Network Working Group
Internet-Draft
Intended status: Informational
Expires: 22 April 2026

J. Rosenberg
Five9
C. Jennings
Cisco

19 October 2025

Framework, Use Cases and Requirements for AI Agent Protocols
draft-rosenberg-aiproto-framework-00

Abstract

AI Agents are software applications that utilize Large Language Models (LLM)s to interact with humans (or other AI Agents) for purposes of performing tasks. AI Agents can make use of resources - including APIs and documents - to perform those tasks, and are capable of reasoning about which resources to use. To facilitate AI agent operation, AI agents need to communicate with users, and then interact with other resources over the Internet, including APIs and other AI agents. This document describes a framework for AI Agent communications on the Internet, identifying the various protocols that come into play. It introduces use cases that motivate features and functions that need to be present in those protocols. It also provides a brief survey of existing work in standardizing AI agent protocols, including the Model Context Protocol (MCP), the Agent to Agent Protocol (A2A) and the Agntcy Framework, and describes how those works fit into this framework. The primary objective of this document is to set the stage for possible standards activity at the IETF in this space.

Table of Contents

1. Introduction

AI Agents have recently generated a significant amount of industry interest. They are software applications that utilize Large Language Models (LLM)s to interact with humans (or other AI Agents) for purposes of performing tasks. Just a few examples of AI Agents are:

* A travel AI Agent which can help users search for travel destinations based on preferences, compare flight and hotel costs, make bookings, and adjust plans

* A loan handling agent that can help users take out a loan. The AI Agent can access a users salary information, credit history and then interact with the user to identify the right loan for the target use case the customer has in mind

* A shopping agent for clothing, that can listen to user preferences and interests, look at prior purchases, and show users different options, ultimately helping a user find the right sports coat for an event

Technically, AI agents are built using Large Language Models (LLMs) such as GPT4o-mini, Gemini, Anthropic, and so on. These models are given prompts, to which they can reply with completions. To creates an AI agent, complex prompts are constructed that contain relevant documentation, descriptions of tools they can use (such as APIs to read or write information), grounding rules and guidelines, along with input from the user. The AI Agent then makes decisions about whether to interact further with the user for more information, whether to invoke an API to retrieve information, or to perform an action that might require human confirmation (such as booking a flight).

AI Agents are often built to be task specific. In the examples above, each AI agent was uniquely tuned (primarily through its prompt structure, though perhaps also with fine-tuned models) to the specific task at hand. This introduces a problem. A user interacting with one agent might require the agent to do something which it is not tuned to do, requiring the agent to interact with a different agent that has the appropriate skills. This is called agent federation or Agent-to-Agent communication.

The result of this is a system which has many different software elements communicating over the Internet. This introduces the potential need for additional areas of protocol standardization to facilitate deployment of these AI Agent systems. This need has been broadly recognized already in the industry, resulting in a rapid stream of open source and protocol efforts. Examples include the

Model Context Protocol (MCP) (https://modelcontextprotocol.io/introduction (https://modelcontextprotocol.io/introduction)) which is being driven by Anthropic, the Agent2Agent (A2A) Protocol, driven by Google (https://google.github.io/A2A/#/documentation?id=agent2agent-protocol-a2a (https://google.github.io/A2A/#/documentation?id=agent2agent-protocol-a2a)) and the Agntcy framework, driven by Cisco (https://agntcy.org/ (https://agntcy.org/)). There are certainly others. These efforts cover parts of the standardization spectrum, partially overlapping in scope in some cases.

With the potential for significant adoption of AI Agents across the Internet, these protocols may become the foundation for the next wave of Internet communication technologies, especially between domains. Indeed, we can think of inter-domain protocol stack as being composed of IP [RFC791], UDP [RFC768], TCP [RFC9293], and QUIC [RFC9000] as the foundation for communicating between hosts. The next layer - HTTP [RFC9114], SIP [RFC3261] and RTP [RFC3550] are the main inter-domain protocols at the application layer. If AI Agents are the new way in which users interact with applications, then the protocols between them could represent the next layer of this stack.

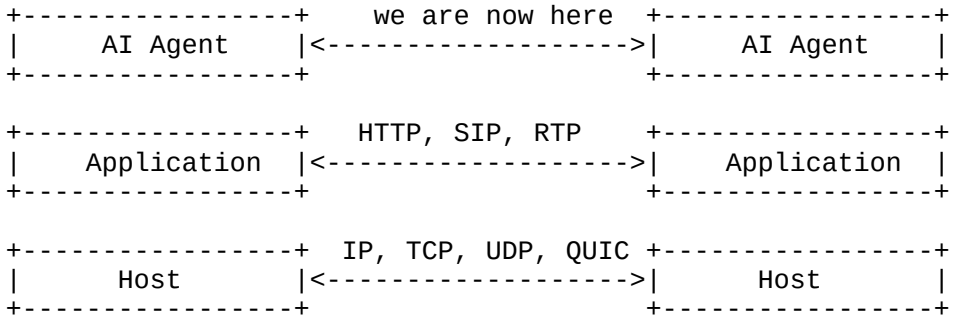

Figure 1: Are AI Protocols the next layer of the modern IP Protocol Stack?

This means they must be designed with care, and deeply consider the necessary security, privacy, and extensibility requirements needed for broad adoption. Most importantly, they must consider the needs of the humans who ultimately use these systems, and be designed to manage the harms that may be introduced through inaccuracy and hallucination that are an unavoidable consequence of using LLMs.

This document provides a framework to aid in standardization efforts in this area. It identifies the different protocols which come into play, and considers different variations in their use cases. It discusses requirements for these protocols, covering functionality

along with considerations for privacy, security and hallucination management. The document also includes a brief overview of some of the existing protocols and open source efforts, and maps them into the framework provided here.

2. The Framework

The overall architecture is shown in Figure 2.

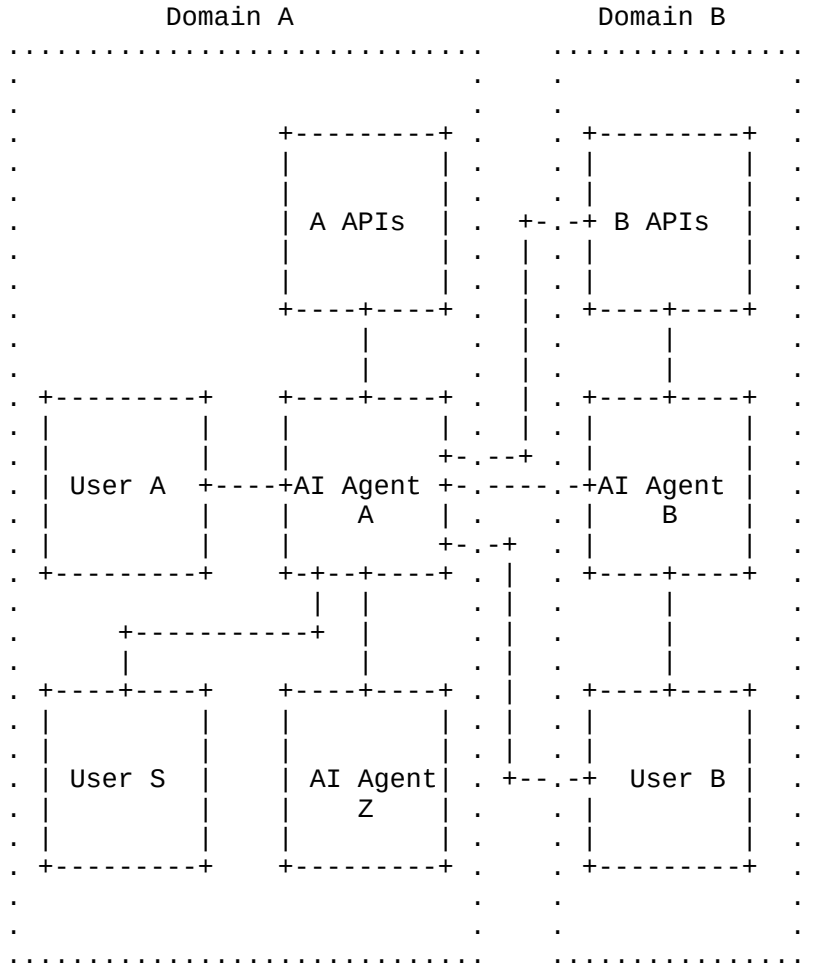

Figure 2: AI Agent Protocol Framework

The framework focuses on the AI agent in domain A - "AI Agent A". This AI agent is providing services to a user, User A. To do that, User A communicates with AI agent A over voice, chat or other

channels. AI Agent A supports this user by invoking APIs - either in its own domain, or in another. AI Agent A might also communicate with other users - ones in domain A, or outside of it. Consequently, there are broadly speaking three classes of protocols in play: (1) Protocols for human to AI Agent, (2) Protocols for AI agents to invoke APIs, and (3) Protocols for AI agents to talk to each other.

2.1. AI Agent to User

We refer to this protocol as the "User to Agent Communications Protocol". These communications can be initiated by the user, or initiated by the agent.

2.1.1. User Initiated

The process generally starts when the user initiates communications with its AI Agent.

User A communicates to AI agent 1 over some kind of communications technology. It could be a via a phone call, or via voice chat via a widget on a web page or inside a mobile or desktop app. It could be via chat, using a chat widget in a webpage, or via SMS, RCS or applications like WhatsApp or Facebook Messenger. These chat apps provide a variety of content types, including text, images, carousels, links, videos, and interactive html cards. The interaction could be via video as well. Often video communications are one way, from the user to the AI agent, to facilitate "see what I see" interactions. Or, it can be two way, in which case the user might see an avatar of the AI agent.

The communications could also be multi-modal. User A1 could be talking to AI Agent 1 using voice, while at the same time, receiving texts from the AI Agent with information or links. User A1 could be using voice and video concurrently. User A could be using a web widget for chat and multimedia content, while using a phone call to communicate with the same agent.

In all of these cases, User A has a relationship with AI Agent 1 (typically by being a user of the domain or site in which it resides). This is why the user is shown within the domain boundary in the picture.

The protocols and technologies used for this communication are well established today, and it is not apparent that there is any additional standards work required to support AI agents. Indeed, users are already able to communicate with AI agents over these channels today.

2.1.2. AI Agent Initiated

In some cases, AI agent 1 may find that - in order to do its job - it needs to communicate with another human user.

As one example, User A may be interacting with an AI agent providing loans. The user might have a request for a lowered loan rate based on their holdings with the bank. The AI agent could be programmed to always reach out to a human in order to approve reduced load rates. When the conversation between User A and the loan AI agent reaches this point, the AI agent needs to communicate with the human supervisor to gain this approval. This is User S in the diagram above.

In the most common use cases, the human user would be in the same domain as the AI agent. This is the case for our loan example above. In these cases, the AI agent could use email, voice, chat or other communications techniques which are known to be used by User B. Often, User B would be a contact center agent connected to the same platform facilitating operation of the agent. In other cases, they are on separate communications systems. The administrators of domain A would be required to configure the systems to communicate with each other using standard protocols ala SIP or SMTP. It may also be the case that the communications system user by User S exposes APIs that enable sending of messages, in which case the interface actually looks like one between an AI agent and an API.

It could be the case that the human user is in another administrative domain entirely. In this case, AI agent may only have an email address or phone number for the user to contact, and not have further information about their capabilities. This is the case for AI Agent A communicating with User B.

2.2. AI Agent to API

For AI Agent A to do its job, it will require access to APIs. Those APIs can provide read-only information - for example a product catalog or list of hotel properties. The API could take actions, for example an API to book a flight, create a case, or start a product return. These APIs could be intra-domain or inter-domain.

In the intra-domain case, the APIs are administered by the same organizational entity that is providing the agent. For example, a travel booking site might run an AI agent for such bookings. The site has microservices which have APIs for retrieving travel destinations and airports, booking flights, or creating trip records. In these cases, the APIs are intradomain.

Alternatively, the APIs could reside outside of domain A. Once again, using the travel booking example, the travel booking site might invoke APIs offered by partner airlines to book the flights. In the case of inter-domain APIs, there are two sub-cases. In one case, it is domain A which has the relationship with domain B, and the APIs are invoked using service accounts that are not specific to User A. That said, they could be operating on behalf of user A and thus be limited or constrained based on User A's data and permissions. In the second sub-case, the AI Agent is operating directly on behalf of User A, using a token granted to the AI agent by an OAuth flow performed by user A. This could happen in cases where there is not necessarily a business relationship between domain A and domain B, but instead a relationship between user A and domain B. Again using the travel booking agent as an example, User A might have accounts with United Airlines and Delta Airlines, both of which expose APIs for booking flights, obtaining flight information, and accessing frequent flier information. User A could authorize AI Agent A to access both United and Delta on his behalf. The AI Agent could then retrieve information from both, compare flight options and costs, and then execute flight bookings. In this case, the AI agent has no advanced knowledge of the APIs offered by United or Delta, and needs to learn what they are and have enough information to know how to use them appropriately.

2.3. AI Agent to AI Agent

The final piece of the framework is AI agent to AI Agent. This is perhaps the most interesting of these communications protocols. As with most of the other protocols, they can be intra-domain or inter-domain.

For an intra-domain example, consider a bank with several lines of business. They have a loan department, a wealth management department, a private equity investment department, and a personal banking department. A particular user - perhaps a customer of the personal banking department - has an account with the bank. While chatting with the personal banking agent, the user inquires about loans. In this case, the banking agent recognizes that it does not have the required expertise, and connects the user with the loan AI agent (agent Z in the diagram above).

In a similar use case, User A is communicating with the personal banking agent, and requests to close out all of their accounts and have a check issued to them. The bank has employed a different AI agent - the "closure agent" which runs through a complex workflow required to close down a customer account. This agent might invoke several APIs to close accounts, update databases, and send out emails and texts to the user. In this case, User A doesn't actually chat

with the closure agent. The closure agent instead executes a complex task which involves reasoning, API invocation and possible communications with humans.

Both of these are intra-domain cases. Inter-domain cases are similar. If we once again consider the travel AI Agent A, that AI agent might assist the user in selecting travel destinations and pick dates. To actually book flights, the travel AI Agent A might connect the user to the flight AI Agent B within the airline. Like the API case, there are two distinct relationships that are possible. Domain A might have the relationship with domain B, in which case the authorization tokens used between them are service accounts and not specific to User A. Alternatively, it is user A which might have the relationship with domain B and its AI Agent. In that case, User A would need to grant domain A access to the AI Agent in domain B, giving it permissions to communicate and take certain actions.

So far, we've been vague about the notion of how user A - having started is communications with AI Agent A - is connected to AI Agents Z or B. We can envision a finite set of techniques, which mirror how this would work when transferred a user between human agents in a contact center:

1. Blind Transfer: In this case, User A is literally redirected from AI Agent A to the receiving agent, AI Agent Z. In a blind transfer, AI Agent A is not a participant in, and has no access to, the communications with AI Agent Z after the transfer. However, there could be contextual information transferred from AI Agent A to AI Agent Z. This could be transcript history, API requests and resulting data, and any other context which had previously been known to AI Agent A.

2. Supervised Transfer: In this case, User A is told by AI Agent A that they will be connected to AI Agent Z. TO do that, AI agent A "conferences" in AI Agent Z, so that now all three are together in a group conversation. AI Agent A would communicate with agent Z - in a format comprehensible to user A - that a transfer is taking place. This aspect of the supervised transfer is for the benefit of the end user, giving them comfort that a transfer is happening and making sure they know that the receiving AI Agent has context. Of course, there may additionally be transfer of programmatic information in the same way described for the blind transfer case. Once the introduction has completed, AI Agent A drops and now the user is talking solely to agent Z.

3. Sidebar: In this case, AI agent A "talks" to AI Agent Z, but none of that communications is shown to User A. Instead, it is purely to provide AI agent A information it requires in order to

continue handling user A. As an example, we return to our travel booking AI agent. User A asks it, "What are the most popular travel destinations for a spring beach vacation?". AI Agent may not have the context to answer this question. So, it instead connects to a travel blog AI Agent, and - in this case - makes the same inquiry to it, adding however additional context about User A. It asks the travel blog agent, "For people living in New York, what are the most popular travel destinations for a spring beach vacation? Please format your response in JSON." When the results come back - here in structured data format - AI Agent A (the travel booking agent) might lookup each city in a database to retrieve a photo, and then render a photo carousel to the user with the three top cities. In this example, the sidebar is a single back and forth. There can be more complex cases where multiple back and forths are required to provide AI Agent A the information it needs.

4. Conference: In this case, AI agent A has AI agent Z join a group conversation, which then involves User A, AI Agent A and AI Agent Z. The three of them converse together. Both AI agent A and AI Agent Z are able to see all messages. The agents would need to be programmed (via their prompting) to try and determine whether each user utterance is directed at them, or the other AI agent. Similarly, it would need to know whether it is required to respond to, or just add to its context, utterances sent by the other AI Agent.

5. Passthrough: In this case, User A connects to AI Agent A, which then realizes it wants to connect the user to AI Agent Z. To do that, it acts as a proxy, relaying content from AI Agent Z to User A, and relaying content from User A to AI Agent Z. The AI Agent doesn’t take action on the content, but it might store it for contextual history. In SIP terminology, this can be considered a "B2BUA".

It is worth noting that these variations are all also possible in the case where it is User S is being added to the conversation, and not AI Agent Z. This has implications on the AI agent to Ai agent protocols, as noted in the discussions below.

We refer to the AI agent which makes the decision to involve a second AI Agent as the invoking AI agent. The AI agent which is then brought into the conversation using one of the topologies above, is called the invoked AI Agent.

3. Protocol Requirements and Considerations

Each of these three areas of protocol - user to AI agent, AI Agent to API and AI Agent to AI Agent, have considerations and requirements that are important for standardization.

3.1. User to AI Agent

As noted above, for the most part, this aspect of communication is well covered by industry standards, and is often done via proprietary means when the user and AI agent are within the same administrative domain.

3.1.1. Authentication

One area of particular interest is user authentication. For many use cases, the AI Agent will need to know who the user is. For a web chat bot on a web page, the identity is easily known if the user has already logged into the web page. However, there are many other communications channels where the user identity is not so easily known. For example, if the user initiates communications over SMS, the AI agent has only the phone number of the originator. Similarly, if it is via a voice call on the PSTN, there is nothing but the caller ID. Even for web chat, the hosting web site might not support authentication. A great example of this are websites for services like plumbing or home repair. Users might be customers of these providers, but the website doesn’t have traditional authentication techniques.

In all of these cases, authentication needs to take place through the communications channel, often with proof of knowledge demonstrations by the end user. This is how it is done when the agent is an actual human and not an AI. When a user calls their plumber, the user provides their account number, or even just their name and address. For other services, such as a bank or healthcare provider, more complex authentication is required. Usually, multiple pieces of personally identifying information are required. When calling a doctor, the patient’s zip code, name, and date of birth may all be required.

These forms of authentication introduce additional complexity for the AI agent to API and AI agent to AI Agent communications. This authenticated identity needs to be communicated. This is noted in sections below.

3.2. AI Agent to API

The first question everyone asks when considering this area is - do we need new standards at all?

After all, there are already standards for APIs, primarily using REST. Standards exist for describing REST APIs, and indeed they contain descriptive information which can be consumed by an LLM to determine how to invoke the API. Indeed, early versions of AI agents have been built by just pointing them at (or uploading) OpenAPI yaml specifications, resulting in functional AI Agents. However, when the fuller scope of use cases are considered, several additional considerations come to light.

3.2.1. Discovery

Initial versions of AI agents required manual upload of OpenAPI specifications. This can be improved through discovery techniques which allow AI agents to quickly find and utilize such specifications.

Discovery can happen for intra-domain cases - where it is simpler - or for inter-domain cases. For inter-domain cases, a primary consideration is one of authentication and access. An AI agent won’t in general just be allowed to invoke any public API it discovers on the Internet. This might be the case for general public read-only services, such as weather or maps. However, in general, APIs will require authentication and authorization, and thus require some pre-established relationship. This means discovery for most APIs is more of an issue of well-known locations for openAPI specifications.

3.2.2. AuthN and AuthZ on APIs

A significant security consideration is around user authentication and authorization. There are many use cases.

3.2.2.1. Intra-Domain Service to Service

In this use case, AI Agent A performs the authentication of the user somehow. As noted above, this may even involve exchange of proof-of-knowledge information over a communications channel, in which case there is no traditional access or bearer token generated for the end user.

AI Agent A, once it has determined the identity of user A, invokes APIs within its own domain. It invokes these APIs using a service account, representing itself as the AI Agent. The APIs allow the AI agent to take action on behalf of any end user within the domain. The APIs themselves would need to just contain user identifiers embedded with the API operations.

This use case - which is largely how AI agents work today - requires no specific protocol standardization beyond what is possible today.

3.2.2.2. Intra-Domain OBO

A significant drawback of the technique in the previous section is that it requires the AI agent to have very high privilege access to APIs within the domain. This is perhaps not a major concern for prior generates of AI agents whose behaviours were programmed, and therefore the risk of malicious users or inaccurate AI was much less.

But, with AI agents, there are real risks that users could maliciously (through prompt injection attacks for example) direct an AI agent to invoke APIs affecting other users. Or, it could manipulate the APIs being invoked to do something different than what is expected. Even without malicious intent, an AI agent might make a mistake and invoke the wrong API, causing damage to the enterprise.

A way to address these is to have the AI Agent operate using a lower privilege access token - one specifically scoped to the user which has been authenticated. This would prevent against many hallucination or prompt injection cases. In essence, we want to have the AI Agent obtain a bearer token which lets them operate on-behalf-of (OBO) the end user it is serving.

This is relatively straightforward when the user has performed normal authentication on a web chat bot, and is anyway passing a user token to the AI Agent. The AI Agent can just use that same token for invoking downstream APIs, or else use that token to obtain other, scope limited tokens from a central authentication service. However, things are more complicated when the user was authenticated using proof-of-knowledge via communication channels. In this case, there was never a user token to begin with!

To support such cases, it is necessary for the AI Agent to obtain a token for the user. To do so, it would ideally communicate with some kind of central identity service, providing the information offered by the end user (the date of birth, address, and name in the example above), and then obtain a token representing that user. The resulting tokens would ideally contain programmatic representations of the techniques by which the user was authenticated. This would allow APIs which accept these tokens to perhaps decide that additional levels of authentication are required.

This is common today when dealing with human agents. A user calling their bank to check on open hours may not require any authentication at all. Once the user asks for an account balance, the human agent will perform an authentication step by requesting identification information from the user. Account status can then be provided. Finally, if the user then asks for a wire transfer, the human agent may ask for further authentication, including perhaps having the end user receive a text at their mobile number and real a one-time code.

To support equivalent security, this needs to manifest within protocols. If an AI Agent attempts to invoke an API for which the token has insufficient strength of authentication, the API request needs to be rejected, and then the AI agent needs to communicate with the user to obtain further identification information, and obtain a new token with higher privilege.

Additionally, the meta-data describing the APIs would specify the required level of identity verification required to invoke them, so that the AI agent can request the information from the user and obtain a new token, rather than trying and failing on the prior, lower privilege token.

Could these tokens also be used for inter-domain use cases? Perhaps.

3.2.2.3. Inter-Domain OAuth

Consider User A talking to AI Agent A which is accessing APIs in domain B. In the case where User A has the relationship with domain B, the user will grant permissions to AI Agent A with a given scope. That scope will restrict the set of APIs which the AI agent is allowed to use. For the AI Agent to operate effectively, it will need to know which APIs are accessible for a given scope. This needs to be via both natural language, and likely programmatic descriptions. This would allow the AI Agent behaviour to adjust based on what access the user has.

To give a concrete example, consider again the travel booking AI agent. User A has executed an OAuth grant flow, and granted the AI Agent access to look up trips and frequent flier status, but not to book trips or modify reservations. The user might ask the AI Agent, "can I change my reservation?" and we want the AI Agent A to be able to respond, "I'm sorry, I dont have sufficient permission to perform that operation. I can initiate a flow by which you'll log into the airline and grant me that permission. Would you like me to do that?" And then the user response, "what exactly would granting this permission do?" and we want the AI to be able to respond along the lines of, "The permission I will request - change reservations - grants me the ability to modify your reservations on your behalf." The user might ask, "are you also allowed to delete them?" and it would respond, "Yes, the scope I would request grants me permission to delete your reservations.". The user could even respond, "I don't want that. I want to make sure you cannot delete my reservations". If the airline offered a scope which excluded deletions, it could respond with, "OK, I'll request permissions only to modify and not delete. Are you ready to grant me those permissions?". At that point, the user would get an OAuth popup asking for modification permissions.

Once those permissions are granted, the AI Agent would need to then know the specific APIs available to it at the granted scope. Indeed, as a matter of privacy, domain B might not want AI Agent A to know ALL of the APIs it has, it may want to restrict it to just those associated with the scopes that its users have granted. This reduction in the amount of detail offered to the AI Agent can also help improve AI accuracy, by reducing the amount of extraneous information it has access to.

For this to work as described, we need additional standardization allowing natural language descriptions of API scopes to be produced by one domain, read by the AI agent in another, and then allow API specifications to be conveyed based on granted scopes.

3.2.3. User Confirmation

When APIs are for read only, there is low risk of the AI doing the wrong thing. If a user requests information about their flight to New York and the AI instead returns the flight to York, this is frustrating for the user but not dangerous. This is not the case for APIs that have side effects, including creation, update and delete operations on RESTful objects. For these operations, there is a real risk that the AI Agent invokes the wrong API, or invokes it with the wrong data. To reduce this risk, it is important that the user provides confirmation, and that this confirmation happen programatically, outside of the LLM prompt inference.

To do that, it will be necessary for the AI Agent system to know which APIs require user confirmation, and which do not. This is a decision which can only be made by the API owner. Consequently, there is a need to annotate the APIs with ones that require confirmation. But, it goes beyond this. Users cannot be expected to view a JSON object and confirm its accuracy. Consequently, for each operation that has side effects, the API needs to be adorned with user visible names and descriptions of attributes which require confirmation, and then a mapping of those into the JSON payloads. This would allow the AI model to know that it needs to collect a set of data, and then once collected, the AI agent - not the LLM, but something programmatic - can render this information to the user, get their confirmation, map it into the API, invoke the API and then feed the results into the LLM. This provides 100% guarantee that the system is doing what the user wants, and not something else. All of this requires additional protocol machinery to convey this information and mappings, and is not covered today by typical API specifications.

3.2.4. API Customization for AI Agents

REST APIs today are crafted by humans, meant to be understood by humans and then implemented by humans into code that exercises them. Consequently, they are often highly complex, with many parameters and variations. Using those APIs - and their corresponding specifications - introduces a problem. While AIs can understand them, they may contain more information than is strictly needed, increasing the risk of inaccuracy or hallucination. An alternative model is to have simplified APIs - ones better aligned to the information that the AI Agent needs to collect from the humans in the conversation.

These specifications will also require descriptions that are consumed by the LLM to understand how to use them. Typically, content and prompts are ideally tuned to a specific model for optimal outcomes. One could imagine that API provider B might have several variations of the API specifications and descriptions optimized for different models. Consequently, the protocol for retrieving them would need to communicate the LLM models used by domain A to retrieve the optimal descriptions.

It is not entirely clear this level of customization is needed, but it is worth calling out as something unique for AI Agents.

3.3. AI Agent to AI Agent

There are many considerations for standardization in this area.

3.3.1. Discovery

One possible area for standardization is discovery - how do agents discover the availability of other agents. Discovery is often not needed however. For intra-domain cases, most of the agents are well known to administrators, and they can be configured into the system. For inter-domain cases, discovery is more interesting. However, as with AI Agent to API discovery, it is usually constrained by the requirements for authentication and access control. AI Agent A cannot communicate with other agents without a prior established security relationship, in which case discovery is less of a consideration.

Indeed, a high level of trust is required for one AI agent to talk to another one. A malicious AI agent that is discovered and used, could perform prompt injection attacks, possibly granting it access to the APIs that the other agent has access to. It could inject malicious or unwanted content that is passed on to the end user.

3.3.2. Routing and Connection

For one AI Agent to communicate with another one, it needs to know when to route communications to that agent. This requires information to be communicated from the agent being invoked, to the invoking agent. This information needs to be sufficient for ingestion by a LLM to make a routing decision, and likely requires text in natural language format to be communicated. This also raises considerations around languages. One can imagine that AI Agent A operates in Spanish, and would prefer to receive information to facilitate routing in Spanish as well.

Capabilities are also relevant for routing. One important area of capability discovery is channel capability. If AI Agent A is communicating with a user using voice from a web page, and AI Agent A wishes to engage AI Agent B using passthrough mode, or even transfer, it will need to know whether or not possible agents support voice. As with other areas of capabilities, there is a possibility for negotiation. For example, AI Agent A might currently be serving user A using voice chat on a web page, but it can also support web chat. If AI Agent Z supports only web chat, it can be utilized and the communications session downgraded to web chat.

Each of these channels will come with their own parameters defining capabilities which need to be considered when both selecting the agent, and connecting to it. If User A is connecting to AI Agent A using web chat, the web chat widget might support images, carousels and links, but might not support embedded videos. When AI Agent Z is connected, it needs to know that it cannot send embedded videos. Furthermore, that agent cannot process images uploaded by users, that feature needs to be disabled from the UI used by the agent.

3.3.3. Lifecycle Management

A key functional component of the inter-agent protocol is agent lifecycle. The connection with AI Agent A will go through states, initiated when the connection is first made, and then eventually terminate once the task for which it was engaged has been completed. There may be intermediate states as well. For example, if the downstream agent needs to perform tasks which have asynchronous components, the agent may need to go into a suspended mode while it waits for a response. This needs to be passed upstream to the invoking agent, so that it can take the appropriate action. It may elect to suspend itself as well, or may choose to retake control and engage with the user in other conversations while the other AI Agent works to complete its tasks.

There will need to be functionality in the protocol allowing the invoking agent to terminate communications with the target agent, either at the behest of the end user or under its own control.

Different agents may also support different types of lifecycles, which needs to be communicated. For example, AI agents can have different properties:

1. Transactional vs. Conversational: A transactional AI agent receives an instruction in natural language format that requests a task to be performed. Once it performs it, the response comes back. There is no back-and-forth with the requester of the task. As an example, an AI agent may provide transactional capabilities for image creation, sending a request along the lines of "Create an image of a solar system with three green planets orbiting a red sun, taken from above". The response that comes back is the requested image. An AI Agent might support transactional interactions, allowing back and forth with the end user. This is possible with the image generation example, where the user could receive the resulting image and then request modifications.

2. Synchronous vs. Asynchronous: A Synchronous AI agent executes communications sessions with a discrete begin and end, interacting back and forth with the invoker. There may be

timeouts if there is inactivity. For these type of agents, all of the work performed by the AI Agent occurs in real-time and can happen during the lifetime of the session. The image generation agent is a good example of it. An AI Agent that is asynchronous needs to perform actions which have a lengthy or even potentially unbounded amount of time to complete. This can happen when tasks take significant computation (for example, a research AI Agent which analyzes documents and prepares a summary, which can take several hours). This can also happen when an AI Agent needs to engage with a human. Using our loan example, AI Agent Z may be a loan approval AI Agent, and depending on the requested loan, may require approval from a human. If the request is made off-hours, a human may not be available until the next morning to service the request.

These two properties are orthogonal. An AI Agent can be transactional and synchronous - our AI image generation example above. But it can also be transactional and asynchronous. A good example of such an AI agent is an HR AI Agent used to approve a hire. This AI Agent would take a simple transactional request - "Obtain approval or disapproval to extend an offer to candidate Bharathi Ramachandran". This requires the AI Agent to gather data by invoking APIs and may require human approval from the head of HR. This can take potentially an unbounded amount of time.

For inter AI agents to work, these characteristics need to be transferred programatically from one agent to another, which then guides its operation.

Support for asynchronous AI Agents in particular requires additional protocol machinery. There is a need for some kind of callback, in which AI Agent Z can reconnect to AI Agent A. The end user may have also disconnected, requiring a reconnection. Whether such reconnection is possible or not depends on the communications channels and user configuration. For example, if User A was communicating with AI Agent A - an internal IT bot supporting a wide range of employee needs - using SMS, and user A asks AI Agent A to approve the hire, this might trigger AI Agent A to reach out to the HR AI Agent noted above. Once approval is obtain, AI Agent Z needs to re-initiate messaging backwards towards AI Agent A. AI Agent A then needs to send an SMS back to the end user. If, on the other hand, user A was connected to AI Agent A using web chat, reconnection to user A may not be possible. This could perhaps mean that the HR agent cannot be invoked at all, and an error needs to be returned to the user. Or, the HR AI Agent may support a modality wherein it emails the employee with the approval decision. These constraints and possibilities must be negotiated between agents to drive appropriate behavior.

Asynchronous agents might also support polling for state updates, wherein the end user can request updates on progress. This will require protocol machinery. Polling for updates could happen several different ways. During an active synchronous session between user A and AI Agent A, the user can request an operation that triggers a transactional async task to be sent to AI Agent Z (again the HR approval agent in our use case). The end user might continue its dialog with AI Agent A (the IT AI Agent) inquiring about an open ticket for a computer problem. During that chat, the user might ask for an update. For example, "Has the HR approval been completed yet by the way?". AI Agent A needs to then poll for an update from AI Agent Z so it can return the result. In this case, the interaction between AI Agent A and Z is more like an API. AI Agent A may not even remember (due to a failure or reconnection) that it has outstanding tasks with AI Agent Z. This requires API support for enumeration of pending tasks on downstream agents.

3.3.4. Data Transferrance

During the lifetime of the interconnection between agents, information will need to be transferred back and forth. The protocol will need to allow for this.

One dimension of protocol requirements are related to capabilities, already noted in the section above.

3.3.4.1. Context Transfer

Another aspect of data transferrance is context transfer. When AI Agent A first connects to AI Agent Z, it will likely want to transfer context about the user, about the conversation, about itself, and about the task that was requested by the end user. This information will be necessary for the invoked agent to be effective. Coming back to our travel use case, when the travel AI Agent invokes the flight booking agent, the flight booking agent will need to know:

1. The name and other contact information for the end user (see discussion below on authentication and authorization),

2. The name and description of the invoking AI agent, so that the flight booking agent can greet appropriately. For example, "Hi Alice, I’m the United Airlines agent. I understand that you’ve been booking a trip with Expedia and need a flight?"

3. The transcript of the conversation to date. For example, if the user had been debating a few options for departure cities and finally settled on New York to LA (instead of Philadelphia to LA) this could be relevant for the booking agent to convey additional

information not known to the travel agent. For example, "Hi Alice, I can help book your flight from New York to LA. However, you may not have been aware that flight costs departing from New York are higher in the summer, and you'd probably be better off traveling from Philadelphia on that day."

To facilitate context transfer, the receiving AI Agent may want to convey, programmatically, to the invoking AI Agent, the set of contextual information it is prepared to process. It might want to only see the conversation transcript. Or, it might want user background. There may be syntactic constraints, such as context length and character sets. Context transfer also introduces security considerations, as noted below.

3.3.4.2. Mid-Session Data Transfer

For conversational sessions, messages will be sent back and forth amongst the participants (user A, AI Agent A and AI Agent Z). The end user will originate new text messages, images or other user provided content, which are in essence targeted at the invoked AI agent, Agent Z. The invoked AI Agent will similarly wish to transfer messages back to the end user. In the case where AI Agent A is in a conference with AI Agent Z and the end user - or when it is operating in passthrough mode - these messages need to be tagged as being sent by the user, or targeted to the user, so that they do not trigger responses from AI Agent A. However, there will be meta-data that needs to be communicated as well. AI Agent Z, when it completes its task, needs to convey this information back towards AI Agent A. This is meta-data and not meant to be rendered to the end user. Similarly, if AI Agent Z is asked to perform a task whose output is programmatic and meant to be consumed by AI Agent A, then it needs to be tagged in this way. As an example, when the flight booking AI Agent has completed its flight booking, it needs to pass information back to the travel AI Agent containing the flight number, origin and destination airport, booking code, and so on. This information might need to be stored or processed in a programmatic fashion by AI Agent A. AI Agent A may then take this information and turn it into a piece of rendered web content which can be shown to the user. In this case, the result of the flight booking is meta-data, and needs to be sent by AI Agent Z, targeted towards AI Agent A.

These use cases require protocol machinery to facilitate data typing, data source and destination identification, and up front negotiation on what is permitted and what is not.

3.3.5. Authentication and Authorization

AuthN and AuthZ are significant considerations for inter-agent communications.

In many cases, the behaviours taken by the invoked AI Agent depend on the identity of the user that connected to the invoking agent. How does the invoked agent determine this identity reliably? There are, broadly speaking, two approaches:

1. Conveyed Identity: The invoking AI agent authenticates the end user, and passes this identity in a secure fashion to the invoked AI Agent

2. Re-Authentication: The invoked AI Agent separately authenticates the end user

In the case of conveyed identity, standards are needed for how the identity is communicated to the invoked AI agent. This undoubtedly involves the transferrance of the token. That transferrance itself needs to be secured. This is of particular concern in inter-domain use cases, to be sure that the token is only accessible by the AI Agent to which it is destined. Invoked AI Agents may have requirements about the strength of the authentication that has been performed. For example, if the invoking AI Agent authenticated the user via proof of knowledge techniques as discussed in Section XX, an invoked AI Agent may require stronger authentication of the user before proceeding.

In the case of re-authentication, standards are also required that allow for such authentication to occur. In the simplest case, the user has credentials with domain B, and meta-data needs to be pushed to the user to trigger an authentication flow, ideally with a web popup. Things are even more complicated when the user doesn’t hold credentials with domain B, and authentication is done via proof-of-knowledge of PII over the communications channel. If AI Agent A is party to these communications, it could reveal PII to AI Agent A which it should no be privy to. In such cases, there would need to be a way for the communication to be encrypted e2e between AI Agent B and the user, or else a way for the authentication to happen out of band without the involvement of AI Agent A.

Authorization is similarly complex. The core objective is that the end user have the ability to control - with certainty (not subject to hallucination error), the actions that the invoked AI Agent can take. This is similar to the core requirement given an AI agent permission to access APIs, where the authorization can be controlled via traditional OAuth grant flows, as described in Section XX. This is

more complex in he case of AI Agents. The whole idea of an AI Agent is that it can perform a wide range of tasks - many of which are not predetermined. How then can a user grant permissions to an AI Agent to limit what tasks it can perform, when those tasks are not simply an enumerated set of APIs within a specific scope? This would need to be sorted out.

3.3.6. User Confirmation

The same requirements for user confirmation as discussed in Section XX for AI agent to API actions, apply to AI agent to AI Agent communication to. When the invoked AI Agent is about to perform an action by invoking APIs with side effect, a confirmation must be generated by the invoked AI Agent, passed back to the invoking AI Agent and then communicated to the end user. The user confirmation is then passed back to the invoked AI Agent. The inter-agent aspect of this confirmation raises additional security considerations. How can the invoked AI agent trust that the confirmation actually came from the end user, and is not instead a hallucination generated by the invoking AI Agent? This will also require protocol machinery to provider cryptographic assurances.

3.3.7. Prompt Injection Attacks

The communications between AI agents introduces additional concerns around prompt injection attacks. Specifically, it introduces a new threat vector. A malicious user A, while communicating with AI Agent A, can request services of a third AI Agent, AI Agent Z (the invoked AI Agent). User A can then send malicious messages, which are conveyed through AI Agent A to AI Agent Z, in an attempt to cause it to perform unanticipated actions with the APIs it has access to. The cascading of AI Agents also introduces another possibility - that user A induces a prompt injection attack against AI Agent A, in an attempt to get it to generate malicious content towards AI Agent Z.

Prompt injection attacks are notoriously difficult to prevent. But, the protocol needs to consider these cases and introduce techniques to aid in diagnoses, logging and attribution.

4. Analysis of Existing Protocols

This section briefly introduces the Model Context Protocol (MCP), the Agent-to-Agent Protocol (A2A) and the Agntcy Framework, and maps them into the framework described in this document.

4.1. MCP

The Model Context Protocol (ref) enables operations between a client - typically an AI Agent acting on the user’s behalf - to access services provided by servers running on the same local host as the client. MCP defines three distinct services that servers can offer, which are:

1. Resources: These are files, DB records, log files, and other forms of static content.

2. Prompts: These are AI prompts, which can perform a transactional operation, taking input and returning an output

3. Tools: There are APIs, which provide programmatic functions on a set of given inputs, providing a specified output. An example is a shell command on a terminal.

MCP provides a way for clients to enumerate these services, obtain descriptions of them, and then access them.

Though MCP is targeted for operation within a single host and not over the Internet, we can extrapolate its usage over the Internet and map it into the framework here. In this framework, MCP covers AI Agent to API communications. Because it is meant for intra-host communication, it does not address many of the use cases around intra-domain and inter-domain security discussed in this framework.

4.2. A2A

The Agent-to-Agent (A2A) protocol focuses on - as the name implies - communications between agents. It specifies an "agent card" which is a programmatic definition of an AI agent which can be discovered by other AI Agents. It provides a protocol offer JSON-RPC for one AI Agent to invoke a task on another agent. If offers protocol machinery for lifecycle management of the task, including completing it and passing back artifacts. It facilitates messages between users and the invoked AI agent. It provides for synchronous and asynchronous invocations.

As the name implies, A2A fits within this framework as an agent to agent protocol. It provides for some of the functional requirements outlined in this framework, most notably lifecycle management, data and meta-data passing. It also offers discovery. It does not address the channel capability use cases discussed in Section XX. It does offer authentication and authorization, largely off-the-shelf OAuth using OpenAPI's authentication flows. It does not cover the use cases identified in Section XX. It doesn't provide the user confirmation functionality described in Section XX.

4.3. Agntcy

Agntcy is a framework, not a protocol per se. Like A2A, if focuses on agent to agent communications. It discusses discovery, hypothesizing the use of a DHT for discovery of agents. It makes use of the Open Agentic Schema Framework (OASF) to provide a standardized way for agents to declare their skills and be discoverable based on searches for those skills. It enables the transferrance of an agent manifest which describes the agent and how to connect to it. It considers cases where the agent is downloaded and run as a docker image or langchain library, in addition to being access over the Internet. It discusses an Agent Connect Protocol (ACP) which is similar in concept to Googles A2A protocol, enabling agents to talk to each other. Each agent declares, through meta-data, the configuration and invocation techniques it supports, including sync and async techniques.

Agntcy is similar to Google A2A and fits into this framework as an AI Agent to AI Agent Protocol. It covers discovery, which is a heavy focus of the framework. Its manifests meet some of the requirements around routing, focusing on skills. It does not address the capability considerations discussed in Section XX. The ACP protocol provides basic invocations but does not address the full scope of lifecycle management discussed here. It facilitates data transferrance but doesn't address the full set of use cases described in Section YY. It doesn't address AuthN or AuthZ and doesn't have any explicit support for user confirmations.

5. Conclusions

In conclusion, the framework described in this document covers a range of use cases and requirements for AI Agent interconnection. Many of the use cases and requirements cannot be satisfied with existing protocols and standards, leaving ample room for additional standards activity to be undertaken. Existing standards work, done outside of the IETF, covers some but not all of the use cases and requirements defined in this framework.

6. Informative References

[RFC3261] Rosenberg, J., Schulzrinne, H., Camarillo, G., Johnston, A., Peterson, J., Sparks, R., Handley, M., and E. Schooler, "SIP: Session Initiation Protocol", RFC 3261, DOI 10.17487/RFC3261, June 2002, <https://www.rfc-editor.org/info/rfc3261>.

[RFC3550] Schulzrinne, H., Casner, S., Frederick, R., and V. Jacobson, "RTP: A Transport Protocol for Real-Time Applications", STD 64, RFC 3550, DOI 10.17487/RFC3550, July 2003, <https://www.rfc-editor.org/info/rfc3550>.

[RFC768] Postel, J., "User Datagram Protocol", STD 6, RFC 768, DOI 10.17487/RFC0768, August 1980, <https://www.rfc-editor.org/info/rfc768>.

[RFC791] Postel, J., "Internet Protocol", STD 5, RFC 791, DOI 10.17487/RFC0791, September 1981, <https://www.rfc-editor.org/info/rfc791>.

[RFC9000] Iyengar, J., Ed. and M. Thomson, Ed., "QUIC: A UDP-Based Multiplexed and Secure Transport", RFC 9000, DOI 10.17487/RFC9000, May 2021, <https://www.rfc-editor.org/info/rfc9000>.

[RFC9114] Bishop, M., Ed., "HTTP/3", RFC 9114, DOI 10.17487/RFC9114, June 2022, <https://www.rfc-editor.org/info/rfc9114>.

[RFC9293] Eddy, W., Ed., "Transmission Control Protocol (TCP)", STD 7, RFC 9293, DOI 10.17487/RFC9293, August 2022, <https://www.rfc-editor.org/info/rfc9293>.

Authors' Addresses

Jonathan Rosenberg
Five9
Email: jdrosen@jdrosen.net

Cullen Jennings
Cisco
Email: fluffy@cisco.com