SOSA™

Supervised Orchestrated Secured Agents

A Methodology for Production-Grade Autonomous AI Operations

Michal Shatz
MSApps Research
michal@msapps.mobi

March 2026

Abstract

The rapid proliferation of large language model (LLM)-based agents in enterprise workflows has exposed a critical gap: the absence of a unifying framework that governs how autonomous agents should be supervised, coordinated, and constrained in production environments. Existing approaches treat autonomy and control as opposing forces, resulting in systems that are either too brittle to scale or too opaque to trust. We propose ***SOSA™ — Supervised Orchestrated Secured Agents*** *— a four-pillar methodology that reconciles agent autonomy with organizational accountability. SOSA provides a formal structure for deploying multi-agent systems that operate continuously, adapt to heterogeneous toolchains, and maintain verifiable compliance with security and governance policies. We present OpsAgent as a reference implementation of the SOSA framework deployed in production across multiple business verticals, demonstrating that principled agent governance does not require sacrificing operational velocity.*

Keywords: AI agents, multi-agent systems, enterprise automation, agent governance, orchestration, human-in-the-loop, zero-trust security, production AI, LLM operations

1. Introduction

The transition of AI agents from research demonstrations to production deployments represents one of the most consequential shifts in enterprise technology since the advent of cloud computing. Large language models (LLMs), equipped with tool-use capabilities and persistent context, now possess the theoretical capacity to automate complex operational workflows that previously required dedicated human staff: lead qualification, financial reconciliation, recruitment screening, content production, and cross-platform coordination.

Yet the vast majority of enterprise agent deployments fail. Industry estimates suggest that fewer than 15% of AI agent pilot programs survive to sustained production use. The failure modes are predictable and systematic. The first is *under-autonomy*: agents configured as elaborate chatbots, requiring human input at every decision point, producing marginal efficiency gains at high interaction costs. The human operator becomes a bottleneck, and the promised automation benefits evaporate. The second is *over-autonomy*: agents deployed with insufficient guardrails that produce cascading errors, hallucinate business-critical actions, or silently drift from their intended objectives. A single unguarded financial agent can generate erroneous invoices for weeks before detection.

Both failure modes stem from the same root cause: the absence of a structured methodology for calibrating the supervision-autonomy spectrum. Organizations lack a principled framework for determining which tasks an agent should handle independently, which require human oversight, and how multiple agents should coordinate without creating brittle dependencies or security vulnerabilities.

This paper introduces **SOSA™ — Supervised Orchestrated Secured Agents** — a four-pillar methodology designed to bridge this gap. SOSA provides formal governance primitives at the framework level, ensuring that agent autonomy is proportional to demonstrated reliability, action reversibility, and domain risk tolerance. We further present OpsAgent, a commercial platform that implements the SOSA framework in production, currently operating across 18+ department types for real businesses.

2. Related Work

2.1 Multi-Agent Frameworks

The emergence of multi-agent architectures has produced several notable frameworks. Microsoft's AutoGen [1] introduces conversable agents that coordinate through structured dialogue patterns. CrewAI [2] emphasizes role-based agent design with sequential and hierarchical task delegation. LangGraph [3] provides a stateful, graph-based orchestration layer built atop LangChain. Each framework addresses coordination mechanics but treats supervision, security, and governance as application-level concerns rather than framework-level primitives. SOSA integrates these dimensions into the core architecture.

2.2 Reasoning and Execution Patterns

The ReAct paradigm [4] demonstrated that interleaving reasoning traces with tool-use actions significantly improves agent reliability. Plan-and-Execute architectures [5] separate high-level planning from step-wise execution, enabling more robust error recovery. Reflexion [6] introduced self-reflective agents that learn from execution failures. SOSA's Plan-Act-Verify loop extends these patterns by adding an explicit verification phase with organizational feedback loops and a formal trust gradient.

2.3 Enterprise AI Governance

The NIST AI Risk Management Framework [7] provides comprehensive guidelines for AI system governance but does not address the specific challenges of autonomous agent deployments. The EU AI Act [8] establishes risk-based classification for AI systems, with high-risk categories requiring human oversight

and transparency—principles that SOSA operationalizes through its Supervised and Secured pillars. ISO/IEC 42001 [9] defines requirements for AI management systems but remains abstract regarding multi-agent coordination. SOSA translates these regulatory intentions into concrete architectural patterns.

2.4 Trust and Safety Frameworks

Capability-based security [10] provides a theoretical foundation for fine-grained access control that SOSA adopts for agent permission management. The principle of least privilege, when applied to AI agents, demands that each agent possesses only the credentials and tool access necessary for its defined role—a requirement that most existing frameworks honor in documentation but not in enforcement. Zero-trust architecture principles [11] inform SOSA's inter-agent communication model, where every message requires mutual attestation regardless of network topology.

3. The SOSA™ Framework

SOSA defines four interdependent pillars that collectively govern agent behavior in production environments. No single pillar is sufficient in isolation; it is their integration that produces systems capable of sustained autonomous operation under enterprise constraints.

3.1 Supervised

Every agent in a SOSA-compliant system operates under a defined supervision policy. Human-in-the-loop checkpoints are not optional add-ons—they are first-class architectural primitives. Supervision is graduated: routine, low-impact tasks execute autonomously, while high-stakes actions require explicit human approval or are bounded by pre-authorized decision envelopes.

The supervision model is governed by an impact scoring function. Each agent action a is assigned an impact score $I(a)$ computed from factors including financial magnitude, external visibility, reversibility, and regulatory sensitivity. The organization defines a risk threshold θ. Actions where $I(a) \leq \theta$ execute autonomously. Actions where $I(a) > \theta$ are gated on supervisor approval $S(a) = 1$.

> *Formally: for any agent action a with impact score $I(a) > \theta$, execution is gated on supervisor approval $S(a) = 1$, where θ is a configurable organizational risk threshold.*

Critically, the threshold θ is not static. SOSA implements a trust gradient that adjusts based on observed agent performance. Agents that consistently produce correct outcomes for actions near the threshold boundary earn expanded autonomy. Agents that exhibit failure patterns are automatically escalated to tighter supervision. This creates an adaptive system where trust is earned through verifiable behavior, not assumed through configuration.

3.2 Orchestrated

Isolated agents produce isolated outcomes. In any non-trivial operational environment, agents must coordinate across temporal, informational, and toolchain dimensions. SOSA mandates an orchestration layer that manages this coordination explicitly rather than relying on ad-hoc inter-agent messaging.

The orchestrator maintains a directed acyclic graph (DAG) $G = (V, E)$ where vertices represent agent tasks and edges encode data dependencies and temporal constraints. This structure ensures conflict-free concurrent execution: agents that share no data dependencies execute in parallel, while dependent tasks respect ordering constraints. The DAG is constructed from business-logic specifications, not inferred from runtime behavior, ensuring deterministic scheduling under normal operating conditions.

> *The orchestrator maintains a directed acyclic graph $G = (V, E)$ where vertices represent agent tasks and edges encode data dependencies and temporal constraints, ensuring conflict-free concurrent execution.*

Context sharing between agents is mediated through structured registries rather than direct message passing. When a lead-qualification agent determines that a prospect is high-priority, it writes a structured context record to the shared registry. The outreach agent reads this record on its next execution cycle. This decoupled architecture prevents cascading failures: if the outreach agent fails, the qualification context

remains intact and available for retry.

3.3 Secured

Security in SOSA is not a perimeter—it is a property of every layer. Each agent runs in an isolated execution environment with scoped credentials, zero-trust network boundaries, and cryptographically verifiable audit trails. No agent can access resources beyond its declared permission set, and all inter-agent communication passes through authenticated channels.

Each agent A_i is assigned a capability set $C_i \subseteq C$ enforced at runtime by the security layer. Capabilities are not inherited from the orchestrator or from peer agents; they are explicitly declared in the agent's manifest and verified before every tool invocation. Cross-agent communication requires mutual attestation: a message $msg(A_i, A_j)$ is valid if and only if $auth(A_i) \wedge auth(A_j) = true$.

> *Each agent A_i is assigned a capability set $C_i \subseteq C$ enforced at runtime. Cross-agent communication requires mutual attestation: $msg(A_i, A_j)$ is valid iff $auth(A_i) \wedge auth(A_j) = true$.*

Every action taken by every agent is logged to an immutable audit store. The audit trail includes the action type, timestamp, input parameters, output results, the agent's identity, and the authorization chain that permitted the action. This provides complete forensic traceability—a requirement for regulatory compliance in financial services, healthcare, and government sectors.

3.4 Agents

SOSA agents are not scripts with LLM wrappers. They are goal-directed autonomous entities with persistent context, tool-use capabilities, and adaptive planning. Each agent possesses a defined role ontology, success metrics, and failure recovery strategies—enabling them to operate as reliable participants in a larger organizational system.

An agent in SOSA is formally defined as a tuple $A = (R, T, M, P)$ where R is the role specification (defining the agent's domain, objectives, and constraints), T is the tool manifest (the set of external APIs, databases, and communication channels the agent is authorized to invoke), M is the memory and context store (providing persistence across execution cycles), and P is the planning policy (governing how the agent selects actions given its current state and objectives).

> *An agent is a tuple $A = (R, T, M, P)$ where R is the role specification, T is the tool manifest, M is the memory/context store, and P is the planning policy governing action selection.*

The role specification R is particularly consequential. Unlike generic agents that rely on prompt engineering to constrain behavior, SOSA agents have their domain boundaries, success criteria, and escalation triggers encoded in structured specifications that are enforced by the runtime, not merely suggested to the LLM. A financial reconciliation agent cannot be prompt-injected into sending emails—its tool manifest does not include email capabilities, and the security layer enforces this constraint at the API level.

4. The SOSA Execution Model

In a SOSA-compliant system, agent execution follows a three-phase loop: **Plan**, **Act**, and **Verify**. This loop operates at two timescales: the individual task level (seconds to minutes) and the organizational learning level (days to weeks).

4.1 Plan

During the planning phase, the agent decomposes its current objective into a sequence of tool calls and information retrievals, subject to the constraints in its capability set C_i. The planning policy P evaluates available actions against the agent's role specification R, filtering out any actions that would violate domain boundaries or exceed the agent's authorization scope. The resulting plan is a partially ordered set of operations with explicit preconditions and expected postconditions.

4.2 Act

During the action phase, each planned step is executed against real external systems—APIs, databases, communication platforms, file systems—with every interaction logged to the immutable audit store. Actions that exceed the impact threshold θ are paused and escalated to the appropriate supervisor. The orchestration layer monitors action execution for timeouts, errors, and unexpected state changes, triggering the agent's failure recovery strategy when anomalies are detected.

4.3 Verify

During the verification phase, the orchestrator evaluates the outcome against declared success criteria defined in the agent's role specification. Verification is not merely a boolean pass/fail check; it produces a structured evaluation record that feeds into the trust gradient. Agents that consistently meet their success criteria earn expanded autonomy boundaries. Agents that exhibit failure patterns are automatically escalated to tighter supervision policies.

This creates a continuous improvement mechanism: the system's governance posture adapts to observed agent reliability rather than relying on static, manually configured trust levels. Over time, well-performing agents require less human intervention, while unreliable agents receive proportionally more oversight—precisely the adaptive trust model that static rule-based systems cannot achieve.

5. Reference Implementation: OpsAgent

OpsAgent is the first commercial implementation of the SOSA framework. Every architectural decision—from isolated virtual machine environments to the orchestration scheduler to the human-approval gates—maps directly to a SOSA pillar. OpsAgent has been operating in production since early 2025, initially deployed to run the entire operations of MSApps, a technology company, before being offered as a service to external clients.

5.1 Architecture

Supervised: OpsAgent implements configurable approval workflows per agent type. A daily briefing system surfaces all actions taken by all agents to the human operator. High-impact operations—financial transactions above a configurable threshold, external communications to new contacts, contract modifications—require explicit human sign-off before execution. The impact threshold is adjustable per client and per department.

Orchestrated: A centralized scheduler coordinates 18+ agent types across temporal and data dependencies. Agents share context through structured registries, not ad-hoc message passing. The scheduler supports both time-triggered execution (daily financial reconciliation, weekly content calendars) and event-triggered execution (new lead received, invoice submitted for matching).

Secured: Each client environment runs on an isolated virtual machine with scoped OAuth credentials. Zero business data is stored on OpsAgent servers—agents process data in transit and write results directly to the client's existing systems. Every action is logged to an immutable audit trail accessible to the client through the OpsAgent dashboard.

Agents: OpsAgent agents are goal-directed entities with persistent memory, real tool access (calendars, CRMs, accounting systems, communication platforms), and adaptive planning. Each agent is defined by a structured role specification, not a prompt template. This ensures behavioral consistency across LLM model updates and prevents prompt injection from altering agent behavior.

5.2 Production Metrics

OpsAgent's production deployment demonstrates the viability of the SOSA methodology under real operational conditions. The system automates 18 department types including lead management, sales outreach, recruiting, financial reconciliation, content production, and customer support. The platform operates 24/7 with a gross margin of 96%, reflecting the cost efficiency of AI-driven operations compared to traditional staffing models. New client deployments are completed in days rather than months, enabled by the modular architecture of SOSA-compliant agents.

6. Implications for Enterprise Adoption

The SOSA methodology directly addresses the three primary barriers to enterprise AI agent adoption.

Accountability. By requiring full audit trails and graduated supervision, SOSA satisfies regulatory and internal governance requirements. Every agent action is traceable to a specific authorization chain. When an agent makes an error, the audit trail identifies exactly what happened, when, and why—enabling rapid remediation rather than opaque debugging. This accountability structure aligns with the requirements of NIST AI RMF, EU AI Act high-risk provisions, and SOC 2 compliance frameworks.

Reliability. By mandating orchestration and structured inter-agent context sharing, SOSA eliminates the coordination failures that plague multi-agent deployments. The DAG-based scheduling ensures that agents respect data dependencies. The structured registry model prevents cascading failures. The Plan-Act-Verify loop with its trust gradient ensures that unreliable agents are automatically constrained before they can cause organizational damage.

Compliance. By treating security as a first-class design constraint rather than an afterthought, SOSA enables deployment in environments where data sensitivity precludes the use of conventional SaaS-based AI solutions. The zero-trust architecture, scoped credentials, and data-in-transit processing model mean that client data never resides on third-party servers—a requirement for financial services, healthcare, legal, and government clients.

These properties collectively justify premium positioning. Organizations adopting SOSA-compliant platforms are not purchasing a chatbot or an automation script; they are deploying a governed, auditable, enterprise-grade AI operations layer. The methodology's rigor is precisely what enables the trust required for mission-critical deployments.

7. Conclusion

SOSA represents a necessary evolution from ad-hoc AI agent deployment to principled, production-grade AI operations. By integrating supervision, orchestration, security, and agent design into a unified methodology, SOSA provides organizations with a framework that scales from initial pilot to full operational deployment without sacrificing governance or accountability.

The OpsAgent reference implementation demonstrates that SOSA is not merely theoretical: it operates in production today, automating real business operations with measurable outcomes. Organizations adopting SOSA can expect to deploy AI agents that are not merely impressive in demos, but durable in production—systems that earn trust through verifiable behavior rather than demanding it through marketing claims.

Future work includes formal verification of agent supervision policies using model-checking techniques, development of cross-organizational SOSA compliance certification standards, and extension of the trust gradient model to incorporate multi-stakeholder approval chains for regulated industries.

References

[1] Wu, Q., et al. "AutoGen: Enabling Next-Gen LLM Applications via Multi-Agent Conversation." arXiv:2308.08155, 2023.

[2] Moura, J. "CrewAI: Framework for Orchestrating Role-Playing Autonomous AI Agents." Open-source framework, GitHub, 2024.

[3] LangChain. "LangGraph: Build Stateful, Multi-Agent Applications with LLMs." Open-source framework documentation, 2024.

[4] Yao, S., et al. "ReAct: Synergizing Reasoning and Acting in Language Models." ICLR, 2023.

[5] Wang, L., et al. "Plan-and-Solve Prompting: Improving Zero-Shot Chain-of-Thought Reasoning." ACL, 2023.

[6] Shinn, N., et al. "Reflexion: Language Agents with Verbal Reinforcement Learning." NeurIPS, 2023.

[7] National Institute of Standards and Technology. "AI Risk Management Framework (AI RMF 1.0)." NIST, January 2023.

[8] European Parliament. "Regulation (EU) 2024/1689 Laying Down Harmonised Rules on Artificial Intelligence (AI Act)." Official Journal of the European Union, 2024.

[9] ISO/IEC 42001:2023. "Information Technology — Artificial Intelligence — Management System." International Organization for Standardization, 2023.

[10] Dennis, J.B. and Van Horn, E.C. "Programming Semantics for Multiprogrammed Computations." Communications of the ACM, 9(3), 1966.

[11] Rose, S., et al. "Zero Trust Architecture." NIST Special Publication 800-207, August 2020.

[12] Wei, J., et al. "Chain-of-Thought Prompting Elicits Reasoning in Large Language Models." NeurIPS, 2022.

[13] Schick, T., et al. "Toolformer: Language Models Can Teach Themselves to Use Tools." NeurIPS, 2023.

[14] Park, J.S., et al. "Generative Agents: Interactive Simulacra of Human Behavior." UIST, 2023.

[15] Bai, Y., et al. "Constitutional AI: Harmlessness from AI Feedback." arXiv:2212.08073, 2022.

