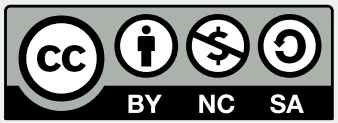
Agent2Agent Protocol (A2A)

Protocolo de comunicación entre agentes de IA

Autor: Alexander Saravia

Protocolos Agénticos

21 de noviembre de 2025



Agenda

1. Introducción
   - Qué es y qué no es A2A
   - Cómo A2A complementa a MCP
2. Conceptos fundamentales
   - Actores y elementos
3. Estructuras de datos A2A
   - Agent Cards
   - Mensajes, Parts y Artifacts
4. Descubrimiento, empresa y ciclo de vida
   - Agent Discovery
   - Enterprise Features
   - Life of a Task
5. Extensiones y operaciones asíncronas
   - Extensiones
   - Streaming & Asynchronous Operations

Introducción

Motivación: por qué A2A

- Ecosistemas de agentes cada vez más heterogéneos:
  - Diferentes frameworks (ADK, LangGraph, CrewAI, …).
  - Distintos proveedores, despliegues y dominios.
- Integraciones ad-hoc y punto a punto:
  - Alto acoplamiento, difícil reutilización.
  - Complejo de auditar, asegurar y escalar.
- Necesidad de **interoperabilidad estandarizada**:
  - Que agentes independientes colaboren como pares.
  - Que soporten tareas de larga duración, streaming y flujos multi-turno.

Introducción

Qué es y qué no es A2A

Qué es A2A

Definición

El **Agent2Agent Protocol (A2A)** es un estándar abierto para habilitar la comunicación y colaboración entre agentes de IA, incluso cuando han sido construidos con distintos frameworks y por diferentes organizaciones.

- Proporciona un **lenguaje común** para que los agentes:
  - Se descubran y describan (*Agent Cards*).
  - Intercambien mensajes, tareas y artefactos.
  - Gestionen el ciclo de vida de tareas de forma trazable.
- Se apoya en tecnologías web existentes:
  - Transporte HTTP(S).
  - Mensajería basada en JSON-RPC 2.0.
  - Streaming con SSE y notificaciones push.
- Diseñado para **colaboración entre agentes** en contextos empresariales y multi-organización.

Qué **no** es A2A

- **No es** un framework de desarrollo de agentes:
  - No reemplaza a ADK, LangGraph, CrewAI, etc.
  - Se limita a definir el *protocolo de comunicación*.
- **No es** un protocolo de conexión a herramientas:
  - Ese rol lo cubre el **Model Context Protocol (MCP)**.
  - A2A no describe funciones ni herramientas individuales.
- **No es** una API de modelos:
  - No define cómo invocar, entrenar o configurar LLMs.
  - Los modelos son internos al agente (caja negra).
- **No decide** la lógica interna del agente:
  - No expone memoria, prompts ni herramientas internas.
  - Solo estandariza mensajes, tareas y artefactos entre agentes.

Introducción

Cómo A2A complementa a MCP

Cuadro: Comparativa de enfoque entre MCP y A2A

	MCP	A2A
Enfoque principal	Conectar modelos/agentes a **herramientas y recursos** (APIs, BD, files).	Estandarizar la **colaboración entre agentes** como pares.
Unidad de interacción	Llamadas a herramientas, recursos y prompts tipados.	Mensajes, tareas y artefactos entre agentes remotos.
Visibilidad	El host ve herramientas y recursos expuestos por el servidor MCP.	El cliente ve al agente remoto como *caja negra* con capacidades declaradas.
Casos de uso típicos	Un LLM llamando una API externa o consultando una base de datos.	Un orquestador que delega sub-tareas en varios agentes especializados.
Complementariedad	Un agente A2A puede usar MCP internamente para acceder a herramientas y datos.	A2A permite que múltiples agentes (cada uno con su MCP) colaboren para completar tareas complejas.

*Idea clave: MCP conecta agentes con **herramientas**; A2A conecta **agentes entre sí**.*

A2A en la pila de agentes

- **Modelos**: LLMs y otros modelos base para razonamiento.
- **Frameworks de agentes**: ADK, LangGraph, CrewAI, etc.
- **MCP**: acceso tipado a datos, herramientas y recursos.
- **A2A**: comunicación estandarizada entre agentes externos.

Principio

A2A no compite con MCP ni con los frameworks: los **orquesta** a nivel de ecosistema multi-agente y multi-organización.

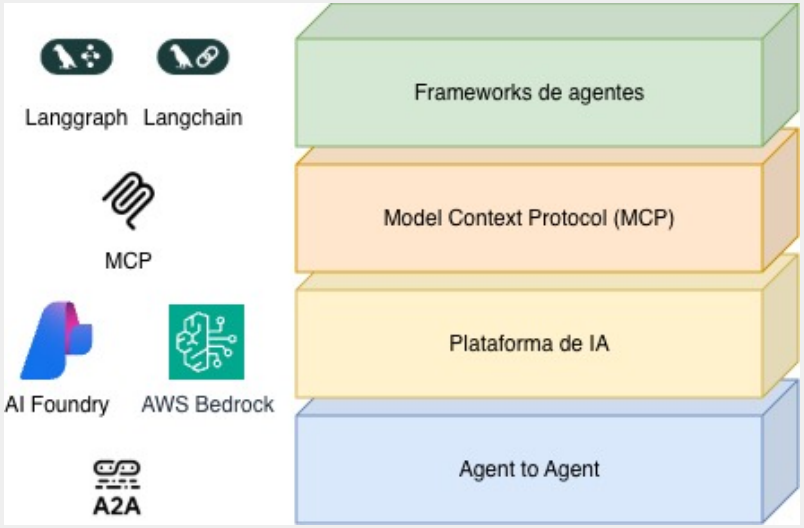


Figura: A2A en la pila: modelos, frameworks, MCP y A2A.

Conceptos fundamentales

Conceptos fundamentales

Actores y elementos

Core Actors in A2A Interactions

- **User**:
  - Persona o servicio que inicia una petición o define un objetivo.
- **A2A Client (Client Agent)**:
  - Agente que actúa en nombre del usuario.
  - Descubre agentes remotos y se comunica usando A2A.
- **A2A Server (Remote Agent)**:
  - Agente o sistema agéntico que expone un endpoint HTTP A2A.
  - Procesa mensajes/tareas y devuelve resultados o estados.
  - Es una **caja negra**: no expone su implementación interna.

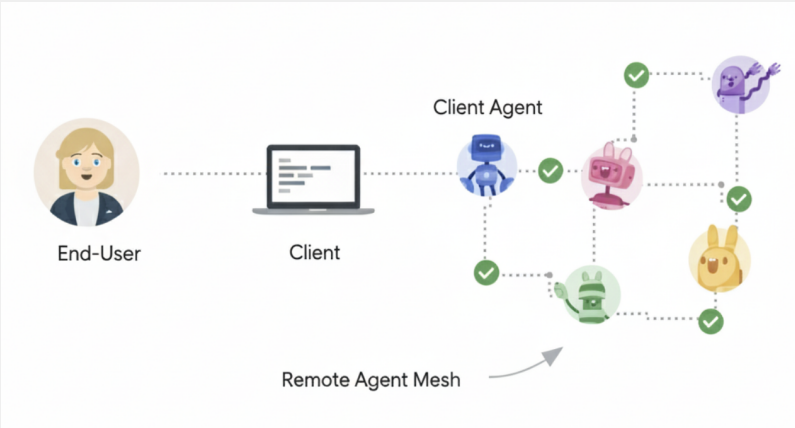


Figura: Relación User, A2A Client y A2A Server.

Cuadro: Elementos de comunicación fundamentales en A2A

Elemento	Descripción	Propósito clave
Agent Card	Documento JSON que describe identidad, endpoint, capacidades, autenticación y *skills* del agente.	Permite que los clientes descubran agentes y comprendan cómo interactuar de forma segura.
Task	Unidad de trabajo con estado, identificada por un `taskId` y asociada a un `contextId`.	Modelar operaciones de larga duración y permitir seguimiento de progreso.
Message	Un turno de comunicación entre cliente y agente (roles `user`/`agent`).	Intercambiar instrucciones, contexto, aclaraciones o respuestas.
Part	Contenedor granular de contenido: `TextPart`, `FilePart`, `DataPart`, etc.	Soportar múltiples modalidades (texto, archivos, datos estructurados).
Artifact	Resultado tangible generado por un agente durante una tarea.	Entregar salidas estructuradas y recuperables (documentos, imágenes, datos).

Estos elementos se combinan para modelar conversaciones y flujos de trabajo multi-agente.

Interaction Mechanisms

- **Request/Response (Polling)**:
  - El cliente envía una petición y recibe una respuesta.
  - Para tareas largas, el cliente consulta periódicamente el estado.
- **Streaming con Server-Sent Events (SSE)**:
  - El cliente abre un stream con `message/stream`.
  - El servidor envía `Task`, `TaskStatusUpdateEvent` y `TaskArtifactUpdateEvent` en tiempo real.
- **Push Notifications**:
  - El cliente registra un webhook mediante `PushNotificationConfig`.
  - El servidor envía notificaciones asíncronas para actualizaciones relevantes de tareas muy largas.

Estructuras de datos A2A

Estructuras de datos A2A

Agent Cards

Idea general

El **Agent Card** es la tarjeta de presentación JSON de un agente A2A. Describe qué hace, cómo se invoca y cómo proteger la comunicación.

- Información típica:
  - **Identidad**: `name`, `description`, `provider`.
  - **Endpoint**: `url` del servicio A2A.
  - **Capacidades**: `capabilities.streaming`, `capabilities.pushNotifications`, extensiones soportadas.
  - **Autenticación**: esquemas (`Bearer`, `OAuth2`, ...).
  - **Skills**: tareas que el agente puede realizar, con modos de entrada/salida.
- Es la base para:
  - Descubrimiento de agentes.
  - Negociación de capacidades.
  - Configuración de clientes A2A.

Agent Card (ejemplo simplificado)

Listing 1: Ejemplo simplificado de Agent Card

```
{
  "name": "Travel Planner Agent",
  "description": "Planificador de viajes internacional",
  "version": "0.1.0",
  "url": "https://travel.example.com/a2a",
  "capabilities": {
    "streaming": true,
    "pushNotifications": true,
    "extensions": [
      {
        "uri": "https://example.com/ext/traceability/v1",
        "required": false
      }
    ]
  },
  "security": [
    {
      "type": "oauth2",
      "scopes": ["travel.read", "travel.write"]
    }
  ],
  "skills": [
    {
      "id": "plan-trip",
      "name": "Planificar viaje",
      "description": "Orquesta vuelo, hotel y actividades.",
      "inputModes": ["application/json"],
      "outputModes": ["application/json"]
    }
  ]
}
```

Estructuras de datos A2A

Mensajes, Parts y Artifacts

Messages and Parts

- Un **Message** representa un turno de comunicación:
  - Rol: `üser" o .ªgent"`.
  - Identificador único: `messageId`.
  - Puede incluir `contextId` y referencias a tareas previas.
  - Contiene una lista de **Parts**.
- **Parts** son contenedores de contenido:
  - `TextPart`: texto plano.
  - `FilePart`: archivo (inline Base64 o vía URL).
  - `DataPart`: datos JSON estructurados.
- Diseño **independiente de modalidad**:
  - Permite mezclar texto, archivos, tablas, etc.
  - Facilita extensiones futuras sin romper el protocolo base.

Ejemplo de Message con Parts

Listing 2: Message con TextPart y DataPart

```
{
  "messageId": "msg-123",
  "role": "user",
  "contextId": "ctx-789",
  "parts": [
    {
      "kind": "TextPart",
      "text": "Planifica un viaje a Tokio en octubre."
    },
    {
      "kind": "DataPart",
      "mimeType": "application/json",
      "data": {
        "budget": 3000,
        "preferences": ["museos", "comida local"]
      }
    }
  ]
}
```

Artifacts

- Un **Artifact** es un resultado tangible generado durante el procesamiento de una tarea.
- Características:
  - `artifactId` único.
  - Nombre legible para humanos.
  - Uno o varios **Parts** (igual que en los mensajes).
- Relación con el ciclo de vida de la tarea:
  - Pueden emitirse de forma incremental (streaming).
  - Quedan asociados a un `taskId` y `contextId`.
- Ejemplos:
  - Itinerario de viaje en PDF.
  - Tabla de costos en CSV / JSON.
  - Imagen generada por un agente creativo.

Agent Response: Task or Message

- Un agente puede responder a un mensaje de dos formas:
  - **Message** (stateless): respuesta inmediata y acotada.
  - **Task** (stateful): trabajo de mayor duración con estado.
- Tipos de agentes:
  - **Message-only**: siempre responden con mensajes.
  - **Task-generating**: siempre responden con tareas.
  - **Hybrid**: negocian con mensajes y luego crean tareas.
- Beneficios:
  - Separar interacciones triviales de flujos complejos.
  - Facilitar trazabilidad, reintentos y observabilidad.

Descubrimiento, empresa y ciclo de vida

Descubrimiento, empresa y ciclo de vida

Agent Discovery

Objetivo

Permitir que un A2A Client encuentre agentes remotos y conozca sus capacidades a partir de sus Agent Cards.

- **Well-Known URI**:
  - Ruta estándar: `https://{dominio}/.well-known/agent-card.json`.
  - Adecuado para agentes públicos o de dominio controlado.
- **Curated Registries** (catálogos):
  - Registro centralizado de Agent Cards.
  - Búsqueda por habilidades, tags, proveedor, etc.
  - Útil en entornos empresariales o marketplaces.
- **Direct Configuration / Private Discovery**:
  - Configuración directa vía archivos, variables de entorno o APIs propietarias.
  - Adecuado para sistemas fuertemente acoplados o entornos de desarrollo.

Descubrimiento, empresa y ciclo de vida

Enterprise Features

Enterprise Features (1/2)

- **Transporte seguro (TLS)**:
  - Todo tráfico A2A en producción debe usar HTTPS.
  - Verificación de certificados para evitar ataques MITM.
- **Autenticación**:
  - Basada en mecanismos web estándar (OAuth2, OpenID Connect).
  - Declarada en el Agent Card (`security`, `schemes`).
  - Las credenciales viajan en cabeceras HTTP (p. ej. `Authorization`).
- **Sin identidad en el payload**:
  - Mensajes JSON-RPC no incluyen identidad de usuario/cliente.
  - La identidad se establece en la capa de transporte.

Enterprise Features (2/2)

- **Autorización**:
  - Lógica específica al agente y sus políticas.
  - Control granular por skill y por acción.
  - Principio de *least privilege*.
- **Privacidad y cumplimiento**:
  - Conciencia de sensibilidad de datos en Messages y Artifacts.
  - Cumplimiento de GDPR, CCPA, HIPAA según el dominio.
  - Minimización de datos y protección en tránsito y en reposo.
- **Trazabilidad y observabilidad**:
  - Integración con herramientas estándar de logging, tracing y métricas.
  - ContextId y TaskId facilitan el seguimiento extremo a extremo.

Descubrimiento, empresa y ciclo de vida

Life of a Task

Life of a Task: visión general

- Un mensaje inicial puede producir:
  - Una respuesta **Message** (stateless).
  - Una **Task** con ciclo de vida definido.
- Identificadores:
  - `contextId`: agrupa interacciones relacionadas.
  - `taskId`: identifica una unidad de trabajo.
- Estados de la tarea:
  - `working`, `input-required`, `auth-required`.
  - Terminales: `completed`, `canceled`, `rejected`, `failed`.

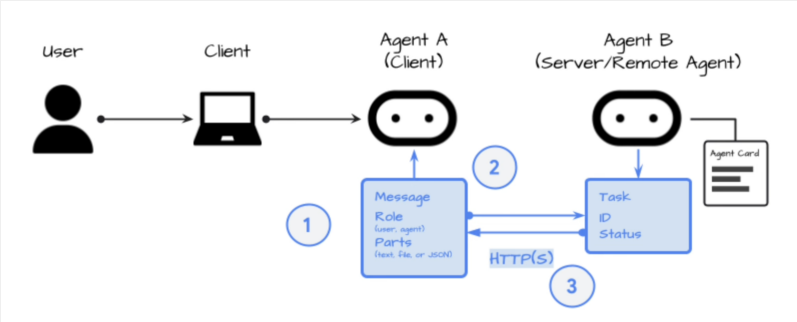


Figura: Esquema del ciclo de vida de una Task en A2A.

Life of a Task: contexto, refinamientos e inmutabilidad

- **contextId**:
  - Agrupa múltiples tareas y mensajes relacionados.
  - Permite sesiones multi-turno y colaboración entre agentes.
- **Refinamiento de tareas**:
  - El cliente puede iniciar nuevas tareas usando el mismo `contextId`.
  - Referencias a tareas previas vía `referenceTaskIds`.
- **Inmutabilidad de tareas**:
  - Una vez que una Task es `completed`/`canceled`/`failed`/`rejected`, no se reinicia.
  - Las interacciones posteriores crean nuevas Tasks vinculadas al mismo contexto.
  - Facilita trazabilidad, auditoría y mapeo limpio entrada–salida.

Extensiones y operaciones asíncronas

Extensiones y operaciones asíncronas

Extensiones

Motivación

El protocolo base es intencionalmente genérico. Las **extensiones** permiten añadir capacidades sin fragmentar el estándar.

- Tipos de extensiones:
  - **Data-only**: añaden datos estructurados (p. ej. cumplimiento GDPR).
  - **Profile**: imponen reglas adicionales sobre mensajes/estados.
  - **Method Extensions**: definen nuevos RPCs (p. ej. `tasks/search`).
  - **State Machine Extensions**: amplían la máquina de estados de Task.
- Declaración en el Agent Card:
  - Cada extensión se identifica por una `uri`.
  - Puede marcarse como `required` para el cliente.

Ejemplo de AgentExtension

Listing 3: Ejemplo de extensión declarada en Agent Card

```
"capabilities": {
  "streaming": true,
  "extensions": [
    {
      "uri": "https://example.com/ext/konami-code/v1",
      "description": "Provee 'cheat codes' para respuestas especiales",
      "required": false,
      "params": {
        "hints": ["arriba", "arriba", "abajo", "abajo"]
      }
    }
  ]
}
```

Extensiones y operaciones asíncronas

Streaming & Asynchronous Operations

Streaming & Asynchronous Operations (1/2)

- Diseñado explícitamente para **tareas de larga duración**:
  - Generación de documentos extensos.
  - Flujos con intervención humana.
  - Coordinación entre múltiples agentes.
- **Streaming con SSE**:
  - El servidor declara `capabilities.streaming: true`.
  - El cliente usa `message/stream` y recibe:
    - `Task` (estado actual del trabajo).
    - `TaskStatusUpdateEvent` (cambios de estado + mensajes intermedios).
    - `TaskArtifactUpdateEvent` (artefactos en chunks).
  - La conexión se cierra cuando `final: true`.

Streaming & Asynchronous Operations (2/2)

- **Resuscripción**:
  - Si el stream se corta, el cliente puede usar `tasks/resubscribe` para reconectarse a una tarea activa.
- **Push Notifications**:
  - El servidor declara `capabilities.pushNotifications: true`.
  - El cliente envía un `PushNotificationConfig` con:
    - `url` (webhook HTTPS).
    - `token` opcional para validación.
    - Esquemas de autenticación hacia el webhook.
  - Útil para clientes móviles, funciones serverless, o tareas de horas/días.
- **Seguridad**:
  - Validar identidad del servidor A2A al llamar al webhook.
  - Proteger el receptor de notificaciones contra abuso.

Referencias

A2A Protocol.

What is A2A?.
Disponible en: https://a2a-protocol.org/latest/topics/what-is-a2a/.

A2A Protocol.

Core Concepts and Components in A2A.
Disponible en: https://a2a-protocol.org/latest/topics/key-concepts/.

A2A Protocol.

A2A and MCP: Detailed Comparison.
Disponible en: https://a2a-protocol.org/latest/topics/a2a-and-mcp/.

A2A Protocol.

Agent Discovery in A2A.
Disponible en: https://a2a-protocol.org/latest/topics/agent-discovery/.

A2A Protocol.

Enterprise Implementation of A2A.
Disponible en: https://a2a-protocol.org/latest/topics/enterprise-ready/.

A2A Protocol.

Life of a Task.
Disponible en: https://a2a-protocol.org/latest/topics/life-of-a-task/.

A2A Protocol.

Extensions in A2A.
Disponible en: https://a2a-protocol.org/latest/topics/extensions/.