Orchestrating Collective Intelligence with Google A2A

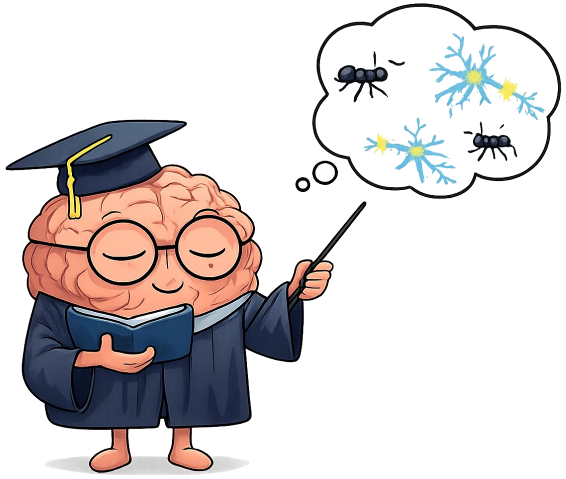
K.A Ashan Priyadarshana
Senior Tech Lead | Sysco LABS

Get the
Github Repo

Intelligence Is Not Built — It Emerges

Is intelligence something you own, or something that emerges?

- A system is not the sum of its parts
- Intelligence emerges from **interaction**, not isolation
- Boundaries define *what can interact* — and what cannot
- Many simple agents can outperform one powerful mind
- Coordination creates intelligence, not complexity



We often think intelligence is something you *install* — more data, bigger models, faster GPUs. But nature teaches us something different. A single ant is not intelligent, A colony is. A neuron alone cannot think, but billions interacting can create consciousness.

So intelligence doesn't live *inside* components… It lives **between them**.

Emergence needs structure → coordination → A2A

A2A (Agent-to-Agent) Protocol is an open standard that enables seamless, direct communication among AI agents.

- Intelligence emerges when agents interact — but interaction alone is not enough
- Without structure, interactions become noise, not intelligence
- Coordination turns emergence into *reliable outcomes*
- Boundaries define *who can talk, when, and how*
- A2A provides the structure that lets intelligence scale safely

A2A is the coordination layer for intelligent systems. It defines *how* agents communicate, collaborate, and align — without exposing their internal logic or losing autonomy.

Emergence becomes useful only when it is orchestrated.

A2A vs. MCP – How Do They Work Together?

A2A and Model Context Protocol (MCP) address different layers of agent functionality – they complement rather than replace each other.

- **MCP (Model Context Protocol):** Standardizes how an agent uses tools/APIs – it's an *agent-to-tool* communication layer
- **A2A (Agent-to-Agent):** Standardizes how agents talk to other agents – an *agent-to-agent* communication layer spanning systems
- **Stateless vs. Dialog:** MCP tools are like stateless functions, whereas A2A enables stateful, multi-turn conversations between agents
- **Complementary Usage:** An agent can use MCP internally for its tools and use A2A externally to collaborate with peer agents
- **Beyond "Agent-as-Tool":** Encapsulating an entire agent as a mere MCP tool is limiting – A2A treats each agent as a full actor with its own tasks, unlocking richer interactions

MCP and A2A are complementary. MCP allows an agent to access services or data (e.g., web search, database query). A2A, the "Internet for agents," enables communication when an agent needs help from another agent (e.g., asking a scheduling agent to handle bookings), crossing boundaries like different apps or organizations. They work together: MCP connects to tools internally; A2A connects to other intelligent agents externally.

Key Concepts of A2A

Understanding A2A's building blocks is crucial for implementing or integrating with the protocol.

- **Roles – Client & Server:** The *A2A Client* (a user-facing app or agent) initiates requests, and the *A2A Server* (remote agent) processes requests and returns results

- **Agent Card:** A JSON "business card" describing an agent – includes its identity, endpoint URL, declared skills/capabilities, and auth requirements for clients

- **Task:** A stateful unit of work with a unique ID and lifecycle. Long-running or multi-step operations are tracked as tasks so progress and results can be managed over time

- **Message & Parts:** A single communication turn. A message has a role ("user" or "agent") and contains one or more *Parts* – e.g. text content, file attachments, or JSON data – allowing rich information exchange

- **Artifact:** A concrete output produced by an agent during a task (e.g. a generated document, image, or dataset). Artifacts have their own IDs and can be streamed to the client as they are created

An analogy: The Agent Card is an agent's API documentation, detailing "who I am, what I can do, and how to talk to me." Messages are single Q&A interactions, while a Task is a longer job or ticket. Artifacts are the structured, ID'd end-products (files/data). Remember the roles: the A2A client is the orchestrator (agent or app), and the A2A server is the service-providing agent.

How Do Agents Discover Each Other?

Before agents can collaborate, they need to find and trust each other. A2A defines flexible discovery methods:

- **Well-Known URL:** Agents can publish their Agent Card at a standard path (/.well-known/agent-card.json) on their domain. A client agent that knows the domain can simply GET this URL to retrieve the card
- **Registry/Directory:** In enterprise or marketplaces, Agent Cards can be listed in a central registry. Client agents query the registry (e.g. “find agents with a ‘booking’ skill”) and get back matching Agent Cards
- **Direct Configuration:** In closed environments, discovery can be manual – e.g. a client is pre-configured with specific Agent Card URLs or JSON files. This works for fixed agent networks or testing purposes
- **Security of Discovery:** Agent Cards may contain sensitive info (internal endpoints, capabilities). A2A supports *authenticated cards* – servers can require OAuth tokens or mTLS to fetch the card, or serve a limited public card and a detailed one to authorized clients
- **Protecting Cards:** Best practices include hosting the card over HTTPS, using access controls (IP allowlists, auth headers), and never embedding secrets directly in the Agent Card JSON

Agent discovery, crucial for agent networking, uses the standard URL approach (Agent B's Agent Card at /.well-known/agent-card.json on known domains). For large scale, a central registry acts like an app store, enabling capability-based queries (e.g., "weather data"). In controlled environments, agents can be pre-configured with partner info. A2A emphasizes standard web security (e.g., authentication) to secure non-public Agent Cards and sensitive services, ensuring only trusted agents connect.

Life of a Task in A2A

A2A enables dynamic, stateful interactions. An agent's request to another may result in an immediate response or a longer task lifecycle.

- **Immediate Response (Message)**: For simple queries, the remote agent responds directly with a one-off Message, completing the interaction in one turn.
- **Long-Running Task**: For requests requiring more work, the agent creates a Task (with a unique taskId), signaling that work has begun and allowing for tracking until completion.
- **Context ID**: A contextId is issued to group related interactions, maintaining a shared conversational state for all follow-up messages and related sub-tasks.
- **Follow-ups and Input**: The client continues the conversation by sending new messages referencing the same contextId (and taskId). This enables multi-turn workflows, handling clarifications (input-required state), or providing incremental results.
- **Task Lifecycle**: Tasks progress through states (e.g., working, completed, failed, input-required). Once a task reaches a terminal state (e.g., completed or failed), it is immutable. Refinements or retries start a new task, often within the same context for continuity.

An example: Agent A requests a report from Agent B, receiving a taskId (e.g., #123). A contextId links follow-up queries to the ongoing work, like a conversation thread. Agent B sends updates or partial results as artifacts. If more input is needed, the task enters an input-required state. Once Agent B completes #123, any new work requires a new task (e.g., #124) but uses the same contextId for continuity. This ensures clear task records for tracking and debugging complex multi-step operations.

How Agents Coordinate via A2A

A2A enables one agent to act as a coordinator, delegating subtasks to other agents and combining their results.

- **Task Delegation:** Agents can offload parts of a job to other agents by sending them A2A requests. This enables complex workflows that no single agent could handle alone
- **Orchestration Example:** A user’s assistant agent (client) can invoke a flight-booking agent, a hotel-booking agent, and a tours agent in parallel. It then compiles all their outputs into one final answer for the user
- **Multi-Turn Interaction:** Agents converse in natural, multi-turn fashion. For instance, if details are unclear, the remote agent can ask a clarifying question (via A2A) instead of failing silently – they essentially “negotiate” the task requirements
- **Opaque Collaboration:** Each agent remains a black box to the others – they only see messages and artifacts, not the inner logic or data of their peers. This opacity means agents can cooperate without exposing proprietary methods or sensitive data
- **Secure Communication:** All coordination happens over secure web channels (HTTPS) with proper authentication. Agents from different teams or organizations can thus safely work together using standard auth tokens and encryption, just as regular web services do

Revisiting the travel planning scenario, the assistant agent orchestrates the process. It breaks the user request into subtasks (flights, hotels, tours) and dispatches them to specialist agents via A2A. Agents can dialogue; the hotel agent might ask the orchestrator for clarifying information (e.g., preferred dates). The orchestrator consolidates results for the user. Crucially, specialized agents cooperate purely via A2A, maintaining security without needing direct access to each other's data or tools. All A2A calls are standard, secure HTTPS calls with authentication (like OAuth), enforcing enterprise security policies. This is coordinated intelligence: multiple AI agents jointly solving complex tasks.

Making an ADK Agent A2A-Compatible

How can you prepare an existing agent (built with Google's ADK or similar) to speak A2A? It's more about adding a communication layer than rewriting logic.

- **Expose an A2A Endpoint**: Run your agent as an HTTP service by implementing the A2A API (start an A2A Server) to receive sendMessage requests.
- **Provide an Agent Card**: Define and publish your agent's Agent Card (JSON) with its name, description, skills, endpoint URL, and required authentication for others to discover its offerings.
- **Use ADK's A2A Tools**: Google ADK provides utilities, such as a single call to "wrap" your ADK agent with an A2A interface and auto-generate the Agent Card.
- **No Core Changes**: Your agent's core logic and Model Context Protocol usage remain unchanged. A2A acts as an overlay, handling message and task packaging for communication.
- **Verify and Test**: Confirm the Agent Card is accessible and other agents can reach the endpoint. Test a sample A2A interaction (e.g., using an A2A client SDK or orchestrator) to ensure correct responses.

Converting an existing ADK agent to A2A is usually straightforward using **to_a2a()** in Python, which automatically spins up an A2A-compatible server and generates an Agent Card by introspecting the agent's skills. Alternatively, you can manually write the Agent Card JSON. The agent receives a network endpoint (URL) for incoming A2A calls. As A2A is only a communication standard, the agent's existing abilities (e.g., using tools via MCP) remain available. After setup, it is crucial to test: fetch the Agent Card and call a skill using client libraries or HTTP requests. A successful response, perhaps returning a Task with an artifact, confirms the agent is integrated into the A2A ecosystem.

Key Takeaways & Next Steps

We've covered the fundamentals of Google's A2A protocol. Here are the main points and what to do next:

- **Practical & Familiar:** A2A is built on web standards (HTTP + JSON), so web developers can pick it up easily – no exotic protocols required

- **Open and Supported:** It's an open-source standard (Linux Foundation project). Official SDKs in multiple languages (Python, JS, Java, etc.) are available to help you build A2A agents quickly

- **Enterprise-Ready:** A2A was designed with real-world needs in mind – it mandates HTTPS for all agent communication and integrates with standard auth (OAuth2/OpenID), tracing, and monitoring tools for security and observability

- **Focus on Logic, Not Plumbing:** By standardizing agent communication, A2A removes a lot of custom "glue code." You can concentrate on your agents' unique reasoning and capabilities, rather than writing networking boilerplate

The Agent2Agent (A2A) protocol enables the creation of composable AI systems by giving agents a standard communication method (like a shared phone line). Using familiar tech (HTTP, JSON), A2A avoids the need for new frameworks. Its open standard fosters a healthy community and prevents vendor lock-in. Enterprises can adopt A2A confidently, as it integrates with existing security, monitoring, and API management. For developers, this means less time on integrations and more time building intelligent agent behaviors.