AGI Solution Architecture

Generated: 2025-09-30 19:38:41Z (UTC)

# Context and High-Level View

This document summarizes the artificial mind solution architecture, contextualizes the system, and presents a high-level view of the major subsystems and their interactions.

## High-Level Architecture

**Overview (from SYSTEM\_OVERVIEW.md):**

# System Overview (High-Level)  
  
```mermaid  
graph TB  
 %% Clients  
 subgraph Clients  
 User[👤 User]  
 Monitor[📊 Monitor UI]  
 end  
  
 %% Event Bus  
 NATS[(📡 NATS Event Bus\nagi.events.\*)]  
  
 %% Control & Cognition Layer  
 subgraph Cognition[🧠 Cognition & Policy]  
 FSM[⚙️ FSM Engine]  
 SMGM[🧭 Self-Model & Goal Manager]  
 Principles[🔒 Principles Server]  
 end  
  
 %% HDN / Execution Layer  
 subgraph HDNLayer[🛠️ HDN Planning & Execution]  
 HDNAPI[🌐 HDN API]  
 Planner[🧩 Planner / Evaluator]  
 Orchestrator[🧾 Workflow Orchestrator]  
 IE[🤖 Intelligent Executor]  
 CG[🧪 Code Generator]  
 end  
  
 %% Data & Infra  
 subgraph Data[💾 Data & Infra]  
 Redis[(Redis)]  
 Qdrant[(Qdrant\nEpisodic Memory)]  
 Neo4j[(Neo4j\nDomain Knowledge)]  
 Docker[(Docker\nExecution Sandbox)]  
 end  
  
 %% Client flows  
 User -->|Requests / Goals| HDNAPI  
 User -->|Policies / Rules| Principles  
 User -->|Observe| Monitor  
  
 %% Monitor observability  
 Monitor -->|Subscribe| NATS  
 Monitor -->|Query| HDNAPI  
  
 %% HDN publishes events  
 HDNAPI -->|Canonical Events| NATS  
 Planner -->|Plan/Exec Events| NATS  
 Orchestrator -->|Workflow Events| NATS  
 IE -->|Exec Results| NATS  
  
 %% FSM ↔ HDN  
 FSM <-->|Delegate/Status| HDNAPI  
  
 %% Policy influence  
 SMGM -->|Active Goals / Priorities| Redis  
 FSM -->|Consult goals| Redis  
 Planner -->|Consult goals| Redis  
  
 %% Goal lifecycle  
 SMGM -->|agi.goal.\*| NATS  
 SMGM <--|agi.perception.fact\nagi.evaluation.result\nagi.user.goal| NATS  
  
 %% Safety checks  
 IE -->|Pre-exec check| Principles  
 FSM -->|Guards| Principles  
  
 %% Data usage  
 HDNAPI --> Redis  
 Planner --> Redis  
 Orchestrator --> Redis  
 IE --> Redis  
  
 Planner -->|Retrieve episodes| Qdrant  
 IE -->|Index episodes| Qdrant  
  
 Planner -->|Domain constraints| Neo4j  
  
 IE -->|Run code| Docker  
```  
  
## Tools Overview  
  
- Tools are registered in the HDN Tool Registry (...

# Component Deep Dives

## FSM Engine (Control Layer)

* State-driven cognition: perception → learning → planning → evaluation → execution
* Reasoning layer: belief querying, forward-chaining inference, curiosity goals, explanation traces
* Integration: Principles gates, knowledge growth, HDN delegation, NATS events

## HDN (Planning & Execution)

* Intelligent code generation and validation (LLM + Docker)
* Capability learning, caching, and dynamic action creation
* Planner/Evaluator integration; workflow orchestration; file/artifact management

## Self-Model & Goal Manager (Motivation)

* Goals, beliefs, episodic history persisted in Redis
* Policy layer prioritizes goals; emits agi.goal.\* over NATS
* Learning loop updates confidence, priorities, and performance metrics

## Principles Server (Ethics & Safety)

* JSON-based rules; dynamic reload; context-aware checks
* Pre-exec gates for tools/actions; audit trails and denials with reasons
* Layered with LLM safety categorization and Docker sandboxing

## Memory & Knowledge

* Working Memory (Redis): ephemeral state, capabilities, workflow artifacts
* Episodic Memory (Qdrant): vector search over episodes for retrieval-augmented reasoning
* Semantic Knowledge (Neo4j): domain concepts, relations, constraints, safety principles

# Technical Architecture

* APIs: RESTful services (HDN 8081, Principles 8080, Monitor 8082)
* Event Bus: NATS (agi.events.\*) for canonical event envelopes
* Execution Sandbox: Docker with resource limits, timeouts, and isolation
* Persistence: Redis (state/cache), Qdrant (episodes), Neo4j (knowledge)
* Observability: Monitor UI, health checks, metrics, workflow and artifact views
* Deployment: Docker/K3s manifests, cronjobs for scheduled tasks
* Security: Principles gating, content safety, tool metrics and audit logging

# Viability Summary

**Highlights (from SUMMARY.md):**

## Artificial Mind Architecture Summary  
  
### Purpose  
High-level summary of the system’s fundamental architecture and why it remains viable for continued development and deployment.  
  
### Core Architectural Layers  
- \*\*FSM Engine (Consciousness/Control)\*\*: State-driven cognition that orchestrates perception → learning → planning → evaluation → execution. Integrates reasoning, knowledge growth, and principles gates.  
- \*\*HDN (Planning & Execution)\*\*: Intelligent code generation, testing, caching, and execution across languages (Python, Go, JS, Java, C++, Rust) in Docker. Learns reusable capabilities and exposes them via actions.  
- \*\*Self-Model & Goal Manager (Motivation/Policy)\*\*: Tracks goals, episodes, beliefs, and performance. Publishes goal lifecycle events via NATS, prioritizes goals, and influences planning/decision layers.  
- \*\*Principles Server (Ethics/Safety)\*\*: JSON rule engine for pre-exec safety checks, dynamic rule loading, context-aware gating, and auditable denials.  
- \*\*Event Bus (NATS)\*\*: Canonical event backbone for perceptions, planning/execution telemetry, tool lifecycle, and monitoring.  
  
### Memory & Knowledge Subsystems  
- \*\*Working Memory (Redis)\*\*: Ephemeral state, goals, beliefs, tool registry, workflow artifacts, capability cache.  
- \*\*Episodic Memory (Qdrant)\*\*: Vector-based storage of execution episodes for retrieval-augmented reasoning and evaluation.  
- \*\*Semantic Knowledge (Neo4j)\*\*: Domain concepts, relations, constraints, and safety principles for validation and plan scoring.  
  
### Reasoning & Knowledge Growth  
- Forward-chaining inference over domain knowledge (e.g., IS\_A, PART\_OF, ENABLES patterns).  
- Curiosity-driven goals from knowledge gaps and external news signals; deduplication and scoring.  
- Hypothesis generation from facts+knowledge; LLM-assisted screening and prioritization.  
- Transparent explanation traces persisted for UI introspection.  
  
### Safety & Security Model  
- Multi-layer: LLM safety categorization → Principles checks → Docker sandbox execution → Post-validation.  
- Rules are transparent, updateable at runtime, and enforced before tool/action invocation.  
- Audit trails across the pipeline; Monitor UI surfaces violations and outcomes.  
  
### Observability & Ops  
- \*\*Monitor UI\*\*: Real-time health, metrics, workflows, artifacts, capabilities, and memory summaries.  
- \*\*K3s/Docker\*\*: Deployment manifests, cronjobs, and containerized execution. Concurrency guarded by semaphores.  
- \*\*Makefile\*\*: Build/test targets, memory infra bring-up, NATS demos, safety validation, and integration tests.  
  
### Why This Architecture Is Still Viable  
- \*\*Self-Improving Loop\*\*: Learns capabilities (code) and knowledge over time; caches and reuses for performance and reliability.  
- \*\*Robust Safety Posture\*\*: Principles-first gating and container isolation reduce operational risk while enabling powerful tools.  
- \*\*Scalable & Modular\*\*: Stateless APIs, event-driven integration, and composable services (FSM, HDN, Principles, Monitor) supp...