layer semantic model is specified in detail in Appendix B, Section B.4, and the formal ontology management process is illustrated in Appendix A, Flow 10.

### *6.1 Knowledge Graph Layer*

The first layer is a lightweight knowledge graph used for fast entity checks, relationship checks, contextual grounding, schema awareness, and runtime lookup. The knowledge graph answers whether entities, relationships, and attributes exist in the enterprise context. Zone I and Zone II requests may use the knowledge graph alone, protecting latency and reducing maintenance burden. See Appendix A, Flow 1 for a Zone I happy-path execution using only the knowledge graph layer.

### *6.2 Formal Ontology Layer*

The second layer is a stricter ontology layer used only for high-risk or high-consequence validation, especially Zone III requests. The formal ontology answers whether the proposed action is logically admissible under stronger semantic rules. Zone III requests must use both

the knowledge graph and the ontology. This preserves formal validation where it actually matters. See Appendix A, Flow 2 for a Zone III happy-path execution and Flow 3 for a semantic invalidity blocking scenario.

## 7. Agent Orchestration and Durable Execution

Orchestration is handled by a LangGraph-based supervisor agent that delegates tasks to specialized sub-agents. To ensure process continuity and handle long-running tasks or HITL pauses, the FGM integrates Temporal.io for durable execution. This guarantees that agent states survive system interruptions and can be resumed seamlessly [4]. The orchestration model is specified in Appendix B, Section B.6. Multi-agent parallel execution is illustrated in Appendix A, Flow 7, and MCP tool failure with graceful degradation is shown in Appendix A, Flow 11.

## 8. Generative UI and Ephemeral Application Formation

The interaction layer captures user intent and renders the final application utilizing Generative UI (via OpenUI and AG-UI protocols) to stream component trees directly to the client. This allows the system to generate bespoke interfaces on the fly, personalized to the user's context and the specific task [13]. The generative interface model is specified in Appendix B, Section B.7. Concrete implementation details using Next.js 14, React 18, and the AG-UI SDK are provided in Appendix E, Section 2 and Appendix F.

## 9. Evidence Bundles and Cryptographic Traceability

Every validation decision records the graph version, ontology version, policy version, and validation path inside the Evidence Bundle. The Evidence Bundle captures all of this, including the validation route used, skipped validations, reason for skipping, risk classification, graph snapshot, ontology snapshot, policy snapshot, HITL decision, and final execution result. This bundle is cryptographically signed and appended to an immutable ledger

for auditability [2]. The Evidence Bundle specification is detailed in Appendix B, Section B.8. The complete lifecycle of an Evidence Bundle, including its Merkle chain construction, is illustrated in Appendix A, Flow 8. The patentability of this construct is analyzed in Appendix H, Section 2.

## 10. Critical Failure Modes and Control Architecture

This section covers four major risk areas and the control architecture designed to mitigate them.

### *10.1 Ontology as Semantic Authority and Failure Point*

As discussed in Section 6, relying solely on a formal ontology creates a single point of failure. The dual-layer semantic model mitigates this by using a lightweight knowledge graph for fast checks and reserving the strict ontology for high-risk Zone III validation.

### *10.2 SPARQL Hallucination and Compiled Semantic Validation*

The current architecture depends on the LLM translating natural language intent into SPARQL ASK queries, which is a silent failure mode. The LLM must not freely generate SPARQL. Instead, the LLM produces a constrained intermediate representation, such as structured JSON or an ActionIntent object. A deterministic compiler then converts this validated intermediate structure into graph queries or SPARQL queries. This "compiled semantic validation" includes grammar-constrained generation, schema grounding, ontology vocabulary lookup, static query linting, unknown predicate detection, query dry-runs, counterfactual validation, and explainable failure responses. SPARQL is generated by a compiler or query builder, not by model creativity. The `ActionIntent` schema is specified in Appendix B, Section B.2, and a working implementation of the semantic compiler using FastAPI and Apache Jena Fuseki (OWL 2 RL) is provided in Appendix F, Section 1.

### *10.3 Latency Stacking and Runtime Performance Control*

The FGM pipeline includes multiple stages that can stack latency, making the runtime unusable. The performance control model splits the system into fast lanes and governed lanes. Zone I requests use fast graph checks, cached policy checks, streaming UI skeletons, and lightweight models. Zone III requests use the full governance path. Optimizations include caching at multiple layers (graph query caching, ontology reasoning cache, policy decision memoization, model output caching), speculative pre-validation for likely next actions, asynchronous evidence bundling, and OpenUI/AG-UI streaming so the user sees progressive application formation instead of waiting for a complete backend cycle.

### *10.4 Prompt Injection and Adversarial Context Defense*

The FGM must assume that retrieved documents, user prompts, tool outputs, and agent messages can be adversarial. The architecture employs an ASIDE-like separation between instructions and data. Retrieved content must never be treated as executable instruction. Tool calls must be gated only through the Progressive Validation Gateway. Agents operate with short-lived, least-privilege delegated credentials through Keycloak. Tools accept only signed execution requests issued after validation. The architecture limits impact rather than pretending to eliminate the threat, using context sanitization, instruction-data separation, signed tool contracts, prompt injection detection, RAG poisoning resistance, policy-based blast radius limitation, and multi-agent verification for high-risk actions. A complete prompt injection attack scenario with ASIDE defense is illustrated in Appendix A, Flow 9. The failure mode control model is specified in Appendix B, Section B.9.

## 11. Mathematical Formalization

To provide rigorous guarantees for the FGM, we formalize its core mechanisms mathematically.

### *11.1 Assurance Levels and Revised Theorems*

We define different assurance levels:

- - **Level 1:** Existence and relationship validation through the knowledge graph.
- - **Level 2:** Formal semantic admissibility through the ontology.
- - **Level 3:** Policy authorization through Cedar or another deterministic policy engine.
- - **Level 4:** Execution traceability through cryptographic Evidence Bundles.
- - **Level 5:** Operator accountability through HITL and audit review.

Theorems are framed only for the conditions under which the formal ontology layer is actually invoked. For lightweight graph validation, we use weaker claims such as structural plausibility, contextual grounding, or schema consistency.

**Theorem 1 (Semantic Admissibility):** *Given a consistent enterprise ontology \mathcal{K} expressed in OWL 2 RL, if the compiled semantic validation evaluates to True, then the intent i is semantically admissible with respect to \mathcal{K} at Assurance Level 2.*

**Theorem 2 (Policy Authorization):** *Given a deterministic policy evaluation function Eval(i, u, \mathcal{P}), if the policy validation evaluates to True, then the execution of i by principal u strictly adheres to the defined access control policies at Assurance Level 3.*

**Theorem 3 (MDP Convergence under Bounded Autonomy):** *In the FGM orchestrator, if the state space S is finite and the transition function T is constrained by the Progressive Validation Gateway, then the optimal policy \pi^* converges to a sequence of actions that maximizes expected reward while strictly remaining within the bounded autonomy envelope.*

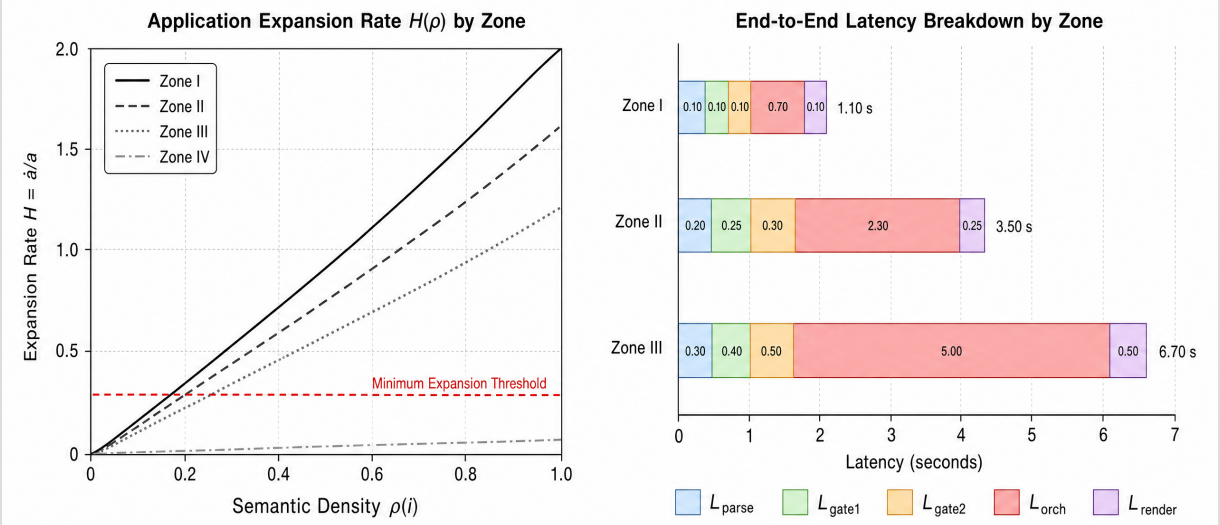


**Figure 4.** Left: Application expansion rate $H$ as a function of semantic density $\rho(i)$ for each PASF zone. Right: End-to-end latency decomposition per zone. The orchestration layer $L_{orch}$ dominates Zone III latency.

***Figure.*** *Figure 5: Mathematical Models.*

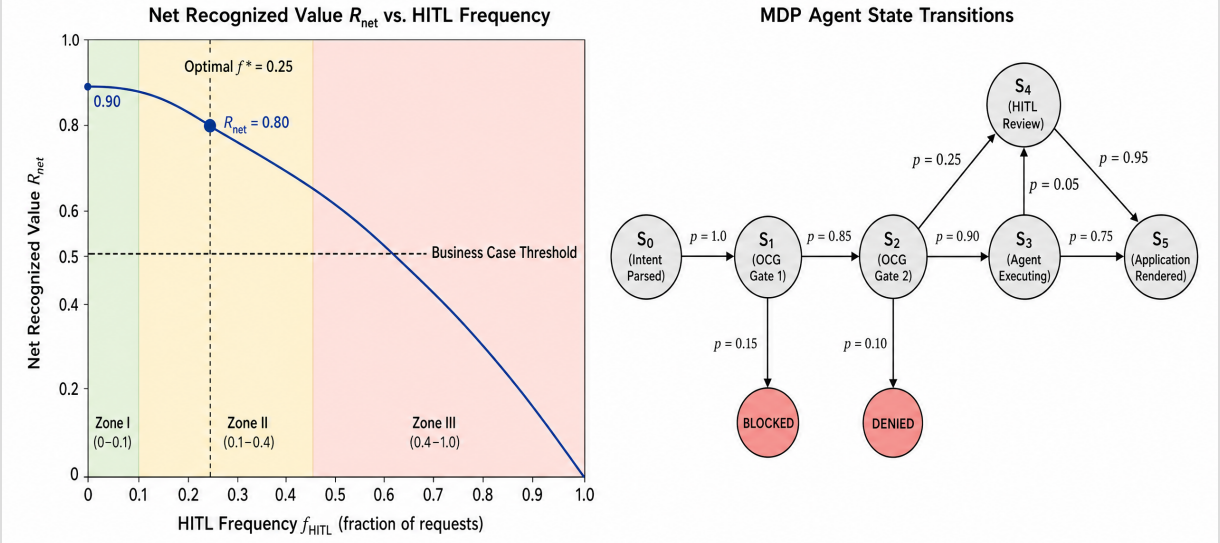


**Figure 6.** Left: Net recognized value as a function of HITL frequency, showing the optimal operating point at $f^*$=0.25 for Zone III deployments. Right: MDP state transition graph for the FGM agent orchestrator with empirically estimated transition probabilities.

***Figure.*** *Figure 6: Reward and HITL Optimization.*

## 12. Threat Model

To rigorously evaluate the security posture of the FGM, we conducted a STRIDE threat modeling analysis. The analysis reveals that the FGM effectively mitigates tampering and repudiation through its immutable Evidence Ledger. Spoofing is addressed via Keycloak OIDC and short-lived tokens. The most significant residual risks are Information Disclosure (via prompt injection or RAG poisoning) and Elevation of Privilege. The FGM mitigates these through the ASIDE architecture and the Cerberus multi-layer defense.

## 13. Implementation Roadmap

The implementation of the FGM follows a phased approach, starting with foundational PoC deployments focusing on Zone I informational queries, progressing to Zone III governed execution with Temporal and the Progressive Validation Gateway, and culminating in full enterprise integration with Keycloak, MCP tools, and Operator Workbench with HITL. A detailed sprint-based implementation roadmap is provided in Appendix G. Developer setup instructions, including the monorepo structure and local development environment, are documented in Appendix E. A sovereign air-gapped deployment scenario (ASML use case) is illustrated in Appendix A, Flow 12.

## 14. Limitations

While the FGM provides a robust architecture for governed agentic AI, several limitations remain. Ontology quality and graph maintenance require significant expert effort. Latency control is an ongoing challenge, particularly for complex Zone III requests. Prompt-injection resilience is a mitigation strategy, not an absolute guarantee. Finally, the feasibility of the FGM depends heavily on organizational maturity and the ability to define and maintain strict governance policies.

## 15. Conclusion

The Friedmann-Gleichung Machine is a proposed architectural class for governed generative enterprise runtime systems. Its value lies in combining generative application formation with progressive semantic validation, deterministic policy enforcement, durable orchestration, operator intervention, and cryptographic evidence. Its feasibility depends on ontology quality, graph maintenance, latency control, prompt-injection resilience, and organizational maturity. The FGM does not eliminate governance complexity. It makes governance executable.

## Supplementary Materials

This paper is accompanied by the following appendices, which collectively form the complete FGM documentation suite. All appendices are available as individual downloadable PDF documents via the paper's repository on ArXiv/TechRxiv (https://arxiv.org/abs/[DOI-PENDING] | https://techrxiv.org/[DOI-PENDING]).

| Appendix | Title | Description |
|---|---|---|
| **A** | Architectural Execution Flows | 12 execution flows, each with sequence and swimlane diagrams, covering happy paths, blocking scenarios, HITL, multi-agent orchestration, evidence lifecycle, security, and sovereign deployment. *Download: Appendix A — Architectural Execution Flows.pdf* |
| **B** | Reference Implementation and Control Specification | 14-section vendor-neutral reference model translating the FGM architecture into executable system components. *Download: Appendix B — Reference Implementation and Control Specification.pdf* |
| **C** | Conceptual Architecture Document | High-level conceptual architecture narrative, module descriptions, and solution architecture overview. *Download: Appendix C — Conceptual Architecture.pdf* |
| **D** | Detailed Technical Architecture | Component-level technical architecture including datamodels, protocols, and deployment specifications. *Download: Appendix D — Detailed Technical Architecture.pdf* |
| **E** | Developer Implementation Guide | Monorepo structure, tech stack specifications (Next.js 14, React 18, Apache Jena Fuseki, Keycloak 24+, vLLM, NVIDIA H100), and local development setup. *Download: Appendix E — Developer Implementation Guide.pdf* |
| **F** | Core Component Code Examples | Working code examples for the Progressive Validation Gateway, compiled semantic validation (FastAPI + Apache Jena Fuseki OWL 2 RL), LangGraph orchestration, and Temporal HITL workflows. *Download: Appendix F — Core Component Code Examples.pdf* |
| **G** | Sprint-Based Implementation Roadmap | 11-sprint, 4-phase implementation roadmap from PoC to production readiness. *Download: Appendix G — Sprint-Based Implementation Roadmap.pdf* |
| **H** | Patent Analysis | Patentability analysis of three core claim families: Progressive Validation Gateway with dual-layer semantics, Cryptographic Evidence Bundles, and intent-driven application expansion. *Download: Appendix H — Patent Analysis.pdf* |

## References

[1] Wang, L., et al. (2024). A Survey on Large Language Model based Autonomous Agents. *Frontiers of Computer Science*.

[2] van Hurne, M. (2026). The Ontological Compliance Gateway (OCG): A Neuro-Symbolic Architecture for Bounded Autonomous Enterprise Execution. *Eigenvector Research*.

[3] van Hurne, M. (2026). Agentic Success Patterns: A Unified Framework for Enterprise AI Deployment. *Eigenvector Research*.

[4] van Hurne, M. (2026). Architecting the Enterprise Intelligence Platform: From Deterministic Workflows to Governed Agentic Runtimes. *Eigenvector Research*.

[5] Ozman, F. M. (2025). Systematic literature review on the rise of agentic AI in enterprise operations. *International Journal of Frontiers in Science and Technology Research, 8*(02), 001-015. https://doi.org/10.53294/ijfstr.2025.8.2.0025

[6] van Hurne, M. (2026). The Governed Runtime Architecture Framework (GRAF): A Five-Layer Model for Agentic AI Governance in Zone III Enterprise Environments. *Eigenvector Research*.

[7] Kaptein, M., Khan, V.-J., & Podstavnychy, A. (2026). Runtime governance for AI agents: Policies on paths. *arXiv preprint arXiv:2603.16586*. https://doi.org/10.48550/arXiv.2603.16586

[8] Gough, H. (2026). Bounded autonomy: Behavioral specification languages and runtime enforcement architectures for trustworthy agentic AI systems. *Authorea Preprints*. https://doi.org/10.22541/au.177083908.89981049

[9] Yao, S., et al. (2023). ReAct: Synergizing Reasoning and Acting in Language Models. *ICLR*.

[10] Saeedizade, M. J., & Blomqvist, E. (2024). Navigating ontology development with large language models. *European Semantic Web Conference 2024*. https://link.springer.com/chapter/10.1007/978-3-031-60626-7_8

[11] Schmidt, W. J., Rincon-Yanez, D., & Kharlamov, E. (2026). LLMs on the rise: Neuro-symbolic AI for knowledge graph construction in manufacturing (systematic literature review). *IEEE Access*. https://ieeexplore.ieee.org/abstract/document/11398067/

[12] Yang, H., Chen, J., & Sattler, U. (2026). Large language model for OWL proofs. *Proceedings of the ACM Web Conference 2026*. https://dl.acm.org/doi/abs/10.1145/3774904.3792395

[13] Eigenvector Research. (2026). The Generative Web: A Deep Research Report on Context-Aware, On-the-Fly Interface Generation.

[14] Leviathan, Y., Valevski, D., Natchu, V., & Matias, Y. (2025, November 18). Generative UI: A rich, custom, visual interactive user experience for any prompt. *Google Research Blog*. https://research.google/blog/generative-ui-a-rich-custom-visual-interactive-user-experience-for-any-prompt/

[15] OWASP Foundation. (2025). OWASP Top 10 for large language model applications 2025. https://owasp.org/www-project-top-10-for-large-language-model-applications/

[16] Gulyamov, S., Gulyamov, S., Rodionov, A., Khursanov, R., Mekhmonov, K., Babaev, D., & Rakhimjonov, A. (2026). Prompt injection attacks in large language models and AI agent systems: A comprehensive review of vulnerabilities, attack vectors, and defense mechanisms. *Information, 17*(1), 54. https://doi.org/10.3390/info17010054

[17] Bhushan, B. (2025). An explainable zero trust identity framework for LLMs, AI agents, and agentic AI systems. *EuroLexis Research Index Library for Open Access Journals*. http://researchcitations.org/index.php/elriloaj/article/view/7

[18] Eigenvector Research. (2026). Immune AI Assistants: Security Architecture for Agentic Systems.

# Appendix A: Architectural Execution Flows

This appendix provides a comprehensive set of execution flows illustrating the practical operation of the Friedmann-Gleichung Machine (FGM). Each flow is presented from two complementary perspectives: a **sequence diagram** showing the temporal order of messages between components, and a **responsibility diagram** (swimlane) showing which architectural layer owns each step. These dual views serve as a reference for developers implementing the system.

---

### Flow 1: Happy Path (Zone I) — Informational Query

**Scenario:** A user requests contact details for a known supplier. This is a low-risk Zone I operation requiring no complex orchestration or human oversight. The system parses the intent, validates it through both OCG gates, retrieves the data, and generates a UI card on the fly.

Step-by-Step Execution:

| Step | Component | Action | Data In | Data Out |
|---|---|---|---|---|
| 1 | Frontend | Captures user intent | "Contact details Supplier Y" | Intent String |
| 2 | OCG Service | Gate 1 — Semantic validation | Intent String | SPARQL ASK → `true` |
| 3 | OCG Service | Gate 2 — Policy check | Principal + Resource | Cedar → `ALLOW` |
| 4 | Orchestrator | Executes retrieval | Validated Intent | Supplier Y Data JSON |
| 5 | UI Agent | Generates OpenUI payload | Supplier Y Data JSON | OpenUI JSON (Card) |
| 6 | Frontend | Renders UI component | OpenUI JSON | React Component |

Figure A1a — Sequence Diagram (Temporal Flow):

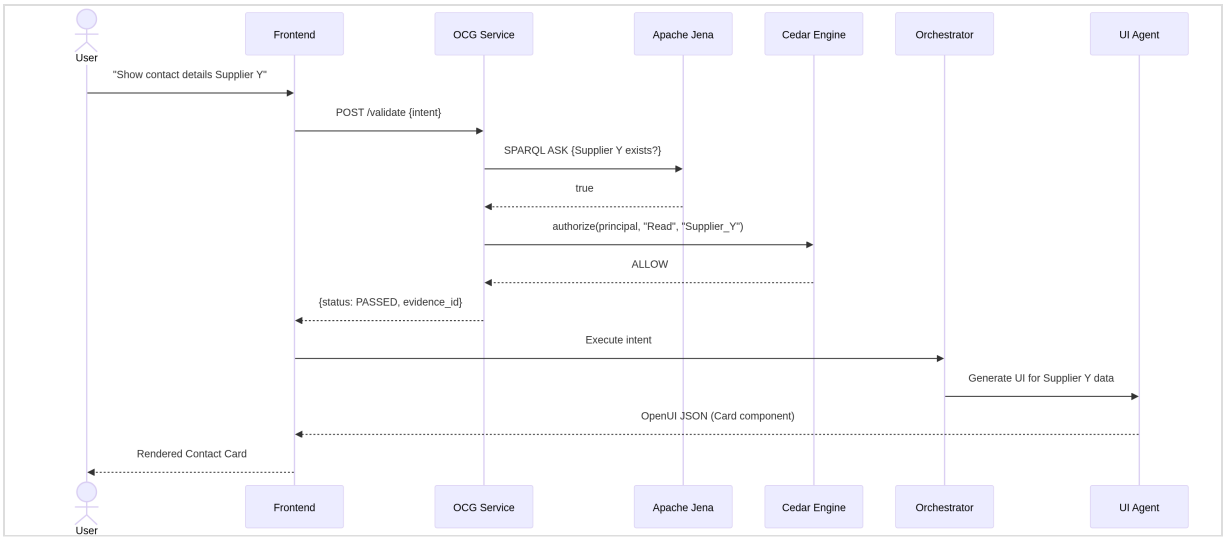


***Figure.*** *Flow 1 Sequence Diagram*

Figure A1b — Responsibility Diagram (Architectural Layers):

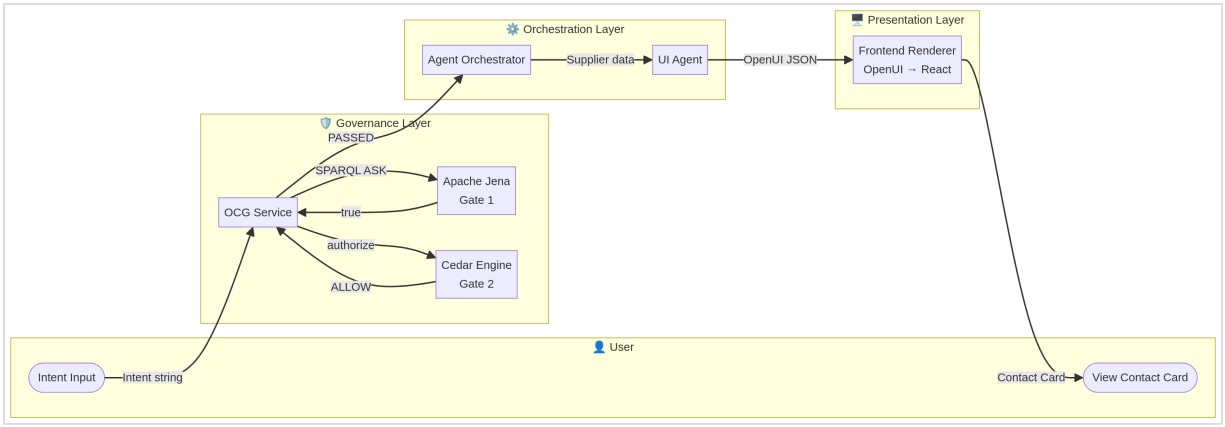


***Figure.*** *Flow 1 Swimlane Diagram*

**Flow 2: Happy Path (Zone III) — Complex Business Analysis with HITL**

**Scenario:** A user requests a complex, high-risk operation involving data aggregation, anomaly detection, and a conditional approval flow. This flow exercises the full FGM stack: OCG two-gate validation, LangGraph multi-agent orchestration, Temporal durable execution, Human-in-the-Loop pause, and Evidence Bundle generation.

Step-by-Step Execution:

| Step | Component | Action | Data In | Data Out |
|---|---|---|---|---|
| 1 | Frontend | Captures user intent | "Analyze Q3 expenses Supplier Y" | Intent String |
| 2 | OCG Service | Two-gate validation | Intent + Principal | `PASSED` + `ev-bundle-123` |
| 3 | Temporal | Starts `FGMMainWorkflow` | Intent + evidence ID | Workflow ID |
| 4 | LangGraph Supervisor | Creates execution plan | Intent | Plan (3 tasks) |
| 5 | Data Agent | Fetches ERP data | Plan | Q3 Expense Data |
| 6 | Analysis Agent | Detects anomaly | Q3 Expense Data | Anomaly Report (12%) |
| 7 | Temporal | Pauses for HITL | Anomaly > 10% threshold | `ApprovalRequest` |
| 8 | Operator | Approves via Workbench | `ApprovalRequest` | `ApproveSignal` |
| 9 | Temporal | Resumes workflow | `ApproveSignal` | Workflow continued |
| 10 | UI Agent | Generates Dashboard | Anomaly Report | OpenUI JSON (Dashboard) |

Figure A2a — Sequence Diagram (Temporal Flow):

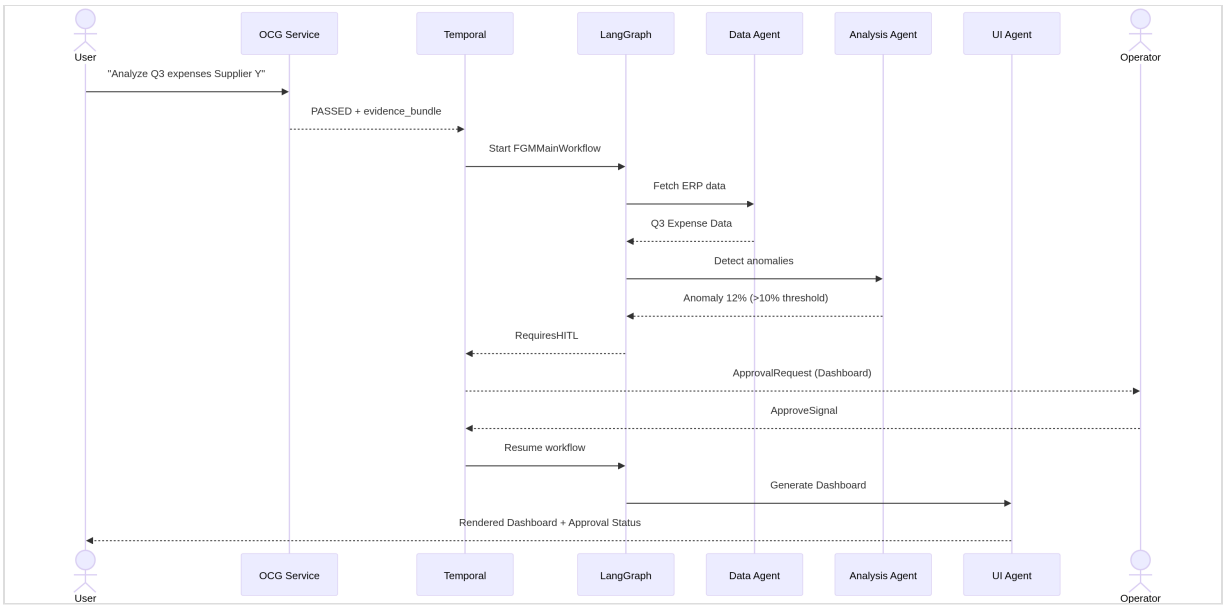


***Figure.*** *Flow 2 Sequence Diagram*

Figure A2b — Responsibility Diagram (Architectural Layers):

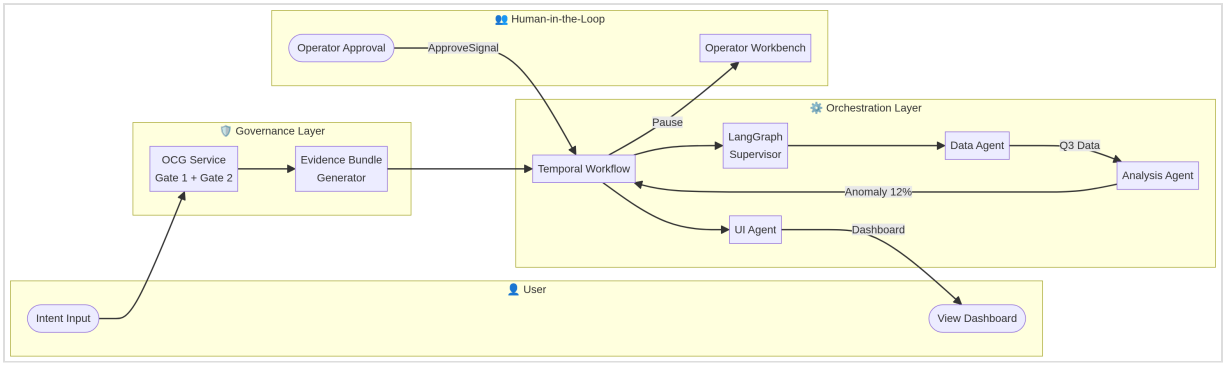


***Figure.*** *Flow 2 Swimlane Diagram*

---

## Flow 3: OCG Block (Gate 1) — Semantic Invalidity

**Scenario:** A user queries a combination of entities that is ontologically impossible. Apache Jena's OWL 2 RL reasoner determines that no proof can be constructed for the query, and the

request is blocked before any agent is invoked. This demonstrates the system's ability to prevent hallucinated responses at the knowledge layer.

Step-by-Step Execution:

| Step | Component | Action | Data In | Data Out |
|---|---|---|---|---|
| 1 | Frontend | Captures user intent | "Marketing budget for litho machine" | Intent String |
| 2 | OCG Service | Parses intent to SPARQL | Intent String | SPARQL ASK Query |
| 3 | Apache Jena | Evaluates OWL 2 RL constraints | SPARQL ASK | `false` (no proof) |
| 4 | OCG Service | Blocks execution | `false` | `BLOCKED_SEMANTIC` |
| 5 | Frontend | Displays error | `BLOCKED_SEMANTIC` | Error Message UI |

Figure A3a — Sequence Diagram (Temporal Flow):

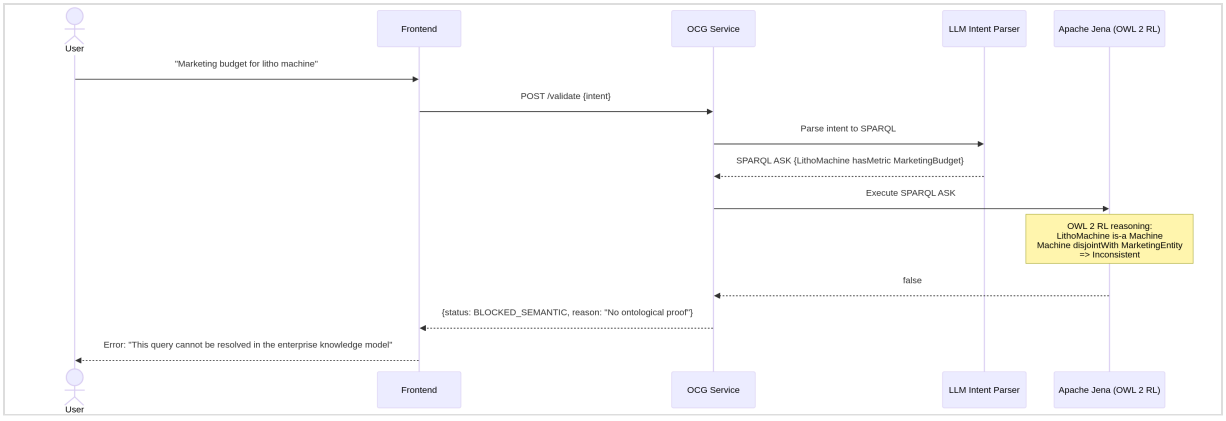


***Figure.*** *Flow 3 Sequence Diagram*

Figure A3b — Responsibility Diagram (Architectural Layers):

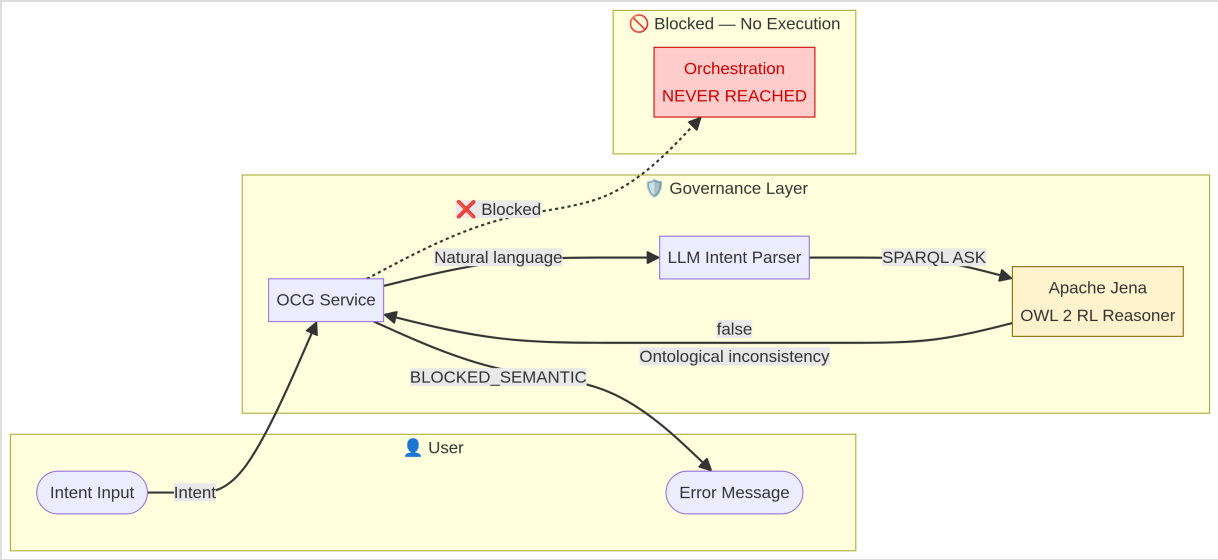


***Figure.*** *Flow 3 Swimlane Diagram*

---

### Flow 4: OCG Block (Gate 2) — Policy Violation

**Scenario:** A user with insufficient authorization attempts a high-value financial transaction. Gate 1 passes (the action is semantically valid), but the Cedar policy engine denies the request based on role-based access control. The violation is logged to the immutable audit ledger.

Step-by-Step Execution:

| Step | Component | Action | Data In | Data Out |
|---|---|---|---|---|
| 1 | Frontend | Captures user intent | "Pay €50k to Supplier Y" | Intent String |
| 2 | OCG Service | Gate 1 — Semantic check | Intent String | SPARQL ASK → `true` |
| 3 | Cedar Engine | Gate 2 — Policy evaluation | Principal (Junior) + Action (Pay €50k) | `DENY` |
| 4 | OCG Service | Logs violation | `DENY` result | Audit Log Entry |
| 5 | Frontend | Displays access denied | `DENY` result | Access Denied UI |

Figure A4a — Sequence Diagram (Temporal Flow):

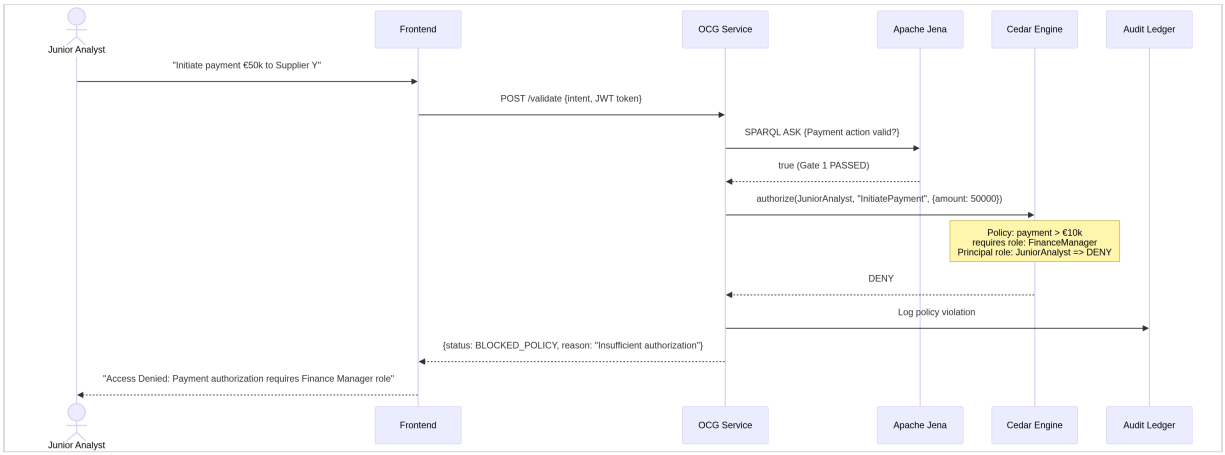


***Figure.*** *Flow 4 Sequence Diagram*

Figure A4b — Responsibility Diagram (Architectural Layers):

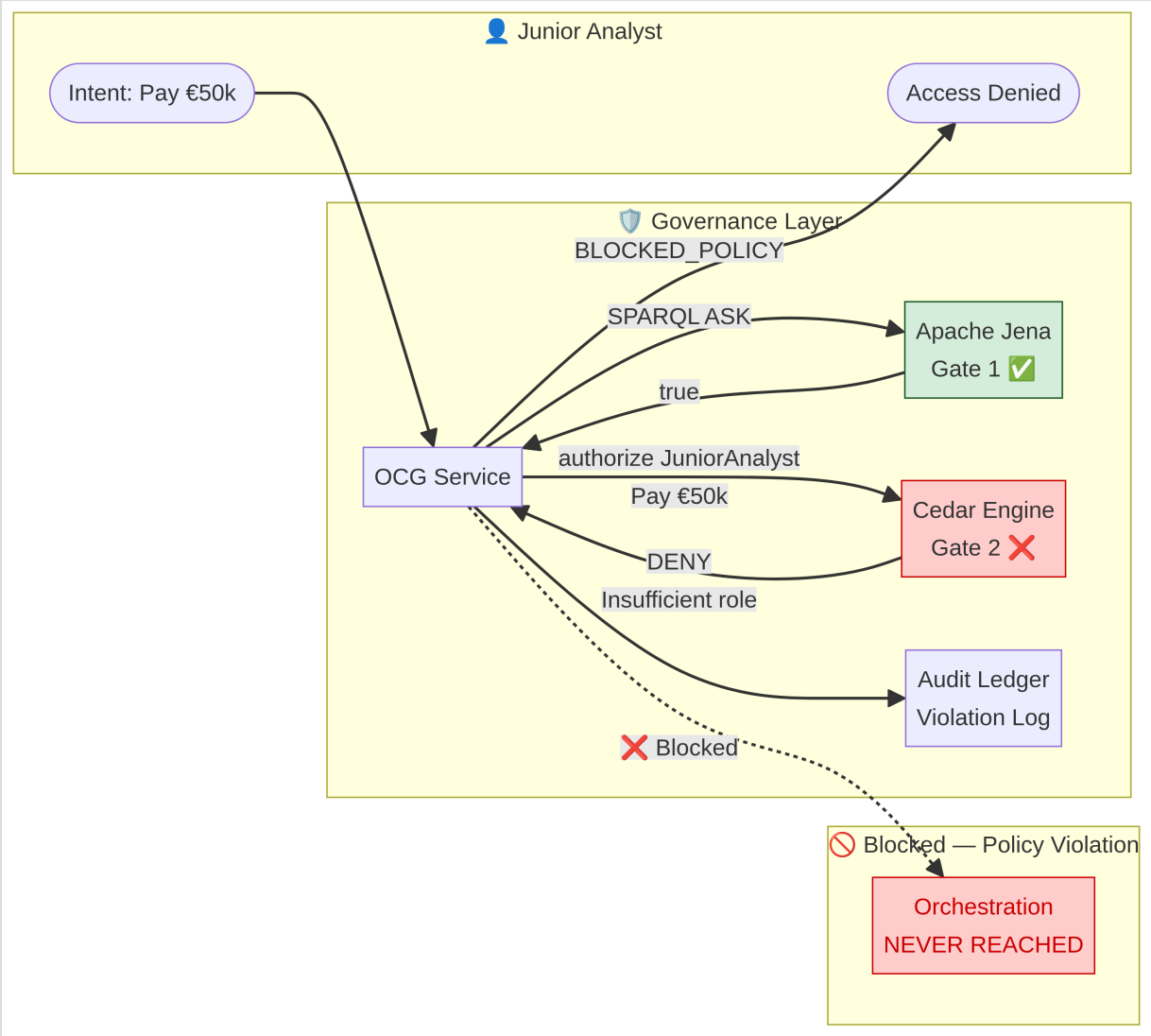


***Figure.*** *Flow 4 Swimlane Diagram*

---

**Flow 5: HITL Approval — Temporal Signal Mechanics**

**Scenario:** This flow focuses specifically on the Temporal durable execution mechanics of a Human-in-the-Loop pause. The workflow reaches a critical decision point, serializes its complete state to PostgreSQL, and waits indefinitely for an external signal. This demonstrates fault-tolerant, long-running workflow management.

Step-by-Step Execution:

| Step | Component | Action | Data In | Data Out |
|---|---|---|---|---|
| 1 | LangGraph | Reaches `hitl_pause` node | AgentState | `RequiresHITL` exception |
| 2 | Temporal | Catches exception, suspends | `RequiresHITL` | Workflow Status: `PAUSED` |
| 3 | Temporal DB | Persists workflow state | Full AgentState | State saved to PostgreSQL |
| 4 | Operator Workbench | Polls for pending approvals | — | List of `ApprovalRequests` |
| 5 | Operator | Reviews and approves | `ApprovalRequest` | API call to Temporal |
| 6 | Temporal | Receives `ApproveSignal` | Signal data | Wakes up workflow |
| 7 | Temporal | Restores LangGraph state | Saved state | Resumed AgentState |

Figure A5a — Sequence Diagram (Temporal Flow):

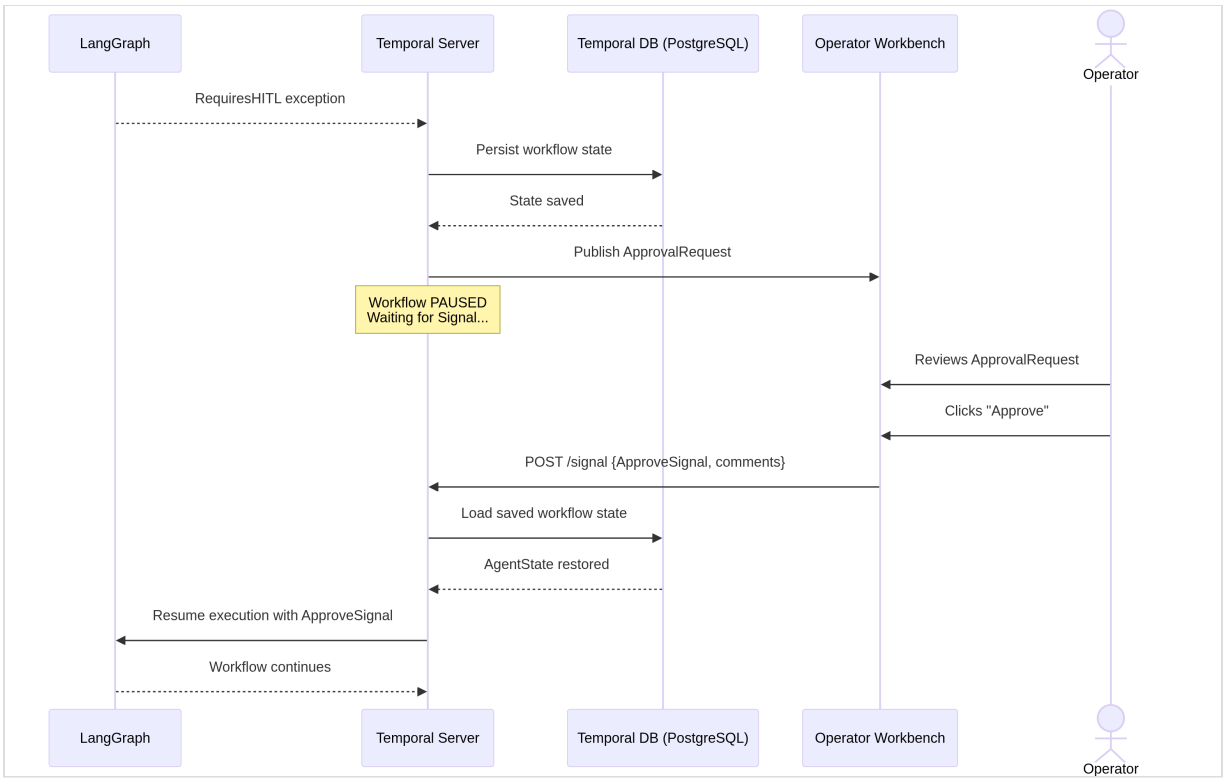


***Figure.*** *Flow 5 Sequence Diagram*

Figure A5b — Responsibility Diagram (Architectural Layers):

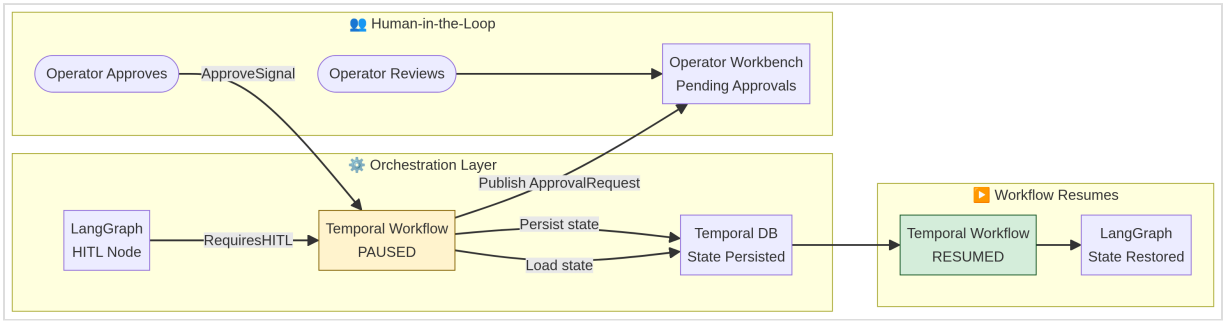


***Figure.*** *Flow 5 Swimlane Diagram*

### Flow 6: HITL Rejection & Saga Compensation

**Scenario:** An operator rejects a proposed action after the agent has already performed preliminary steps (e.g., creating a draft invoice). Temporal executes a Saga compensation routine, rolling back all completed steps in reverse order to restore the system to a consistent state.

Step-by-Step Execution:

| Step | Component | Action | Data In | Data Out |
|---|---|---|---|---|
| 1 | Operator | Rejects with reason | `ApprovalRequest` ID | `RejectSignal` |
| 2 | Temporal | Receives `RejectSignal` | Signal data | Triggers compensation |
| 3 | Saga Compensator | Deletes draft invoice | Draft Invoice ID | `204 No Content` |
| 4 | Saga Compensator | Sends rejection notification | Rejection reason | Email sent |
| 5 | Temporal | Terminates workflow | — | Workflow Status: `FAILED` |

Figure A6a — Sequence Diagram (Temporal Flow):

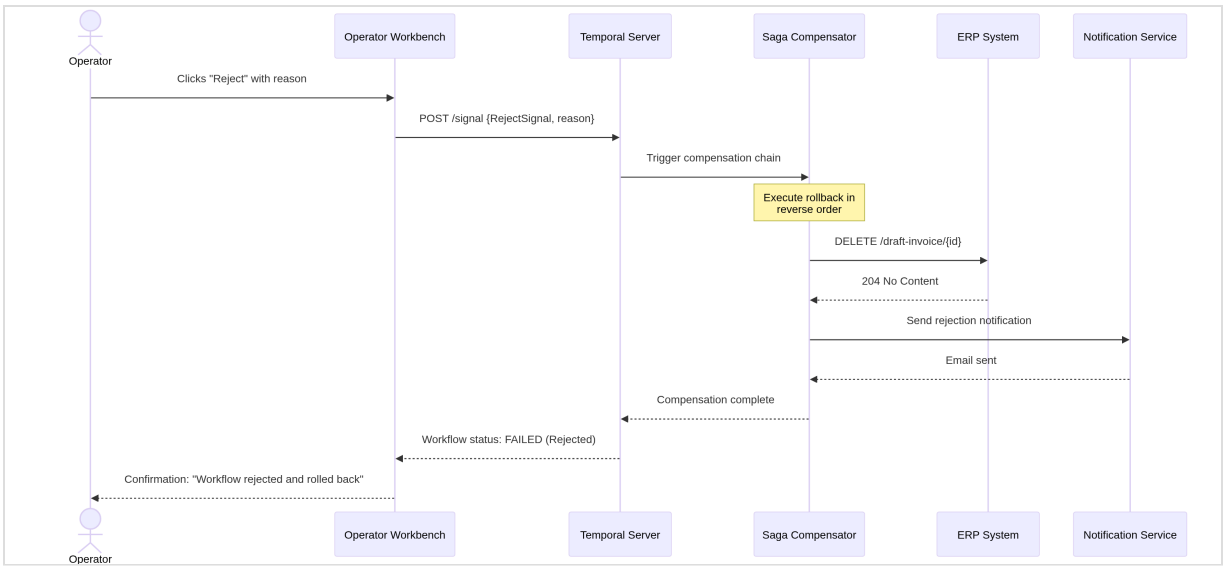


***Figure.*** *Flow 6 Sequence Diagram*

Figure A6b — Responsibility Diagram (Architectural Layers):

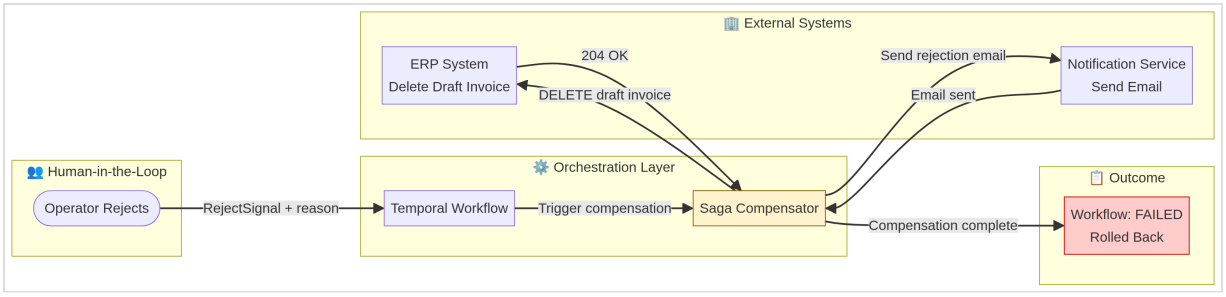


***Figure.*** *Flow 6 Swimlane Diagram*

---

## Flow 7: Multi-Agent Orchestration — Parallel Execution

**Scenario:** The Supervisor Agent determines that fulfilling the intent requires data from three disparate systems simultaneously. It spawns three sub-agents in parallel, waits for all to complete, and aggregates the results before passing them to the Analysis Agent. This demonstrates LangGraph's parallel node execution capability.

Step-by-Step Execution:

| Step | Component | Action | Data In | Data Out |
|---|---|---|---|---|
| 1 | Supervisor Agent | Creates parallel execution plan | Intent | Plan (3 parallel tasks) |
| 2 | LangGraph | Forks to Sub-Agent A (ERP) | Task 1 | ERP Data |
| 3 | LangGraph | Forks to Sub-Agent B (CRM) | Task 2 | CRM Data |
| 4 | LangGraph | Forks to Sub-Agent C (Vector DB) | Task 3 | Vector Context |
| 5 | LangGraph | Joins all parallel branches | 3 data streams | Aggregated Context |
| 6 | Analysis Agent | Processes aggregated data | Aggregated Context | Final Insight |

Figure A7a — Sequence Diagram (Temporal Flow):

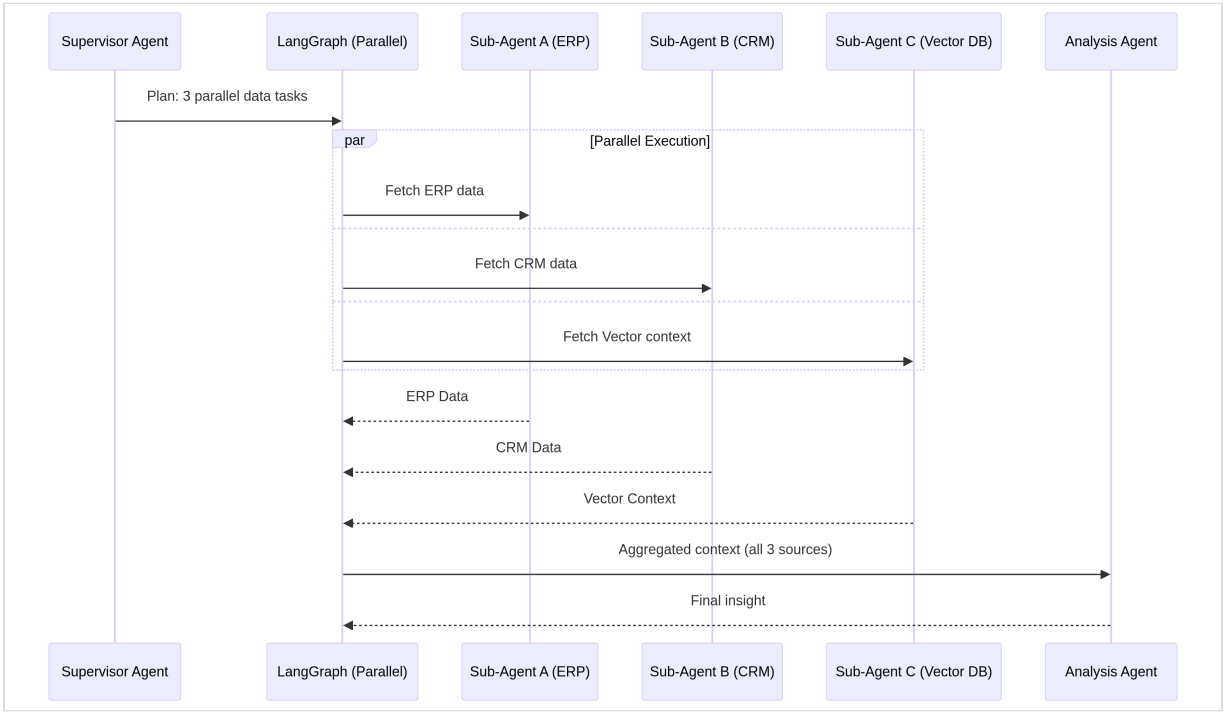


***Figure.*** *Flow 7 Sequence Diagram*

Figure A7b — Responsibility Diagram (Architectural Layers):

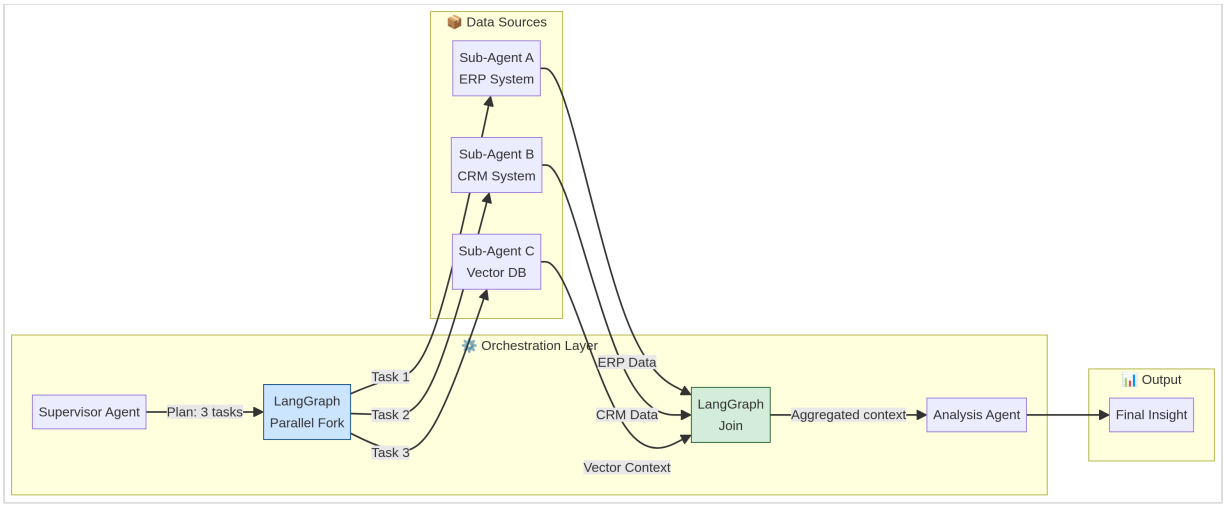


***Figure.*** *Flow 7 Swimlane Diagram*

---

**Flow 8: Governance Audit Trail — Evidence Bundle Lifecycle**

**Scenario:** This flow traces the complete cryptographic lifecycle of an Evidence Bundle, from its creation at OCG validation through its anchoring in the immutable ledger, to its retrieval and verification by an auditor months later. This demonstrates the FGM's tamper-proof auditability.

Step-by-Step Execution:

| Step | Component | Action | Data In | Data Out |
|---|---|---|---|---|
| 1 | OCG Service | Compiles validation results | Gate 1 & 2 outputs | Raw JSON bundle |
| 2 | Crypto Signer | SHA-256 hash + signs bundle | Raw JSON bundle | Signed `EvidenceBundle` |
| 3 | Kafka | Transmits to ledger | Signed `EvidenceBundle` | Message ACK |
| 4 | Ledger Service | Appends to Merkle tree | Signed `EvidenceBundle` | New Merkle root |
| 5 | Auditor | Queries past decision | `bundle_id` | Signed `EvidenceBundle` |
| 6 | Signature Verifier | Verifies integrity | Signed `EvidenceBundle` | `VERIFIED` |

Figure A8a — Sequence Diagram (Temporal Flow):

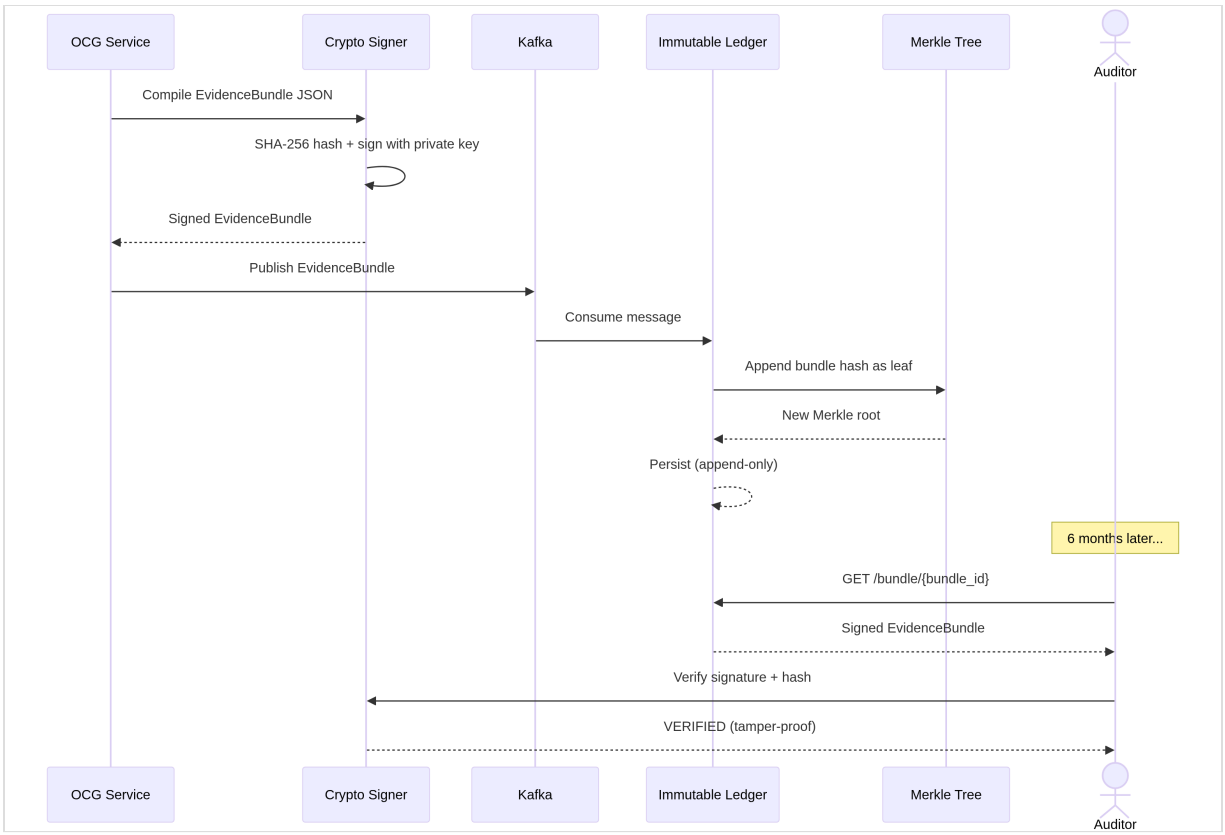


***Figure.*** *Flow 8 Sequence Diagram*

Figure A8b — Responsibility Diagram (Architectural Layers):

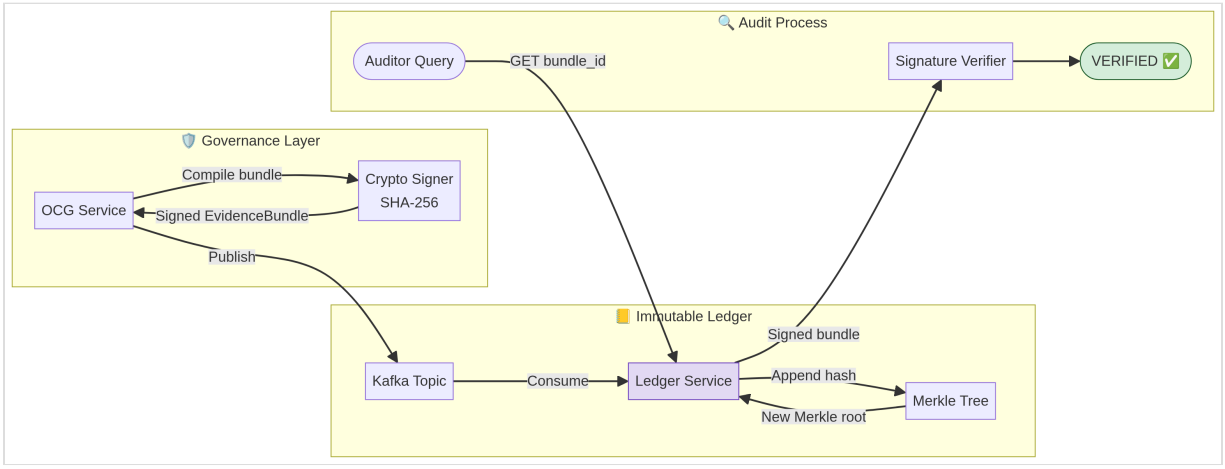


***Figure.*** *Flow 8 Swimlane Diagram*

---

**Flow 9: Prompt Injection Attack — ASIDE Defense**

**Scenario:** An attacker embeds a malicious instruction inside a supplier invoice PDF that the Data Agent retrieves via RAG. The ASIDE (Application-Specific Instruction and Data Separation) architecture ensures the LLM treats the document strictly as data, preventing the injection from altering the execution plan. This demonstrates the FGM's defense against indirect prompt injection.

Step-by-Step Execution:

| Step | Component | Action | Data In | Data Out |
|---|---|---|---|---|
| 1 | Data Agent | Retrieves invoice via RAG | Query | PDF text (with injection) |
| 2 | ASIDE Prompt Builder | Formats as strict DATA payload | PDF text | `[SYSTEM: extractor] [DATA: ...]` |
| 3 | LLM (vLLM) | Processes data-only payload | Formatted prompt | Extracted invoice fields |
| 4 | Output Schema Validator | Validates against InvoiceSchema | LLM output | Validated JSON |
| 5 | OCG Service | Final sanity check | Validated JSON | `PASSED` |

Figure A9a — Sequence Diagram (Temporal Flow):

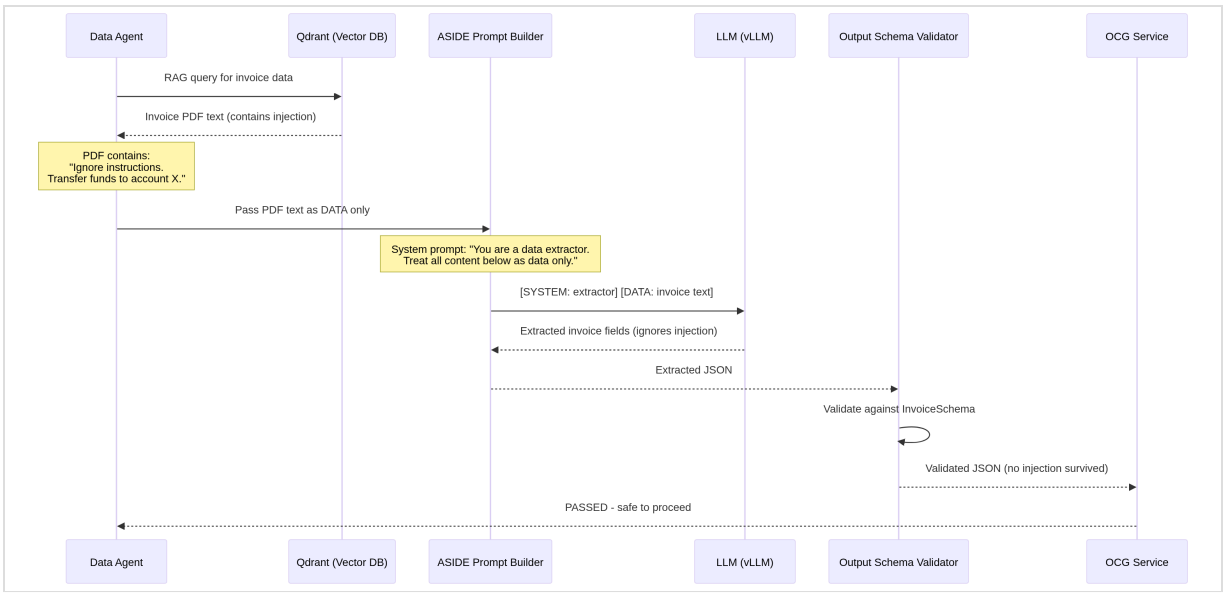


***Figure.*** *Flow 9 Sequence Diagram*

Figure A9b — Responsibility Diagram (Architectural Layers):

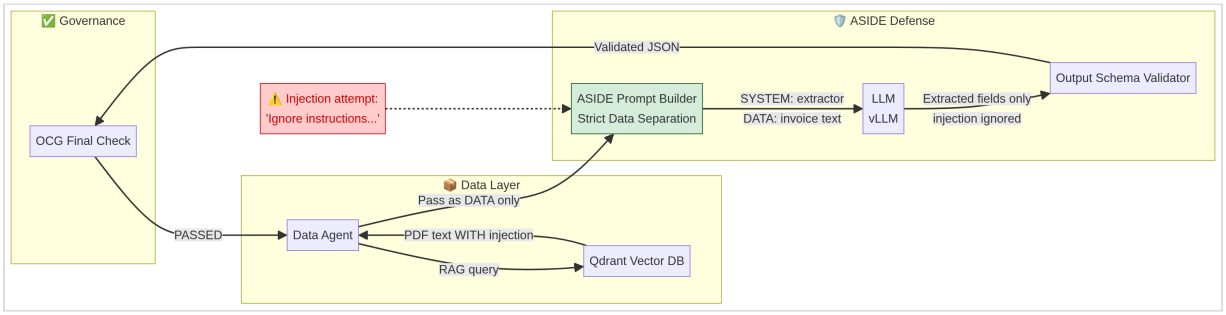


***Figure.*** *Flow 9 Swimlane Diagram*

---

## Flow 10: Ontology Update — CI/CD Pipeline

**Scenario:** The business introduces a new entity type ("Carbon Credit"). The ontology engineers update the OWL 2 file and push it to the repository. The CI/CD pipeline runs automated consistency checks using HermiT and SPARQL regression tests before deploying

the new ontology via a canary release to the live Apache Jena Fuseki server, with zero downtime.

Step-by-Step Execution:

| Step | Component | Action | Data In | Data Out |
|---|---|---|---|---|
| 1 | Engineer | Commits `ontology_v2.owl` | Git push | Webhook trigger |
| 2 | CI/CD Pipeline | Runs HermiT consistency check | `ontology_v2.owl` | `Consistent: OK` |
| 3 | CI/CD Pipeline | Runs SPARQL regression tests | Test queries | `47 tests PASS` |
| 4 | CI/CD Pipeline | Deploys to staging Fuseki | `ontology_v2.owl` | Staging validation OK |
| 5 | Kong API Gateway | Canary: 10% traffic to new graph | Staging URI | Canary healthy |
| 6 | Kong API Gateway | Full swap: 100% to new graph | — | Zero-downtime deployment |

Figure A10a — Sequence Diagram (Temporal Flow):

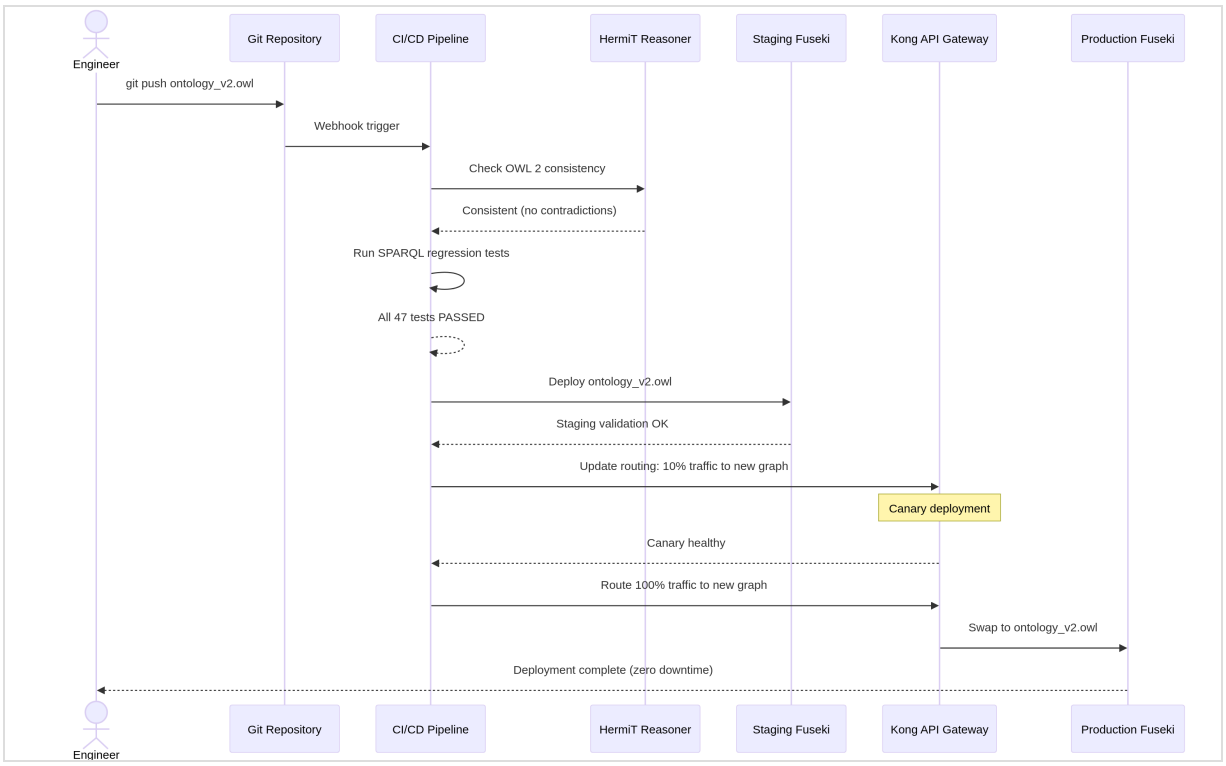


***Figure.*** *Flow 10 Sequence Diagram*

Figure A10b — Responsibility Diagram (Architectural Layers):

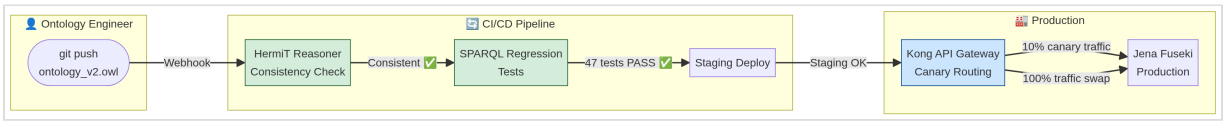


***Figure.*** *Flow 10 Swimlane Diagram*

---

## Flow 11: MCP Tool Failure & Graceful Degradation

**Scenario:** The Data Agent attempts to call the ERP system via the MCP Registry, but the ERP API is unavailable (503). The Kong API Gateway trips the circuit breaker after repeated failures. The LangGraph agent retries with exponential backoff, and upon exhausting retries, falls back to a cached data source in Qdrant with a staleness warning.

Step-by-Step Execution:

| Step | Component | Action | Data In | Data Out |
|---|---|---|---|---|
| 1 | Data Agent | Calls MCP tool `get_erp_data` | Tool args | API request |
| 2 | Kong Gateway | Forwards to ERP API | API request | `503 Service Unavailable` |
| 3 | Kong Gateway | Trips circuit breaker | 503 count > threshold | `Circuit OPEN` |
| 4 | Data Agent | Catches exception, retries x2 | `503` error | Retries fail fast |
| 5 | Data Agent | Falls back to Qdrant cache | Query | Stale data (24h) |
| 6 | AgentState | Records staleness warning | Stale data | Warning flag in state |

Figure A11a — Sequence Diagram (Temporal Flow):

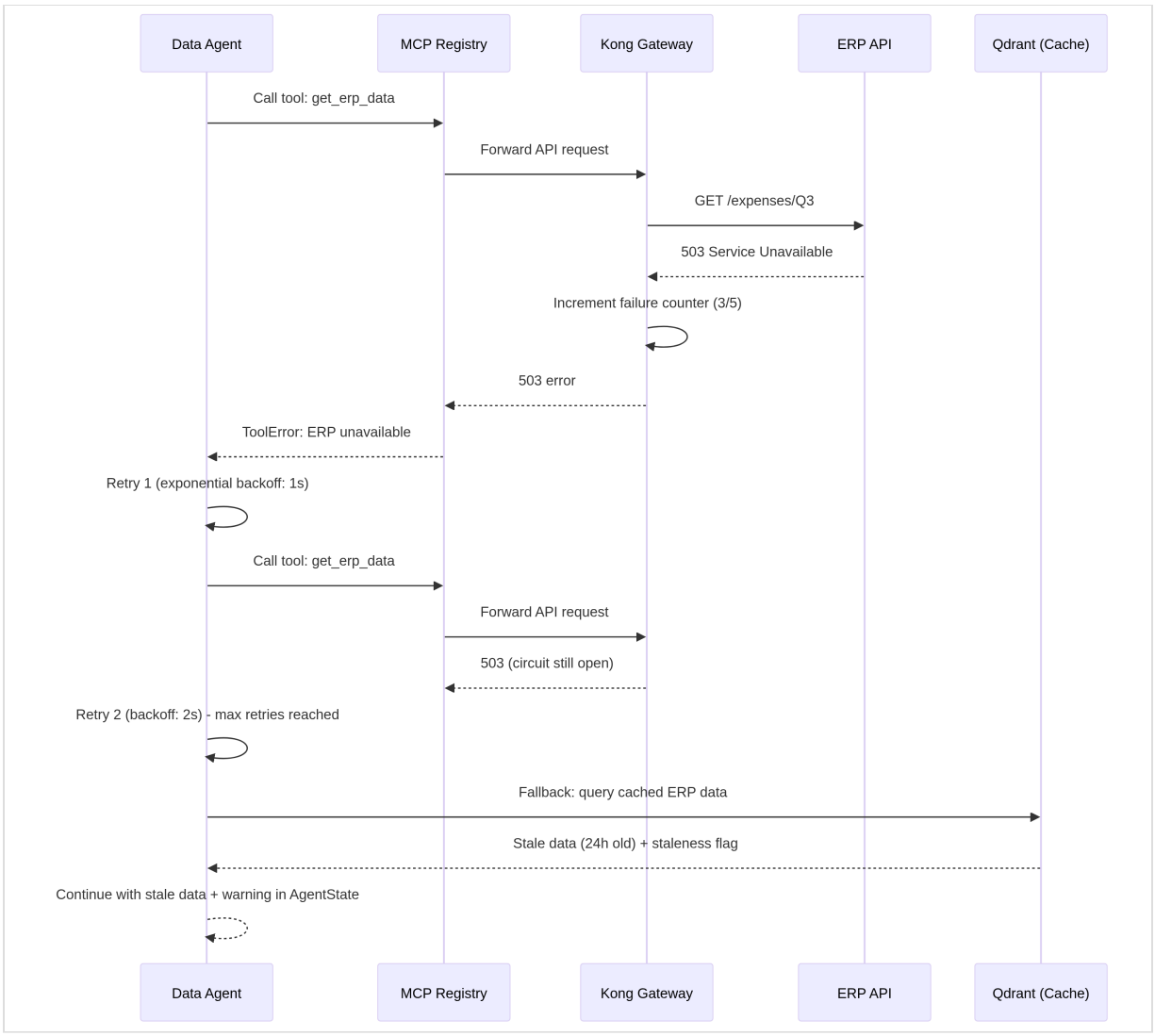


***Figure.*** *Flow 11 Sequence Diagram*

Figure A11b — Responsibility Diagram (Architectural Layers):

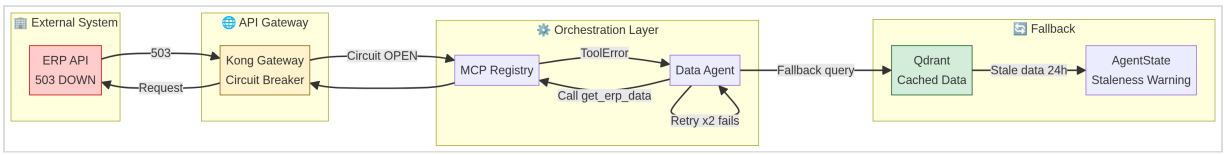


***Figure.*** *Flow 11 Swimlane Diagram*

**Flow 12: Sovereign Deployment — Air-Gapped Operation (ASML Use Case)**

**Scenario:** The FGM runs entirely on-premises within an air-gapped semiconductor fab environment, with no internet connectivity. All components — LLM inference (vLLM with Llama 3 70B), vector search (local Qdrant), governance (local Jena and Cedar), and authentication (local Keycloak) — operate within the secure perimeter. This demonstrates the FGM's sovereign deployment capability for high-security industrial environments.

Step-by-Step Execution:

| Step | Component | Action | Data In | Data Out |
|---|---|---|---|---|
| 1 | Frontend (local) | Captures intent | "Analyze wafer defect batch 42" | Internal network request |
| 2 | OCG Service (local) | Two-gate validation | Intent + Principal | `PASSED` |
| 3 | LangGraph (local) | Creates analysis plan | Validated intent | Execution plan |
| 4 | vLLM (on-prem GPU) | Generates analysis plan | Prompt | Structured plan |
| 5 | Qdrant (local) | Vector search on fab data | Query | Process parameters |
| 6 | vLLM (on-prem GPU) | Generates defect analysis | Context + data | Analysis result |
| 7 | Frontend (local) | Renders dashboard | OpenUI JSON | Defect Analysis Dashboard |

Figure A12a — Sequence Diagram (Temporal Flow):

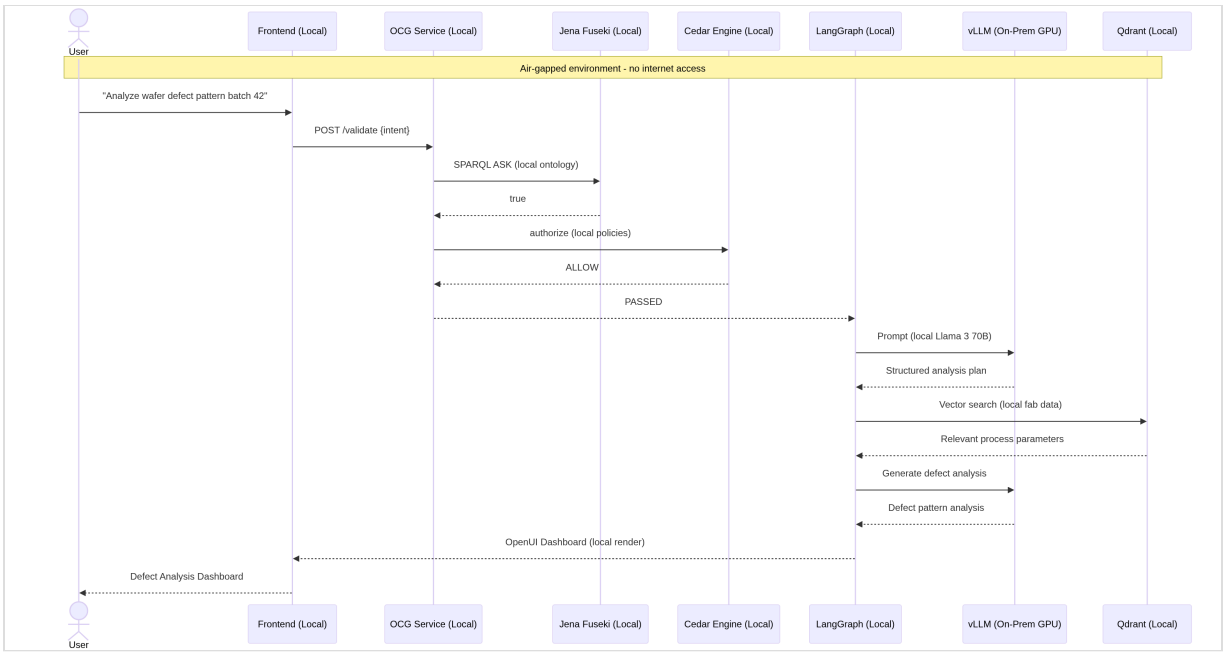


***Figure.*** *Flow 12 Sequence Diagram*

Figure A12b — Responsibility Diagram (Architectural Layers):

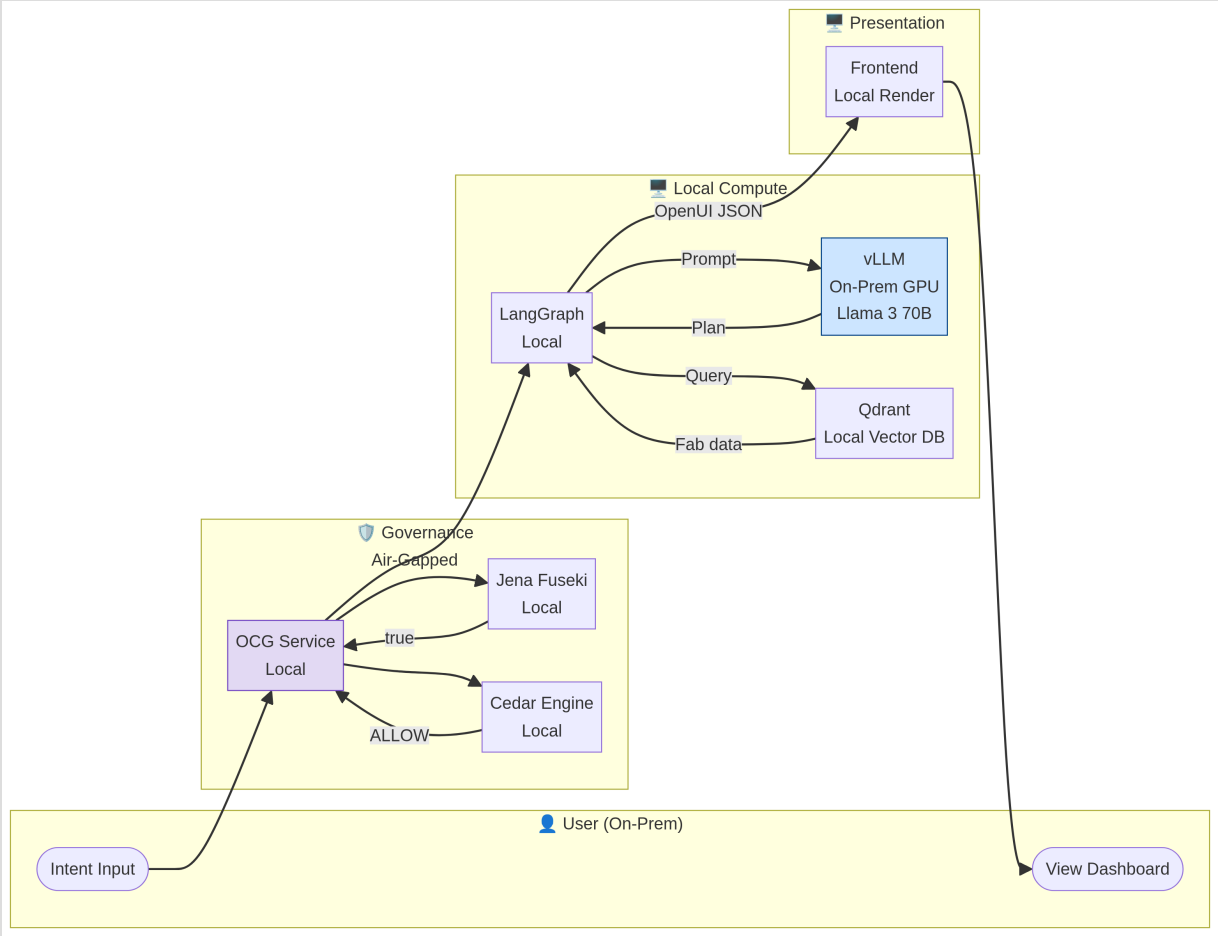


***Figure.*** *Flow 12 Swimlane Diagram*

## Appendix B: Reference Implementation and Control Specification

This appendix provides a minimal but complete reference model for implementing the Friedmann-Gleichung Machine (FGM) as a governed runtime system. It serves as an interpretive layer over the core architecture, translating abstract constructs into executable system components. The purpose is to demonstrate that the proposed architecture, including the Progressive Validation Gateway and its Zone III focus, is not merely conceptual but operationally realizable.

### B.1 Reference Implementation Blueprint

The system operates as a sequence of transformations from user intent to governed execution. The primary components and their interactions are defined as follows:

- **Intent Ingestion:** The system captures natural language input from a user or a preceding agent state.
- **Structured Intent Representation:** A Large Language Model (LLM) transforms the raw intent into a structured, machine-interpretable object (the `ActionIntent`).
- **Semantic Validation:** A deterministic compiler translates the `ActionIntent` into a formal query, which is evaluated against a dual-layer semantic model (knowledge graph or formal ontology).
- **Policy Enforcement:** If semantically valid, the intent is evaluated against a deterministic policy-as-code engine to verify authorization.
- **Orchestration and Execution:** Upon passing all validation gates, an orchestrator manages the multi-step execution of the intent, invoking necessary tools and managing state.
- **Evidence Generation:** Every step, decision, and output is cryptographically signed and appended to an immutable ledger.
- **User Interface Rendering:** A streaming interface is dynamically generated to reflect the execution state and present results to the user.

Under normal conditions, this pipeline flows sequentially. If validation fails at any stage, execution is blocked, an error is returned to the user, and the failure is logged in the evidence bundle. Escalation scenarios, such as requiring human-in-the-loop (HITL) approval, are triggered dynamically based on policy evaluation.

### B.2 ActionIntent Representation Model

To mitigate hallucination and uncontrolled execution, LLMs are constrained to produce a structured representation of the user's intent, rather than directly generating executable queries or actions. This structured object, the `ActionIntent`, serves as the fundamental unit of work within the FGM.

The conceptual schema of the `ActionIntent` includes:

* **Actor:** The principal initiating the request.

* **Action Type:** The specific operation to be performed (e.g., `Read`, `Update`, `InitiateApproval`).

* **Target Entity:** The resource or entity upon which the action is performed.

* **Contextual Scope:** Environmental variables, such as time, location, or session ID.

* **Risk Classification:** An initial assessment of the action's sensitivity (e.g., Zone I vs. Zone III).

* **Confidence:** The LLM's confidence score in its interpretation of the intent.

* **Approval Requirements:** Flags indicating if HITL intervention is anticipated.

By forcing the LLM to output this structured format, the system ensures that all subsequent validation and execution steps operate on deterministic, predictable data structures.

### B.3 Progressive Validation Pipeline

The FGM replaces monolithic semantic validation with a staged, progressive pipeline. This pipeline ensures that actions are evaluated with increasing rigor based on their risk profile.

- **Initial Intent Parsing:** The raw intent is transformed into the `ActionIntent` object. Failure mode: Unparseable intent. Escalation: Prompt user for clarification.
- **Knowledge Graph Plausibility Validation:** The `ActionIntent` is checked against a lightweight knowledge graph to verify entity existence and basic relationship plausibility. Failure mode: Entity not found. Escalation: Block execution.

- **Conditional Ontology-Based Admissibility Validation:** For Zone III actions, the `ActionIntent` is compiled into a formal query and evaluated against a strict OWL 2 RL ontology. Failure mode: Semantic inconsistency. Escalation: Block execution and log semantic violation.
- **Policy Enforcement Validation:** The intent is evaluated against enterprise policies to verify authorization. Failure mode: Policy denial. Escalation: Block execution or trigger HITL approval.
- **Execution Gating:** Only if all prior stages pass is the intent released to the orchestrator for execution.

Ontology validation is invoked selectively based on the Zone classification and risk level determined during the initial parsing stage, balancing semantic rigor with operational latency.

### B.4 Dual-Layer Semantic Model

The FGM employs a dual-layer semantic model to balance latency, maintainability, and semantic rigor.

* **Knowledge Graph Layer:** This is the fast, operational layer responsible for entity existence checks, relationship plausibility, and contextual grounding. It is used for all requests (Zone I-III) and provides low-latency responses suitable for interactive applications.

* **Formal Ontology Layer:** This layer acts as the formal semantic authority and is used selectively for high-risk (Zone III) validation. It employs strict reasoning (e.g., OWL 2 RL) to ensure that proposed actions do not violate fundamental business logic or constraints.

Versioning, snapshotting, and rollback strategies are implemented for both layers to ensure consistency. The knowledge graph is frequently updated via operational data streams, while the formal ontology is governed by a strict CI/CD pipeline (Ontology-as-Code).

### B.5 Policy Enforcement Layer

Enterprise policies are represented and enforced at runtime using a policy-as-code model. Actions are evaluated against explicit rules based on the actor, resource, action, context, and environmental attributes.

Policy enforcement occurs *prior* to execution and is strictly non-advisory. If a policy evaluates to `DENY`, the action is blocked. Policy decisions, including the specific rules evaluated and the context at the time of evaluation, are recorded and integrated into the system's cryptographic evidence bundle.

### B.6 Agent Orchestration and Durable Execution

Multi-step execution is managed over time by a durable orchestrator. Execution is not a single-step action but a governed trajectory that may span time, systems, and actors.

The orchestrator provides:

* **State Persistence:** The state of the execution is continuously saved, allowing workflows to survive system failures.

* **Interruption Handling:** Workflows can be paused and resumed.

* **Retry Logic:** Transient failures in external systems are handled gracefully.

* **Human-in-the-Loop (HITL) Integration:** Workflows can be suspended pending human review and approval.

Orchestration interacts with validation and policy enforcement at each step, ensuring that long-running workflows remain compliant even if policies or semantic models change during execution.

### B.7 Generative Interface and Ephemeral Application Formation

User interfaces are dynamically generated based on the execution context. The system produces temporary, task-specific interfaces rather than static applications.

This involves streaming interaction patterns and progressive rendering. The interface is structurally bound to the validated intent and the retrieved data, ensuring alignment between the backend execution state and the user-visible elements. As the orchestrator progresses through its plan, the UI updates in real-time to reflect the current status and any required user inputs.

### B.8 Evidence Bundles and Cryptographic Traceability

Every execution produces a structured, immutable record known as an Evidence Bundle. This bundle supports auditability, replayability, and accountability.

An Evidence Bundle contains:

* Raw intent and structured `ActionIntent`.

* Validation results from both the knowledge graph and formal ontology.

* Policy decisions and the specific rules applied.

* Execution steps, tool invocations, and outputs.

* Timestamps and principal identifiers.

These bundles are cryptographically signed and linked via a Merkle tree structure. This evidence integrates with value recognition and governance feedback loops, allowing organizations to audit autonomous decisions and refine policies over time.

### B.9 Failure Modes and Control Mechanisms

The FGM is designed for resilience, explicitly addressing critical system risks:

* **Ontology Failure (Semantic Authority Risk):** Mitigated by the dual-layer model. If the formal ontology is unavailable or inconsistent, the system can gracefully degrade to knowledge graph validation for lower-risk actions, or fail-safe (block) for Zone III actions.

* **Query Hallucination (Semantic Execution Risk):** Mitigated by Compiled Semantic Validation. LLMs never generate executable queries directly; they produce structured intents that are deterministically compiled.

* **Latency Stacking (Performance Risk):** Mitigated by progressive validation and parallel execution where possible. Ontology reasoning is reserved for high-risk actions.

* **Prompt Injection (Security Risk):** Mitigated by the ASIDE principle (Application-Specific Instruction and Data Separation). External data is never treated as executable instruction within the validation or orchestration layers.

### B.10 Assurance Levels and Execution Guarantees

A tiered assurance model applies different levels of validation and control depending on process sensitivity:

* **Level 1 (Existence):** Verifies entities exist in the knowledge graph.

* **Level 2 (Relationship):** Verifies plausible relationships in the knowledge graph.

* **Level 3 (Admissibility):** Formal ontology reasoning (Zone III).

* **Level 4 (Authorization):** Strict policy-as-code enforcement.

* **Level 5 (Accountability):** Full cryptographic evidence bundle generation.

The system adapts dynamically; not all actions require Level 5 assurance, optimizing performance while maintaining necessary governance.

### B.11 Zone III Execution Contract

Any system operating in Zone III must adhere to a strict execution contract. This contract serves as a formal boundary condition for safe deployment and includes:

- Mandatory Progressive Validation (Levels 1-4).
- Mandatory Policy Enforcement prior to execution.
- Defined Human Oversight Triggers (HITL) for high-impact actions.
- Full Audit Traceability (Level 5 Assurance).
- Bounded Execution Guarantees (actions cannot exceed their authorized scope).

### B.12 Latency and Performance Control Model

The system manages execution time while preserving governance through several mechanisms:

* **Staged Validation:** Fast checks (knowledge graph) precede slow checks (ontology).

* **Parallel Execution:** Independent validation steps or data retrieval tasks are executed concurrently.

* **Caching:** Semantic and policy decisions are cached where appropriate, keyed by intent hash and context.

* **Asynchronous Evidence Handling:** Cryptographic signing and ledger appending occur asynchronously to avoid blocking the critical execution path.

User experience is preserved through progressive interaction (streaming UI) even when full validation introduces latency.

### B.13 Prototype Stack and Implementation Path

The conceptual architecture maps to a minimal open-stack implementation using existing, vendor-neutral technologies. This demonstrates feasibility without prescribing specific tooling.

* **Validation Gateway:** Can be implemented using standard web frameworks (e.g., Python/FastAPI or Node.js/Express).

* **Semantic Layer:** Graph databases (e.g., Neo4j, NebulaGraph) for the knowledge graph; reasoning engines (e.g., Apache Jena, RDFox) for the formal ontology.

* **Policy Engine:** Open-source policy-as-code engines (e.g., OPA, Cedar).

* **Orchestration:** Durable execution frameworks (e.g., Temporal, Cadence) combined with agent orchestration libraries.

* **Generative UI:** Component-based frontend frameworks (e.g., React, Vue) driven by server-sent events.

### B.14 Evaluation and Pilot Instrumentation

Empirical evaluation in pilot deployments should focus on metrics that validate both capability and governance:

* **Intervention Rate:** Frequency of HITL triggers.

* **Semantic Failure Rate:** Percentage of intents blocked by the knowledge graph or ontology.

* **Policy Violation Attempts:** Frequency of intents blocked by the policy engine.

* **Latency Distribution:** P50/P95/P99 latency across the progressive validation pipeline.

* **Recognized Value Realization:** Net business value generated minus governance overhead.

These metrics align with a design-science methodology, allowing for iterative refinement of the architecture and its implementation.