SagaChain A2A and MCP Positioning

David Beberman

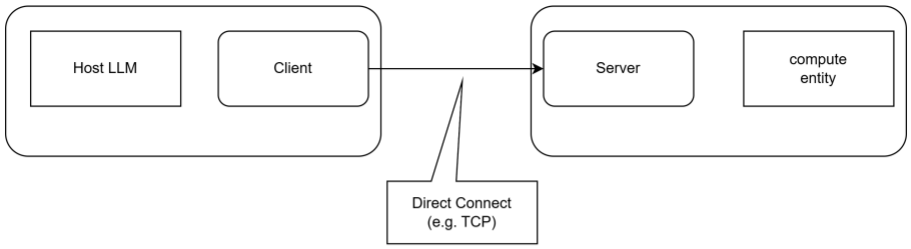
20260331

Abstract

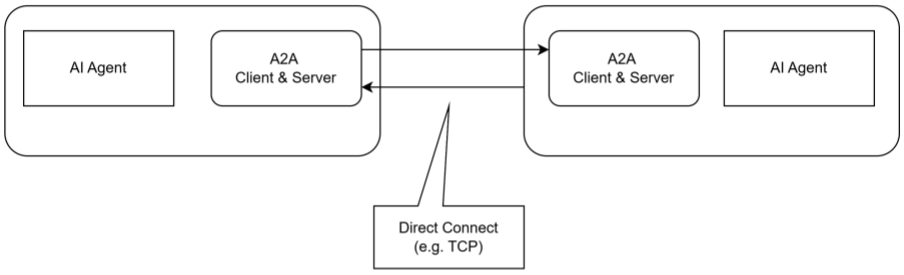
Google has published the Agent 2 Agent (A2A) protocol specification for AI Agents to communicate with each other. Anthropic has published the Model Context Protocol (MCP) protocol specification to communicate from a Host LLM to non-AI data sources and tools. Both protocols leverage existing https/json rpc style communications with one entity assuming the role of a client and the other assuming the role of a server. The specifications imply a direct network connection (local on a machine or across any network architecture) from the client role to the server role. The specifications do not preclude a bank of servers that use any form of load balancing either transparently or explicitly between the client and the server in a specific communications instance. The A2A protocol enables the client in one exchange to assume the server role in any other exchange. Both protocols provide an immediate request/response, streaming response and asynchronous response communication model.

Prasaga proposes two alternative models that provide equivalent capabilities of both A2A and MCP while augmenting both models with non-bypassable audit logging, account management and accountability, respectively. That is, Prasaga's proposed model addresses the "5 A's": Authorization, Authentication, Account Management, Audit Logging, and Accountability. Each proposed new model relies on the general purpose mechanisms of SagaChain's immutable decentralized object-oriented object state database, and related infrastructure. In comparison, as will be discussed, neither A2A nor MCP support non-bypassability. This limitation prevents realization of non-disputable Account Management, Audit Logging and Accountability for both protocols.

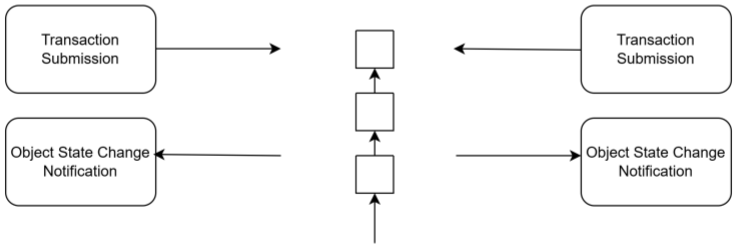
MCP Protocol Architecture

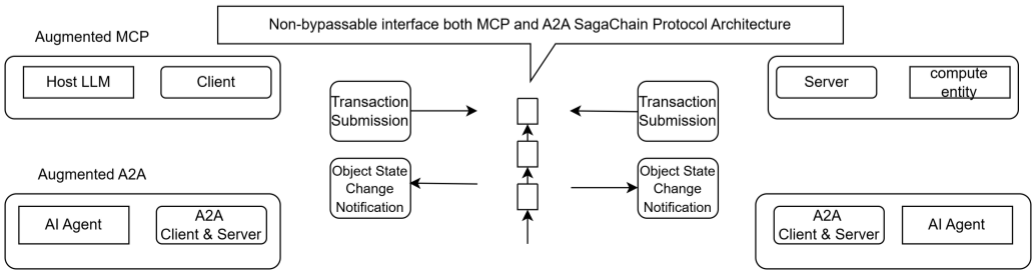


A2A Protocol Architecture



SagaChain Transaction/Object Notification Protocol Architecture





Google A2A and Anthropic MCP Brief Overview

AI Agent to AI Agent protocol specification (A2A) by Google makes use of https and JSON RPC for connecting from an AI Agent acting as a client (or any other client following the protocol) and an AI Agent acting as a server. The protocol includes a discovery capability similar to OpenAPI discovery called "Agent Card". Additionally, the protocol includes a persistent session capability for protocol exchanges that can be streaming response or asynchronous response (via webhook) to requests.

MCP protocol specification consists of an AI Agent, chatbot, or similar acting in a client role, and an MCP process in a server role. The MCP provides access to computing entities that normally are non-AI implementations (e.g. business applications, physical devices, etc.). The MCP protocol includes a discovery mechanism that provides the AI LLM with the available prompts for the server at a URL resources/list, similar in concept to the Agent Card for the AI Agent.

Note that MCP is for Host LLMs to external tools, devices, etc. While A2A is targeting AI Agent to AI Agent (resulting in LLM to LLM engine) communications.

A2A and MCP Client Server Architecture

The A2A can be viewed as a peer-to-peer protocol as the AI Agents are logically peers of each other, while for each protocol exchange, there is a client and a server. The client sends a request and the server responds either immediately, streaming, or asynchronously. For any given exchange an AI Agent that was previously a client may act in the server role in a following or concurrent exchange. This assumes the AI Agent implements the server role as well as the client role, which is the usual case.

The MCP protocol is more clearly a fixed client – server protocol, where the client, identified as the "host" instantiates a client entity that connects to the MCP server.

In both cases the protocol exchange is architecturally a direct connection from the entity in the client role to the entity in the server role.[1][2][3]

The 5 A's, A2A and MCP Failure

Authorization, Authentication, Account Management, Audit Logging and Accountability are the "5 A's" commonly described for "cloud security" and more generally applicable for various distributed use-cases. [4] Of these, authorization and authentication are fully covered by the client-server protocols of A2A and MCP. Both protocols take full advantage of existing security models for web-based protocols. Prasaga's contention is that both protocols fall short for the remaining 3 A's due to the client-server architecture nature of the protocols, not due to any specific realization in an implementation of the protocols. The shortcoming is with respect to non-disputability.

For any given A2A or MCP implementation, the entity in the client role and the entity in the server role can perform all of the 5'As. This includes the remaining 3 A's that are in contention. The disputability problem arises when there is a conflict between the data that either the client or the server party possesses, either during an exchange, during any sort of

---

[1] https://a2a-protocol.org/latest/specification/

[2] https://a2a-protocol.org/latest/topics/a2a-and-mcp/

[3] https://modelcontextprotocol.io/docs/getting-started/intro

[4] https://securityboulevard.com/2023/08/a-comprehensive-guide-to-the-five-as-of-cloud-identity-management/

finalization and settlement, during any auditing event, or any other type or reconciliation event. Each party contains their own data related to the protocol exchange without any means to resolve a disagreement. By the architecture of the protocols there is no "single source of the truth", no computational way to resolve a dispute.

Non-bypassability Mandatory [5]

The mandatory requirement to create a non-disputable single source of the truth is the inclusion of an entity that both the client and server must involve in the protocol exchanges and must be inherently independently verifiably trusted by both parties for each protocol exchange. The minimal functionality to meet this requirement is non-bypassability. A given protocol exchange between a client and a server shall not be able to bypass the entity providing the single source of the truth for all data associated with the 5 A's. As described in the above section, all client-server protocol models are unable to provide non-bypassability and as a direct result are unable to provide a non-disputable single source of the truth. By extension neither the A2A nor the MCP protocols are suitable for applications that require all 5 A's.

SagaChain Protocol Concept and A2A and MCP Non-bypassable

The SagaChain non-bypassability protocol concept consists of all interactions between any entity acting in the client role and any entity acting in a server role to be replaced by two similar entities in a transaction submission and "observer notification" role with the SagaChain blockchain. No entities directly connect with each other and do not directly exchange any data with each other with respect to the 5 A's. The two features, transaction submission and observer notification are used for all protocol exchanges and enforce non-bypassability.

Observer Notification

SagaChain blockchain nodes provide a unique capability termed observer notification for any object in the SagaChain object state database. Any entity such as an A2A or MCP

client or server registers with one or more SagaChain blockchain nodes to receive notifications if an identified object changes state. All SagaChain objects are identified by globally unique ledger object identifiers (LOID). An entity may register to receive notifications for one or more objects subject to the specific protocol design and implementation.

Transaction Object State Update

A transaction script submitted to SagaChain may update the state of any number of objects in the object state database. (A specific class, SPClassTouch is provided to inherit from, to enable any object to have a state change for notification purposes). Once a transaction has been submitted to SagaChain, has been included in a block, reached consensus, and gossip broadcast to all nodes, any node that has a registered observer notification for relevant objects will deliver a notification to the registering entity.

Client-Server Protocol Replacement Concept

The A2A and MCP client to server direct connect protocols are replaced by a transaction submission and object notification model. The protocol abstract description is as follows:

- The entity acting in the client role submits a transaction to SagaChain blockchain that updates the state of an object that an entity acting in the server role has registered to receive notifications on state changes.
- The transaction is included in a block, reaches consensus, and a node with the registered server entity fires a notification to the server entity.
- The entity acting in the server role reads the object state from the object state database and performs the requested activity.
- The entity acting in the server role submits a transaction to the SagaChain blockchain with any results of the activity, updating the state of an object that the entity acting in the client role has registered to receive notifications on.
- The transaction is included in a block, reaches consensus and a node the registered client entity first a notification to the client entity.

The protocol abstract description can be directly extended to support synchronous request/response, streaming responses and asynchronous responses, maintaining non-bypassability for all protocol interaction models.

SagaChain Immutability and Single Source of the Truth

The consensus nature of the blockchain concept encompasses immutability by design. SagaChain uses a combination of mathematically proven consensus protocols based on Byzantine Fault Tolerance (BFT), Proof-of-work (POW) and Prasaga's patented Distributed Proof-of-work (DPOW) to enforce immutability of the consensus blocks of the blockchain. The object state database is updated solely by execution of transactions in the blocks, by the blockchain nodes acting as validators for each block. The immutability of the blockchain and the object state database provide a global single source of the truth.

Non-bypassable accounting and auditing are fully satisfied by the append-only nature of the blockchain combined with ability to retrieve the history of all submitted transactions that were included in consensus blocks and all changes to the object state database.

SagaChain Non-bypassability, Single Source of the Truth and AI Governance

An increasing concern related to the use of LLM-powered AI is governance of the AI implementation's actions. To address this concern, standards and regulations are developing requirements for full audit trails, pre-commit risk assessment and human-in-the-loop enforcement. The SagaChain transaction script and message passing object execution model, coupled with the non-bypassability protocol architecture enables implementation of rules, boundaries, guardrails, etc. to enforce governance policies defined by stakeholder parties.

SagaChain's non-bypassable protocol architecture serves as an enabling technology mechanism and does not limit governance policies that may be implemented using the transaction execution/object notification model, the immutable blockchain and decentralized object state database.

Conclusion

SagaChain provides a unique viable solution for the non-bypassability problem for A2A and MCP AI protocol exchanges, resulting in satisfying the 5 A's: Authentication, Authorization, Account Management, Audit Logging and Accountability. The proposed protocol concept enables non-disputability eliminating the need for reconciliation between all parties involved in any series of protocol and data exchanges.

Addendum: SagaChain Architecture Supporting AI Server Redundancy & Load Balancing

Fault Tolerance Redundancy

A protocol implementation utilizing the SagaChain transaction execution/observer notification model creates a loosely coupled relationship between entities acting in a client role and entities acting in a server role. Computing elements by their nature are prone to failure events, crashes, restarts, etc. Server systems that require some level of fault tolerance usually provide a mix of active-active, active- hot standby and standby backup servers to take the place of a failed server. A protocol may be designed implementing the SagaChain model that creates opacity of the specific server entity that serves and responds to a client entity. The protocol may be designed such that the client entity has no explicit knowledge of the address of the server entity for any given protocol exchange. This loose coupling enables any mix of fault tolerant server redundancy needed to meet availability requirements for the server entity.

It should also be noted that SagaChain inherently provides fault tolerant redundancy by the availability of multiple nodes (intended to be in the thousands) supporting the blockchain operation. A well designed fault tolerant AI Agent entity server should connect to multiple nodes to provide connectivity redundancy to SagaChain.

Load Balancing

As a direct consequence of the loosely coupled nature of using the SagaChain transaction execution/observer notification protocol concept, multiple server entities can cooperate to respond to object notifications using a load balancing algorithm among the server entities. The load balancing algorithm may make use of SagaChain objects in the object state database. As a simple example, a load balancing object may contain a simple incrementing index modulus the number of entity servers available. An entity client submits a transaction that includes incrementing the load balancing object's index. A server entity matching the current index responds to the object notification enabling an opaque round-robin load balancing protocol.