Autonomous Reasoning Systems (ARS)

A Preliminary Technical Note

Lorenzo Masetti[1], Robert Gomez-Reino[1], Oliver Holme[1], Mindaugas Janulis[1], Ioannis Papakrivopoulos[1], Frank Glege[2], Wassef Karimeh[2], Juan Antonio Garcia Garcia[3]

[1]Cleverdist
[2]CERN
[3]Naturgy

October 23, 2025

Contents

Abstract

We introduce Autonomous Reasoning Systems (ARS), a network of continuously operating AI agents organized as a pre-designed responsibility topology. Through recursive cognitive decomposition during the design phase, the job is factored into a hierarchy where each agent reasons about a bounded information surface—a defined view of job state it can observe and act upon. Parent agents reason about aggregated views from their children, delegating finer-grained reasoning downward; this recursive decomposition distributes reasoning complexity across the topology rather than concentrating it in any single agent. This dimensional shift—decomposing job state space rather than tasks—allows the vast implicit context humans rely on to become explicit and bounded within each agent's information surface. Agents use reasoning-as-control (LLM-based planning and self-critique) rather than rule execution, and interact through tools with structured inputs and outputs. Unlike typical agent systems that assemble workflows dynamically from user prompts, ARS embodies mission-by-design: the topology, agent responsibilities, and vertical coordination paths are defined up front and reused across all situations the job faces. Critically, no single agent reasons about the complete job; instead, job-level intelligence emerges from vertical interactions. The system operates indefinitely through event-driven activation: agents dynamically create and manage their own subscriptions to relevant events, respond to scheduled tasks, and handle configured triggers. By decomposing cognition into bounded information surfaces, ARS prevents cognitive collapse and enables long-term operation on complex jobs. This technical note defines ARS as a new category of autonomous systems, establishes conformance criteria, and positions it within the current landscape of agent SDKs and frameworks.

Key Terms

Agent	An LLM-based worker running its own reasoning loop with dedicated tools, operating over a bounded information surface.
Job	The specific, ongoing work the ARS is designed to perform (e.g., operating a facility, managing a service, coordinating logistics).
Topology	The pre-designed network of agents with defined parent-child relationships and coordination paths; created through recursive cognitive decomposition.
Bounded information surface	The limited portion of job state an agent observes and reasons about; prevents cognitive collapse by ensuring no agent faces unbounded complexity.
Event plane	The pub/sub infrastructure through which agents create subscriptions, receive scheduled triggers, and handle configured events for continuous operation.
Actuation/Commit	Actions that create tracked commitments (writes to shared state, system changes, or owner-backed tasks) rather than pure advice.

1 Core Concept

An Autonomous Reasoning System (ARS) is a network of AI agents organized as a pre-designed responsibility topology (tree or DAG) created through recursive cognitive decomposition. Each agent runs its own LLM-based reasoning loop over a bounded information surface—a defined view of job state it can observe and act upon. Parent agents reason about aggregated views from their children, delegating finer-grained reasoning downward; this distributes reasoning complexity across the hierarchy rather than concentrating it. Job-level intelligence emerges from vertical interactions—no single agent reasons about the complete job. Agents use reasoning-as-control (LLM-based planning, self-critique, and adaptive decision-making) and operate continuously through event-driven activation.

Key characteristics:

- Mission-by-design: topology and responsibilities defined up front, reused across all situations
- Event-driven scalability: agents manage their own subscriptions and schedules, enabling real job-scale operation
- Vertical coordination with emergent intelligence: delegate down, execute within scope, report up—job-level intelligence emerges from aggregation
- Bounded information surfaces: prevents cognitive collapse through limited observational and reasoning scope

One-liner

Job-first autonomy: a mission-driven, event-fed, vertically coordinated network of bounded agents that thinks with context and acts with intent.

2 Why This Is a New Category

The landscape of agent frameworks—LangGraph, AutoGen, CrewAI, OpenAI's AgentKit, AWS Bedrock—has matured rapidly. These platforms excel at assembling multi-agent workflows dynamically in response to user prompts. They are mostly **mission-on-demand**: given a goal, they orchestrate agents, allocate tasks, and coordinate execution. This flexibility is valuable for general-purpose AI assistance.

ARS takes a fundamentally different approach: **mission-by-design**. The job's responsibility topology, agent responsibilities, and coordination paths are defined during a design phase—informed by operational expertise about the specific job—and then reused across all situations that job encounters. Human prompts become just another type of event, with priority and routing determined by the pre-designed structure, not by dynamic interpretation.

The dimensional shift: Traditional agent frameworks decompose tasks into sub-tasks. This approach fails at job-scale complexity because even simple tasks require vast implicit context for coherent execution—organizational norms, domain knowledge, relationship dynamics, timing considerations. LLMs struggle to maintain this unbounded implicit context, leading to incoherent decisions as jobs scale in complexity and duration.

ARS decomposes along a fundamentally different dimension: the job's state space itself, not its tasks. Through recursive cognitive decomposition during the design phase, the job is factored into a topology where each agent reasons about a bounded information surface—a defined slice of job state it continuously observes and acts upon. This dimensional shift enables a critical transformation: the vast implicit context humans rely on becomes explicit, bounded, and embedded within each agent's defined information surface. Parent agents reason about aggregated views summarized from their children, delegating finer-grained reasoning downward.

At every level, agents interact through tools with structured inputs and outputs—communicating with parents, children, external systems, or humans. Critically, no single agent reasons about the complete job; instead, job-level intelligence emerges from vertical interactions across the topology. Each agent faces only a bounded reasoning problem regardless of overall job scale, because the implicit context required for coherence has been made explicit within its information surface boundaries.

This is not "hardcoding" in the limiting sense. Rather, it encodes a combination of operational knowledge and human thinking process into the system architecture, eliminating the continuous human-in-the-loop realignment and prompt engineering required to keep general-purpose frameworks on track for specialized, long-running jobs. The mission is embedded in the topology itself.

The category emerges from the combination: pre-designed topology + standing mission + continuous operation + vertical coordination + event-driven activation + bounded information surfaces. Existing frameworks may offer subsets of these capabilities, but ARS makes them foundational requirements. The result is a system optimized not for prompt-to-answer flexibility, but for job-to-completion persistence. (See Section 5 for further discussion of current frameworks.)

3 Minimum Conformance Checklist

Core Principles

The ARS category is built on four foundational principles. The conformance requirements that follow implement and enforce these principles:

I. **Mission-by-design.** The job's topology, agent responsibilities, and coordination paths are designed during a design phase—informed by operational expertise—and reused across all situations, not assembled dynamically per task.

II. **Recursive cognitive decomposition.** Through recursive decomposition, the job is factored into a topology where each agent reasons about a bounded information surface. Parent agents reason about aggregated views, delegating finer-grained reasoning downward. No single agent reasons about the complete job; job-level intelligence emerges from vertical interactions across the topology.

III. **Vertical coordination with emergent intelligence.** Agents delegate downward, execute within bounded scope, and report upward. Intelligence aggregates through the topology rather than being centralized in any single agent.

IV. **Continuous, event-driven operation.** The system operates indefinitely through subscriptions, schedules, and configured triggers, with agents managing their own activations within the standing mission structure.

Conformance Requirements

An implementation may claim to be an ARS if it provides the following capabilities. Each requirement serves a specific purpose in enabling the core principles above:

1. **Topology specification (declarative).**
   A machine-readable definition of agent responsibilities, information surfaces, vertical links (parent/child relationships), and allowed delegation/reporting patterns. This encodes the job's reasoning architecture and enables the mission-by-design approach—operational expertise is embedded in the topology itself, not discovered at runtime.

2. **Persistent, recoverable state.**
   Each agent maintains its state as an ongoing conversation thread, including subscribed events and messages exchanged through the topology. The critical value lies in the distributed, interconnected context across all agents—knowledge aggregates at different abstraction levels, enabling the system to zoom in for detail or zoom out for high-level view as questions demand, without cognitive collapse. While state persistence across restarts is strongly recommended for production systems, even reinitializing preserves this distributed knowledge architecture.

3. **Event plane with pub/sub and backpressure.**
   In addition to built-in triggers and schedules, agents can dynamically create subscriptions to event streams, schedule periodic tasks, and handle configured events. Backpressure mechanisms prevent overload. This is essential for scalability—event-driven activation replaces polling and enables agents to operate asynchronously at real-world job scale, including scenarios where multiple ARS systems may subscribe to similar event streams.

4. **Vertical messaging semantics.**
   Clear primitives for agents to communicate within their authority: delegation/demands to children, execution within scope, responses/reports to parents. Any agent can initiate messages based on its responsibility and context, not just in response to parent requests. Messages are correlated to track exchanges. This implements the coordination model that prevents agents from becoming disconnected reasoners.

5. **Memory surfaces.**
   Each agent maintains working memory (current case/context) and can access experience memory. The system-level view emerges from the properly distributed responsibility topology itself. For real-world tasks, external/retrieval memory via tools (databases, knowledge bases) is typically essential—this is where operational expertise is built and distributed across the system.

6. **Actuation with commit semantics.**
   Agents execute actions that create tracked commitments: writes to shared job state, external system changes, or owner/SLA-backed tasks. This distinguishes ARS from purely advisory systems—actions have commit semantics, creating obligations that the system must track and fulfill. Pure advice without tracked commitment does not constitute actuation.

7. **Scalability assurance through bounded information surfaces.**
   The topology must be designed such that each agent's information surface—the

portion of job state it observes and reasons about—remains bounded and within current model capabilities. Leaf agents observe detailed views of narrow domains; parent agents observe aggregated summaries from their children, not raw data. When an agent encounters questions or tasks outside its defined responsibility, it delegates to a more specialized child or escalates to its parent—it does not expand beyond its information surface boundaries. This cognitive decomposition prevents collapse: each agent faces a bounded reasoning problem regardless of overall job complexity or duration. During development and testing, detecting information surfaces that prove too complex for model coherence requires topology reengineering; this is part of the design process, not a runtime failure.

Note: Systems lacking any of these elements may be useful multi-agent frameworks, but they do not conform to the ARS category as defined here.

Production recommendation: Systems deployed in production environments should include forensic replay ledgers (capturing inputs, model versions, prompts, outputs, and causal links) to support audit and debugging, though this is not required for v0.1 conformance.

Design challenge: Creating an ARS topology requires two distinct capabilities: (1) domain expertise—deep knowledge of the job's operational reality, and (2) cognitive introspection—the ability to articulate one's own thinking process. The designer must identify what implicit context enables coherent reasoning at each level: What portions of job reality must each agent observe? What implicit knowledge must become explicit? What are the natural aggregation boundaries where detailed context compresses into summary views? This is not conventional task breakdown; it is the externalization of implicit operational knowledge into an explicit topology of bounded observational surfaces. This dual requirement (domain knowledge + cognitive self-awareness) represents a significant design barrier. Future ARS implementations may enable self-improvement through collaboration with human experts or advanced models to extend or modify their own operational topologies, potentially lowering this barrier, but this is not a conformance requirement for v0.1.

4 Industry Landscape

During the past few months and continuing through October 2025 marked a maturation point for production multi-agent systems, with major vendors shipping integrated tooling for agent design, orchestration, and governance.

OpenAI launched AgentKit (October 6, 2025) [1], providing Agent Builder for visual multi-agent workflow design, ChatKit for embeddable UIs, and enhanced Evals with reinforcement fine-tuning hooks. The toolchain is workflow-centric with session-scoped execution; whether it supports standing missions depends on the chosen hosting runtime (e.g., deployed on LangGraph Server, Microsoft Agent Framework, or AWS AgentCore).

AWS took Bedrock AgentCore to general availability (October 13) [2], offering a session-isolated microVM runtime (Firecracker) with max 8-hour sessions/async jobs, A2A protocol, MCP Gateway, and governance features. AgentCore provides production infrastructure for multi-agent systems (integrating with LangGraph, CrewAI, OpenAI Agents); mission continuity beyond 8 hours requires session rollover and externalized topology state.

Anthropic released Claude Sonnet 4.5 and the Claude Agent SDK (September 29) [3, 4], emphasizing long-horizon autonomy with observed 30+ hour coherent operation on complex tasks. The SDK exposes agent loop primitives (subagents, context compaction, memory tools) and positions itself for sustained autonomous work—closer to ARS's continuous operation model, though without pre-designed job topologies.

LangGraph reached 1.0 (October 17 tag / October 22 blog) [5], formalizing a durable runtime via LangGraph Server with built-in persistence (PostgreSQL), task queues (Redis), assistants/threads/runs, cron jobs, and webhooks. The platform provides checkpointing, human-in-the-loop interrupts, and tracing. LangGraph Server supports standing agents with durable state across threads, though topology governance (authority schemas, responsibility constraints) must be implemented at the application level.

Microsoft introduced the Agent Framework (October 1) [6], consolidating AutoGen and Semantic Kernel into a unified SDK/runtime for multi-agent applications. **CrewAI** continued platform updates [7] with enterprise features (agent repositories, orchestrated workflows) and MCP integration.

Across this landscape, the industry has converged on key enablers—durable state, governance, MCP/A2A interop, and long-context models. Importantly, pre-designed workflows with persistent state are now first-class in several platforms (LangGraph Server with persistence/queues/cron, Microsoft Agent Framework with stateful workflows, AgentCore with session-isolated runtimes). The substrate for standing missions has arrived. What remains distinctive about ARS is the **topology governance layer**: mission-by-design with authority schemas, vertical coordination semantics, and bounded responsibility enforcement across long-lived deployments.

5 ARS vs. Agent SDKs/Frameworks

ARS is a system category, not a competing framework. The platforms described in Section 4 provide valuable infrastructure that could be leveraged to implement an ARS. However, none enforce the ARS pattern as defined in Section 3. Here we examine what each framework provides and where implementers would encounter gaps:

LangGraph

LangGraph 1.0 with LangGraph Server offers the richest foundation for ARS implementation among current frameworks. The platform provides built-in persistence (PostgreSQL), task queues (Redis), assistants/threads/runs, cron jobs, webhooks, checkpointing, and human-in-the-loop interrupts—delivering durable, event-driven execution out of the box. The graph abstraction naturally supports vertical message passing through parent-child node relationships.

Implementation gaps: LangGraph Server provides the durability and eventing infrastructure; the gap for ARS is topology governance across long-lived deployments. An ARS implementer would need to: (1) define authority schemas that constrain who may delegate to whom and when, (2) encode responsibility boundaries that persist across all job situations, not just individual workflows, and (3) implement vertical coordination semantics (delegate/report/escalate) as enforced patterns rather than application-level conventions. The framework provides the substrate but doesn't enforce the ARS governance pattern—developers could equally well build mission-on-demand systems.

Anthropic Claude Agent SDK

The Claude Agent SDK emphasizes long-horizon autonomy, with observed 30+ hour coherent task execution. Its loop primitives (subagents, context compaction, memory tools) support sustained reasoning, and Sonnet 4.5's extended context makes it viable for complex, multi-step work. This positions Anthropic's offering closest to ARS's continuous operation goal at the model level.

Implementation gaps: The SDK provides autonomous task completion capabilities but no job topology abstraction. An ARS implementation would require building: (1) a responsibility topology layer above individual agents, (2) vertical coordination semantics for delegation and reporting, (3) a persistence architecture ensuring the topology (not just individual agent state) survives across sessions, and (4) an event plane for subscription-based activation rather than task-driven invocation. The 30+ hour runtime is powerful for individual agents but doesn't translate directly to indefinite job-level operation with distributed context across a topology.

AWS AgentCore

AgentCore provides production-grade infrastructure with governance, isolation (micro-VMs), A2A protocol for inter-agent communication, and max 8-hour sessions/async jobs. Its integration with multiple orchestration frameworks (LangGraph, CrewAI, OpenAI Agents) positions it as a deployment platform rather than an orchestration pattern.

Implementation gaps: AgentCore orchestrates dynamically assembled teams per execution request. To implement ARS, one would need to: (1) maintain a persistent agent topology that runs indefinitely rather than per-invocation, (2) encode mission-by-design such that the job's structure is fixed and reused, not assembled on demand, (3) implement vertical coordination as a first-class pattern rather than generic A2A messaging, and (4) ensure the runtime doesn't terminate after task completion but continues monitoring subscriptions and schedules. AgentCore provides robust infrastructure, but the standing mission model must be architected on top.

OpenAI AgentKit

AgentKit's Agent Builder emphasizes visual multi-agent workflow design with per-task assembly. Its design-evaluate-deploy toolchain optimizes for rapid workflow creation and refinement through reinforcement fine-tuning, targeting general-purpose task automation rather than job specialization.

Implementation gaps: The per-task workflow model is fundamentally at odds with ARS's standing mission approach. Implementing ARS would require: (1) abandoning or constraining the visual workflow builder to enforce a fixed topology, (2) adding persistence such that workflows don't terminate after task completion, (3) implementing vertical coordination patterns not native to the workflow model, and (4) shifting from prompt-driven invocation to event-driven continuous operation. While the underlying GPT models are capable, the framework's design philosophy diverges from ARS requirements.

Microsoft Agent Framework & CrewAI

Microsoft's unified SDK consolidates AutoGen and Semantic Kernel into a multi-agent application platform. CrewAI offers orchestrated workflows with agent repositories and MCP integration. Both follow similar patterns: dynamic team assembly, task-oriented execution, and flexible orchestration.

Implementation gaps: Similar to other frameworks, the primary gaps are: lack of standing mission enforcement, no job topology abstraction, and task-scoped rather than job-persistent operation. Implementers would need to build ARS-specific layers atop the provided primitives.

Summary

These frameworks solve important problems in task automation, dynamic agent coordination, and production deployment. An ARS implementation could leverage them as infrastructure—LangGraph for stateful orchestration, AgentCore for governance and deployment, Claude models for capable reasoning—but would require additional architectural layers to enforce the seven conformance requirements (Section 3) and the mission-by-design philosophy (Section 2). The frameworks provide tools; ARS defines the pattern those tools must implement to achieve continuous, job-focused autonomous operation.

6 Implementation Pattern: Industrial Operator (IO)

Industrial Operator (IO) is an ARS implementation designed for continuous autonomous operation of industrial facilities. IO is currently deployed at Naturgy for multi-plant control room operations and undergoing pilot deployment at CERN's CMS experiment, with additional applications in development for petrochemical and production facilities. This section demonstrates how the ARS pattern manifests in a real system operating in critical infrastructure.

Mission and Context

IO's standing mission is continuous facility operation: monitoring equipment state, detecting anomalies, coordinating responses, and autonomously managing routine operational decisions while escalating exceptional situations to human operators. The system operates 24/7, handling hundreds of concurrent events across facility subsystems through a pre-designed topology of specialized agents.

ARS Conformance

IO satisfies the seven ARS conformance requirements through the following architecture:

1. **Topology specification:** Defined declaratively using SmartNodes++ (SN++) [8] to configure WinCC OA Plant Model trees, which formalize agent responsibilities and information surfaces.

2. **Persistent state:** Agent conversations and subscriptions are stored in persistent databases, maintaining distributed context across system restarts.

3. **Event plane:** Built on Siemens WinCC OA [10] event-driven SCADA platform, which provides high-performance pub/sub infrastructure. Agents create

both pre-configured and dynamic subscriptions, including complex expressions over event streams.

4. **Vertical messaging:** Implemented through an IO orchestration layer (Python-based) coordinating agent communication. Agents maintain explicit knowledge of their parent, children, and bounded responsibilities.

5. **Memory:** Agent working memory and experience memory are maintained in separate databases with different lifecycle policies, distinct from the conversation threads.

6. **Actuation:** IO both suggests and executes actions—preparing commands for operator review and, within validated boundaries, autonomously executing operational procedures (setpoint adjustments, system commands, notifications).

7. **Scalability:** Topology depth adapts to facility complexity; agents observe bounded information surfaces (equipment-level workers, area-level domain coordinators, facility-level root).

Technical Foundation

IO leverages Siemens WinCC OA [10] SCADA as its event infrastructure and real-time distributed database, integrating with operational technology (OT) and information technology (IT) systems through WinCC OA's extensive driver ecosystem. The SmartNodes++ framework [8] provides the ontology and lifecycle management layer. Python orchestrates agent reasoning loops and inter-agent coordination, delegating tool execution to the SCADA layer and external systems.

Deployment Status

IO is currently deployed in Naturgy's multi-plant control room, autonomously supporting operations across 10+ gas turbine power plants throughout Spain and growing. The system operates with progressive autonomy expansion—selected processes run fully autonomously, while scope increases as operational validation accumulates. An ARS topology for CERN's CMS Detector Control System [9] is in development, with pilot deployment scheduled for completion by year-end 2025. Additional applications are in development for petrochemical and production facilities. The system demonstrates that the ARS pattern—mission-by-design with recursive cognitive decomposition—can operate reliably in critical infrastructure contexts where continuous, coherent decision-making is essential.

Detailed performance metrics, operational insights, and topology design patterns from these deployments will be published in subsequent releases of this technical note.

References

[1] OpenAI. "Introducing AgentKit." October 6, 2025.
https://openai.com/index/introducing-agentkit/

[2] Amazon Web Services. "Make agents a reality with Amazon Bedrock AgentCore." October 13, 2025.
https://aws.amazon.com/blogs/machine-learning/amazon-bedrock-agentcore-is-now-generally-available/

[3] Anthropic. “Introducing Claude Sonnet 4.5.” September 29, 2025.
https://www.anthropic.com/news/claude-sonnet-4-5

[4] Anthropic. “Building agents with the Claude Agent SDK.” September 29, 2025.
https://www.anthropic.com/engineering/building-agents-with-the-claude-agent-sdk

[5] LangChain Blog. “LangGraph Platform in beta: New deployment options for agents.” October 22, 2025.
https://blog.langchain.com/langgraph-platform-announce/

[6] Microsoft Azure. “Microsoft Agent Framework.” October 1, 2025.
https://azure.microsoft.com/en-us/blog/introducing-microsoft-agent-framework/

[7] CrewAI. “CrewAI Documentation.” Accessed October 2025.
https://docs.crewai.com/

[8] Cleverdist. “SmartNodes++: Ontology Framework for WinCC OA SCADA.”
https://www.cleverdist.com/smartnodes

[9] Gomez-Reino, R., et al. “The Compact Muon Solenoid Detector Control System.” CMS Conference Report, CERN, 2009.
https://cds.cern.ch/record/1358846/files/CR2009_346.pdf

[10] Siemens/ETM. “SIMATIC WinCC Open Architecture.”
https://www.winccoa.com/