Standardized communication in AI agent ecosystems: IBM's ACP and Google's A2A

IBM's Agent Communication Protocol (ACP) and Google's Agent2Agent Protocol (A2A) both highlight the growing need for standardized communication mechanisms to facilitate interoperability and collaboration between diverse AI agents.

IBM's Agent Communication Protocol (ACP)

* Open Standard: ACP is an open standard designed to enable seamless communication between AI agents, regardless of their underlying framework, programming language, or runtime environment.
* Benefits: It helps to overcome challenges of framework diversity, reduce reliance on custom integrations, and facilitate cross-organizational workflows in multi-agent systems.
* BeeAI & Linux Foundation: ACP is developed under the Linux Foundation's BeeAI project, promoting open governance and community participation.
* Features: ACP features include REST-based communication, multimodal message support, asynchronous communication by default (with synchronous support), and offline/secure agent discovery.

Google's Agent2Agent (A2A) protocol

* Open Protocol: A2A is an open protocol providing a standard way for AI agents to collaborate, irrespective of the underlying framework or vendor.
* Key Capabilities: A2A's central capabilities include capability discovery using "Agent Cards" (JSON files describing agent capabilities), task management, collaboration through message exchange, and user experience negotiation for optimal content display.
* Complementary to MCP: A2A is complementary to Anthropic's Model Context Protocol (MCP), where MCP helps connect agents to tools and data, while A2A enables agents to communicate and coordinate actions.

The importance of standardized agent communication

* Breaking Down Silos: Both ACP and A2A aim to bridge the fragmentation in the current agent ecosystem, moving away from isolated systems towards a more interconnected network of agents.
* Facilitating Collaboration: Standardized communication allows agents developed in different environments to understand each other and work together on complex tasks, leveraging their collective intelligence.
* Enhancing Scalability and Adaptability: Standardized protocols enable easier integration of new agents and services, supporting dynamic updating and enhancing system scalability and adaptability.
* Improving Security and Auditability: Standardized protocols, like A2A's use of HTTPS and support for authentication schemes, contribute to enterprise-grade security and improve observability for auditing and compliance.

In essence, both IBM's ACP and Google's A2A are significant steps towards establishing a standardized language for AI agent communication, paving the way for more robust, scalable, and interoperable multi-agent systems

<https://developers.googleblog.com/en/a2a-a-new-era-of-agent-interoperability/>

```mermaid

graph TD

subgraph User Interaction

User[👤 User]

end

subgraph Primary Agent Layer

Orchestrator[🤖 Primary Orchestrator Agent]

end

subgraph A2A Ecosystem

direction LR

A2A\_Server[🌐 A2A Server/Registry]

subgraph Specialized Agents

IT\_Agent[👩‍💻 IT Provisioning Agent]

Finance\_Agent[💰 Finance Agent]

end

end

subgraph MCP and Tools Layer

direction LR

subgraph IT Tools

MCP\_Server\_IDM[🔌 MCP Server: Identity API] --> Tool\_IDM[🛠️ Tool: Okta/Azure AD API]

MCP\_Server\_Hardware[🔌 MCP Server: Hardware DB] --> Tool\_Hardware[🛠️ Tool: Asset DB]

end

subgraph Finance Tools

MCP\_Server\_Payroll[🔌 MCP Server: Payroll API] --> Tool\_Payroll[🛠️ Tool: ADP/Gusto API]

MCP\_Server\_Billing[🔌 MCP Server: Billing System] --> Tool\_Billing[🛠️ Tool: Stripe API]

end

end

%% --- Connections ---

%% 1: User Request

User -- "1: 'Onboard new employee'" --> Orchestrator

%% 2: Agent Discovery via A2A

Orchestrator -- "2: Discover Agents" --> A2A\_Server

A2A\_Server -- "3: Return Agent Cards" --> Orchestrator

%% 4: Task Delegation via A2A

Orchestrator -- "4a. A2A Task: 'Provision IT account'" --> IT\_Agent

Orchestrator -- "4b. A2A Task: 'Set up payroll'" --> Finance\_Agent

%% 5: Agent Uses Tools via MCP

IT\_Agent -- "5a. MCP Client Request" --> MCP\_Server\_IDM

IT\_Agent -- "5b. MCP Client Request" --> MCP\_Server\_Hardware

Finance\_Agent -- "5c. MCP Client Request" --> MCP\_Server\_Payroll

%% 6: Results return

Tool\_IDM -- "6a. Result" --> MCP\_Server\_IDM

MCP\_Server\_IDM -- "7a. MCP Response" --> IT\_Agent

Tool\_Hardware -- "6b. Result" --> MCP\_Server\_Hardware

MCP\_Server\_Hardware -- "7b. MCP Response" --> IT\_Agent

Tool\_Payroll -- "6c. Result" --> MCP\_Server\_Payroll

MCP\_Server\_Payroll -- "7c. MCP Response" --> Finance\_Agent

%% 8: A2A Task Completion

IT\_Agent -- "8a. A2A Task Complete" --> Orchestrator

Finance\_Agent -- "8b. A2A Task Complete" --> Orchestrator

%% 9: Final Response to User

Orchestrator -- "9: 'Onboarding complete!'" --> User

%% --- Styling ---

classDef agent fill:#D6EAF8,stroke:#5DADE2,stroke-width:2px;

classDef tool fill:#E9F7EF,stroke:#58D68D,stroke-width:2px;

classDef protocol fill:#FDEDEC,stroke:#F1948A,stroke-width:2px;

class User,Orchestrator,IT\_Agent,Finance\_Agent agent;

class A2A\_Server protocol;

class MCP\_Server\_IDM,MCP\_Server\_Hardware,MCP\_Server\_Payroll,MCP\_Server\_Billing protocol;

class Tool\_IDM,Tool\_Hardware,Tool\_Payroll,Tool\_Billing tool;

```

Of course. Here is an explanation of how the Model Context Protocol (MCP) server/client, the Agent-to-Agent (A2A) server, agents, and tools fit together in a modern AI architecture, complete with references.

### **The Big Picture: Vertical vs. Horizontal Communication**

Think of your AI architecture in two directions:

* **Vertical Communication (Agent-to-Tool):** This is about a *single* AI agent reaching out to get information or perform a specific task. This is where **MCP** and **Tools** come in.
* **Horizontal Communication (Agent-to-Agent):** This is about multiple, independent AI agents collaborating and delegating tasks to each other. This is the role of the **A2A protocol**.

Here's a breakdown of how each component fits into this model:

### **1. Agents**

An **agent** is the core reasoning engine of your system. It's typically a Large Language Model (LLM) that can understand goals, make plans, and decide what actions to take. However, an agent on its own is limited to the knowledge it was trained on. To be effective, it needs to interact with the outside world.

### **2. Tools**

**Tools** refer to the external functions and data sources that an agent can utilize. They are what make an agent useful in the real world. Examples of tools include:

* **Functions:** send\_email(), query\_database(), get\_current\_stock\_price()
* **APIs:** A connection to a weather API, a Salesforce API, etc.
* **Data Sources:** A file system, a vector database, or a knowledge base.

Without a standardized way to connect to these tools, developers would have to write custom code for every single integration, which is inefficient and hard to maintain.

### **3. MCP (Model Context Protocol) Server/Client**

This is where the **Model Context Protocol (MCP)** comes in. MCP is a standardized protocol that acts as a universal adapter for connecting an agent to tools and data.

* **MCP Server:** You would wrap each of your tools or data sources in a lightweight MCP server. This server's job is to expose the tool's functionality in a standardized way. For example, you could have an MCP server for your company's user database.
* **MCP Client:** The agent uses an MCP client to discover and communicate with these MCP servers. The client asks the server what it can do and then calls the appropriate functions.

**In the architecture, MCP creates a secure and standardized "vertical" bridge between a single agent and its toolbox.** The agent doesn't need to know the specific details of how each tool's API works; it just needs to know how to speak MCP.

**Analogy:** Think of MCP as a universal power adapter. Your agent is a device, and the tools are all the different power outlets in the world. MCP allows your device to plug into any outlet without needing a custom adapter for each one.

### **4. A2A (Agent-to-Agent) Server and Protocol**

While MCP is great for giving one agent its tools, complex tasks often require the collaboration of multiple specialized agents. This is where the **Agent-to-Agent (A2A) protocol** is essential. A2A is a standard for "horizontal" communication, allowing different agents to discover each other, delegate tasks, and work together.

* **A2A Server (or Registry):** An A2A server acts as a directory or "yellow pages" for agents. When an agent comes online, it publishes an "Agent Card" to the A2A server. This card describes what the agent does and how to communicate with it.
* **A2A Protocol:** When one agent (the "client" agent) needs a task done that's outside its expertise, it can query the A2A server to find another agent (the "remote" agent) that can help. It then uses the A2A protocol to send a task request to that agent.

**In this architecture, A2A allows for the creation of sophisticated, multi-agent systems where tasks are delegated to the most suitable agent.**

**Analogy:** Think of A2A as a project management platform for a team of specialists. One agent, the project manager, doesn't know how to do everything. But it knows how to find the right team member (the designer, the engineer, the writer) and assign them tasks.

### **How It All Fits Together: An Example**

Imagine an HR onboarding system:

1. A user interacts with a primary **"Onboarding Orchestrator" Agent** and says, "Onboard our new hire, Jane Doe."
2. The Orchestrator Agent realizes it needs to create an IT account. It uses the **A2A protocol** to find and delegate a task to a specialized **"IT Provisioning" Agent**.
3. The IT Provisioning Agent accepts the task. To actually create the account, it needs to interact with the company's identity management system. It uses an **MCP client** to connect to an **MCP server** that wraps the identity management API (the **Tool**).
4. The IT Agent uses the tool via MCP to create the account and then reports back to the Orchestrator Agent via A2A that the task is complete.

In this flow:

* **A2A** handled the high-level collaboration between the orchestrator and the specialist.
* **MCP** handled the low-level, secure connection from the specialist agent to its specific tool.

### **References**

Here are some excellent resources for a deeper dive:

* **MCP vs. A2A Explained:** Clarifai and Composio have published clear articles that break down the differences and synergies between the two protocols.
* **Model Context Protocol Official Documentation:** The official MCP website ([modelcontextprotocol.io](https://modelcontextprotocol.io/)) provides the protocol specification, architectural overviews, and examples.
* **Agent-to-Agent (A2A) Project:** The A2A project on GitHub ([github.com/a2aproject/A2A](https://github.com/a2aproject/A2A)) is the home for the A2A protocol and includes sample implementations.
* **IBM's Explanation of Agent Communication Protocol (ACP):** IBM has a great article on ACP, which is conceptually similar to A2A and highlights the need for standardized agent communication.