Nick Clark · Published June 22, 2026 · PDF

Model Context Protocol: Agent Tool Use as Protocol

The Model Context Protocol standardizes how applications give language models access to tools and data, replacing bespoke integrations with one client-server protocol. It standardizes how an agent reaches its tools. It does not standardize what an agent is.

Protocol and Adoption Reality

The Model Context Protocol, introduced by Anthropic and since adopted broadly across the industry, standardizes how an application supplies context and tools to a language model. Through a client-server protocol, an MCP server exposes tools the model can call, resources it can read, and prompts it can use, and an MCP client, embedded in the host application, brokers the model's access to them. The achievement is real and the adoption is wide: before MCP, every integration between a model and an external tool or data source was bespoke, and MCP replaced that sprawl with one protocol that a tool provider implements once and any MCP-aware host can consume. It is now a de facto standard for connecting models to the world.

What MCP standardizes is access: how a model reaches tools and resources. That is a necessary layer, and the agent schema is not a competitor to it. The schema addresses a different question, one MCP does not attempt to answer.

The Structural Choice: An Agent Is a Client of Endpoints

In the MCP model, the unit of design is the tool-and-resource interface. A server advertises an endpoint surface, and the agent is a client that consumes it: the protocol defines the calls between them, the shapes of requests and responses, and the negotiation of what is available. What the protocol does not define is what the agent itself is. There is no portable representation of the agent's intent, its accumulated memory, the policy that governs it, the mutation it is about to perform, or the lineage of how it arrived at its current state. The agent, in MCP terms, is whatever the host application happens to maintain in its own runtime, and that internal state is neither standardized nor portable nor governed by the protocol. MCP makes the agent's tools interoperable; it leaves the agent itself an unstandardized, host-local thing.

What the Agent Schema Provides

The agent schema standardizes the agent rather than its tool access. A semantic agent object carries six canonical typed fields: an intent that names what it seeks, a context that situates it, a memory that holds its accumulated commitments, a policy reference that governs what it may do, a mutation descriptor that declares the change it proposes, and a lineage that records how it got here. Because the agent is a structured object rather than host-local runtime state, it is portable across systems, serializable and resumable, structurally validatable, and governable by construction, since the policy and lineage it must satisfy travel with it. An agent that calls tools through MCP and is itself a semantic agent object gains a governed, portable identity that MCP does not provide, while losing nothing of MCP's tool interoperability. The two layers compose cleanly: MCP defines how the agent reaches its tools, and the schema defines what the agent is as it does so.

Composable, Not Competing

MCP and the agent schema occupy different layers of the same stack. One is a protocol for tool and context access; the other is a structural definition of the agent object. A host that speaks MCP can carry semantic agent objects as its agents, and a semantic agent object can reach its tools over MCP, so adopting both is additive rather than a choice between them. The schema is the missing object layer beneath the access layer MCP standardized. No relationship, endorsement, or infringement is asserted; the comparison is architectural.

Disclosure Scope

The semantic agent object and its six canonical typed fields, intent, context, memory, policy reference, mutation descriptor, and lineage, together with structural validation, partial-agent support, and traceable semantic lineage that make the agent a portable, governable object, are disclosed in the agent schema filing (U.S. Application No. 19/452,651). This article compares that disclosed object model with Anthropic's publicly documented Model Context Protocol and positions the schema as the agent-object layer beneath MCP's tool-access protocol. References to MCP are to its public specification and are used for comparison only.

Agent Schema (/agent-schema) All 36 steps → (/inventive-steps)

Define what an autonomous agent is — structurally.

PRIMARY TECHNICAL DISCLOSURE

– Cognition-Compatible Semantic Agent Objects and Structural Validation (/articles/cognition-compatible-semantic-agent-objects-and-structural-validation)

SECONDARY TECHNICAL

- Partial Agent Structural Validity: Fewer Fields, Still Deterministic (/articles/agent-schema/partial-validity)
- Minimum Two-Field Validation Threshold: The Floor of Semantic Structure (/articles/agent-schema/two-field-threshold)
- Field Interaction Rules: Deterministic Constraints Between Canonical Fields (/articles/agent-schema/field-interaction-rules)
- Field-Based Role Typing: Agent Roles Derived From Structural Composition (/articles/agent-schema/role-typing)
- Semantic Templates: Predefined Field Arrangements as Agent Class Contracts (/articles/agent-schema/semantic-templates)
- Structural Scaffolding Logic: Resolving Missing Fields Through Inference or Defaulting (/articles/agent-schema/scaffolding-logic)
- Field-Aware Default Resolution: Deterministic Behavior When Fields Are Absent (/articles/agent-schema/default-resolution)
- Traceable Semantic Lineage Graph: Mutation History Embedded in Agent Objects (/articles/agent-schema/lineage-graph)
- Serialization With Stateless Compatibility: Reconstruction Without External Session State (/articles/agent-schema/stateless-serialization)
- Schema Governance Through Versioned Policies: Cross-Version Structural Interoperability (/articles/agent-schema/versioned-policies)

APPLICATIONS · GENERAL

- Enterprise AI Agent Interoperability Through Canonical Schema (/articles/agent-schema/enterprise-interoperability)
- Robotic System Standardization via Structural Field Composition (/articles/agent-schema/robotic-standardization)
- Multi-Vendor AI Agent Interoperability (/articles/agent-schema/multi-vendor-ai-agents)
- Digital Twin Standardization Through Canonical Fields (/articles/agent-schema/digital-twin-standardization)
- Healthcare AI Agent Portability (/articles/agent-schema/healthcare-agent-portability)
- Defense Coalition Interoperability (/articles/agent-schema/defense-coalition-interop)
- Insurance Claims Processing Through Standard Agents (/articles/agent-schema/insurance-claims-agents)
- Legacy System Integration via Schema Bridging (/articles/agent-schema/legacy-system-integration)

APPLICATIONS · SPECIFIC

- LangChain Built the Agent Framework. It Did Not Define What an Agent Is. (/articles/agent-schema/langchain)
- AutoGen Enabled Multi-Agent Conversations. The Agents Have No Structural Definition. (/articles/agent-schema/autogen)
- CrewAI Organized Agents Into Teams. The Agents Still Have No Schema. (/articles/agent-schema/crewai)
- Semantic Kernel Integrated AI Into Enterprise Code. The Agents It Creates Have No Schema. (/articles/agent-schema/semantic-kernel)
- OpenAI Assistants API Provides Agent Tooling. It Does Not Define Agent Structure. (/articles/agent-schema/openai-assistants)
- Google Vertex AI Agents Provide Managed Agent Infrastructure. The Agents Have No Canonical Schema. (/articles/agent-schema/google-vertex-agents)
- Amazon Bedrock Agents Orchestrate Foundation Models. The Agents Have No Structural Definition. (/articles/agent-schema/amazon-bedrock-agents)
- Haystack Built Composable NLP Pipelines. The Pipeline Components Have No Agent Schema. (/articles/agent-schema/haystack)
- LlamaIndex Built the Data Framework for LLM Applications. The Data Objects Have No Agent Schema. (/articles/agent-schema/llamaindex)
- Dify Made LLM Application Development Visual. The Applications Have No Agent Schema. (/articles/agent-schema/dify)
- Microsoft AutoGen and CrewAI Multi-Agent Frameworks (/articles/agent-schema/autogen-crewai)
- LangChain and LangGraph Agent Framework (/articles/agent-schema/langchain-langgraph)
- LlamaIndex Agent Framework (/articles/agent-schema/llamaindex-agents)
- ROS 2 (Robot Operating System) Middleware (/articles/agent-schema/ros2-robotics)
- Cursor AI-Native Code Editor (/articles/agent-schema/cursor-coding-agent)
- Replit Agent and Replit Workspaces (/articles/agent-schema/replit-agent)
- **Model Context Protocol: Agent Tool Use as Protocol (/articles/agent-schema/anthropic-mcp)**
- Google Agent2Agent: Inter-Agent Communication, Top-Down (/articles/agent-schema/google-a2a)
- AGNTCY: Internet of Agents, by Committee (/articles/agent-schema/cisco-langchain-agntcy)

Agent Schema overview → (/agent-schema)