What are AI agent protocols?

AI agent protocols establish standards of communication among artificial intelligence agents and between AI agents and other systems. These protocols specify the syntax, structure and sequence of messages, along with communication conventions such as the roles agents take in conversations and when and how they respond to messages.

Agent-based AI systems often run in silos. They're built by different providers using diverse AI agent frameworks and employing distinct agentic architectures. Real-world integration becomes a challenge, and coupling these fragmented systems requires tailored connectors for all possible types of agent interaction.

This is where protocols come in. They turn disparate multi-agent systems into an interlinked ecosystem where AI-powered agents share a way of discovering, understanding and collaborating with each other.

While agentic protocols are part of AI agent orchestration, they don't act as orchestrators. They standardize communication but don't manage agentic workflow coordination, execution and optimization.

Benefits of AI agent protocols

AI agent protocols offer these advantages:

- Interoperability
- Reduced agent development complexity
- Standardization and smoother integration

Interoperability

Agent protocols break down silos, allowing agentic AI to communicate with each other regardless of their own underlying implementation. They facilitate seamless agent collaboration across different devices, environments and platforms.

Reduced agent development complexity

Because protocols tackle the intricacies of agentic interaction and abstract away complexities through software development kits (SDKs), they help streamline the process of building multi-agent systems. AI developers can focus more on creating new agent functionalities and enhancing existing ones.

Standardization and smoother integration

AI agent protocols offer a set and structured means of communication. And since many of these standardized protocols are built on top of established technologies, they help ensure compatibility with current technology stacks, making for a smoother enterprise integration.

Your subscription will be delivered in English. You will find an unsubscribe link in every newsletter. You can manage your subscriptions or unsubscribe here. Refer to our IBM Privacy Statement for more information.

Examples of AI agent protocols

Many protocols are still in their early stages, so they have yet to be widely used or applied on a large scale. This lack of maturity means that organizations must be prepared to act as early adopters, adjusting to breaking changes and evolving specifications.

As agentic technology evolves, new protocols might emerge. Here are some current AI agent protocols:

- Agent2Agent (A2A) protocol
- Agent communication protocol (ACP)
- Agent network protocol (ANP)
- Agent-user interaction (AG-UI) protocol
- Agora
- LMOS protocol
- Model context protocol (MCP)

Agent2Agent (A2A) protocol

The A2A protocol is an open standard for AI agent communication initially launched by Google and now managed under the Linux Foundation. It follows a client-server model setup with a three-step workflow:

1. **Discovery** occurs when an entity (a human user or another AI agent) initiates a task request to a client agent, which then looks up remote agents to determine the best fit.
2. Once the client agent identifies a remote agent capable of fulfilling the task, it then goes through **authentication**. The remote agent is responsible for authorization and granting access control permissions.

3. **Communication** proceeds with the client agent sending the task and the remote agent processing it. Agent-to-agent communication happens over HTTPS for secure transport, with JSON-RPC (Remote Procedure Call) 2.0 as the format for data exchange.

Agent communication protocol (ACP)

Like A2A, the Agent Communication Protocol (ACP) is another open standard for agent-to-agent communication, initially introduced by IBM's BeeAI and now part of the Linux Foundation.

Its main components include an ACP client and an ACP server. The ACP client sends requests to the ACP server through a RESTful API over HTTP. The ACP server hosts one or more agents behind a single HTTP endpoint and routes tasks to the appropriate agent.

Here are ACP's other key features:

- The protocol can be used with standard HTTP tools like Postman or even a browser, but SDKs are also available.
- Discovery can happen online by querying ACP servers directly and public manifest files at well-known URLs. Offline discovery occurs through a centralized registry or by embedding agent metadata directly into their distribution packages.
- ACP accepts different message types, such as audio, images, text, video or custom binary formats.

Agent network protocol (ANP)

The Agent Network Protocol (ANP) is an open-source protocol whose goal is to be "the HTTP of the agentic web era." As such, it employs HTTP for data transport and JSON-LD (JSON for Linked Data) for data formatting.

ANP adopts a peer-to-peer architecture composed of three layers:

- The **identity layer** implements both end-to-end encryption for secure communication and decentralized identity authentication based on the W3C DID (Decentralized Identifiers) standard.
- The **meta-protocol layer** lets agents negotiate and agree on how to communicate.
- The **application protocol layer** allows autonomous agents to describe their capabilities and provides support for agent discovery.

Agent-User Interaction (AG-UI) protocol

The Agent-User Interaction (AG-UI) protocol intends to standardize how back-end AI agents connect to front-end or user-facing applications. It's designed for real-time human-agent interaction like chatting with AI assistants and chatbots, live streaming state updates and other agentic automations entailing a human-in-the-loop approach.

AG-UI's event-driven architecture lets AI agents produce events based on certain system triggers or inputs from users. The protocol defines a number of event categories, including those for sending and

receiving messages, tool calling and accomplishing tasks.

Its middleware layer supports several transport methods, such as server-sent events (SSE), webhooks and WebSockets. AG-UI also allows for a secure proxy to route requests securely between agents and user interfaces.

Agora

Agora is a communication protocol for agents powered by large language models (LLMs). It relies on a few core LLM agent capabilities: natural language understanding, instruction following, writing and running code and autonomous negotiation.

LLM agents can implement and support their own protocols, which they describe in plain text within a protocol document. The first part of the document contains metadata identifying the protocol name, description and whether it's for a single or multiple rounds of conversation. The second part outlines how communication takes place, with instructions in a mix of natural language and code. Agents are then left to autonomously negotiate which protocol to adopt.

Agora employs HTTPS for transmitting data and JSON for formatting. It also uses a hash-based identification system for protocol documents.

LMOS protocol

Developed by the Eclipse Foundation, the Language Model Operating System (LMOS) protocol aims to usher in an Internet of Agents (IoA), a multi-agent ecosystem on an internet scale. Similar to ANP, its structured architecture consists of three layers:

- The **identity and security layer** provides encrypted communication and supports different authentication schemes, such as W3C DID and OAuth 2.0.
- The **transport protocol layer** allows agents to choose and adapt the transport protocol that suits their purpose for each interaction.
- The **application protocol layer** outlines formats for agent and tool descriptions, discovery methods, a semantic data model and a websocket subprotocol.

The LMOS protocol uses JSON-LD to describe tool and agent capabilities and other metadata. Discovery happens either dynamically, by querying a central directory or through decentralized networks.

Model context protocol (MCP)

Introduced by Anthropic, the Model Context Protocol (MCP) provides a standardized way for AI models to get the context they need to carry out tasks. In the agentic realm, MCP acts as a tier for AI agents to connect and communicate with external services and tools, such as APIs, databases, files, web searches and other data sources.

MCP encompasses these three key architectural elements:

- The **MCP host** contains orchestration logic and can connect each MCP client to an MCP server. It can host multiple clients.

- An **MCP client** converts user requests into a structured format that the protocol can process. Each client has a one-to-one relationship with an MCP server. Clients manage sessions, parse and verify responses and handle errors.

- The **MCP server** converts user requests into server actions. Servers are typically GitHub repositories available in various programming languages and provide access to tools. They can also be used to connect LLM inferencing to the MCP SDK through AI platform providers such as IBM and OpenAI.

In the transport layer between clients and servers, messages are transmitted in JSON-RPC 2.0 format using either standard input/output (stdio) for lightweight, synchronous messaging or SSE for asynchronous, event-driven calls.

Factors to consider when choosing an AI agent protocol

With the lack of benchmarks for standardized evaluation, enterprises must conduct their own appraisal of the protocol that best fits their business needs. They might need to start with a small and controlled use case combined with thorough and rigorous testing.

Here are a few aspects to keep in mind when assessing agent protocols:

- Efficiency
- Reliability
- Scalability
- Security

Efficiency

Ideally, protocols are designed to limit latency, resulting in swift data transfer and rapid response times. While some communication overhead is expected, it must be kept to a minimum.

Reliability

AI agent protocols must be able to handle changing network conditions throughout agentic workflows, with mechanisms in place to manage failures or disruptions. For instance, ACP is designed with asynchronous communication as the default, which suits complex or long-running tasks. Meanwhile, A2A supports real-time streaming using SSE for large or long outputs or continuous status updates.

Scalability

Protocols must be robust enough to cater to growing agent ecosystems without a decline in their performance. Evaluating scalability can include increasing the number of agents or links to external tools over a period of time, either gradually or suddenly, to observe how a protocol operates in those conditions.

Security

Maintaining security is paramount, and agent protocols are increasingly incorporating safety guardrails. These include authentication, encryption and access control.