Web Authorization Protocol
Internet-Draft
Intended status: Experimental
Expires: 9 May 2026

Y. Song
L. Li
Huawei
F. Liu
Huawei Singapore

5 November 2025

# OAuth2.0 Extension for Multi-AI Agent Collaboration: Applier-On-Behalf-Of Authorization
draft-song-oauth-ai-agent-collaborate-authz-00

## Abstract

This draft proposes an authorization method for task-oriented multi-AI agent collaboration scenarios, where a leading agent coordinates sub-AI agents to complete complex tasks. The method extends the OAuth 2.0 protocol by adding an optional applier_id field, enabling the leading agent to apply for access tokens on behalf of other sub-AI agents. This approach greatly simplifies the sub-AI agents' authorization process, avoids efficiency loss from repeated interactions between multiple sub-AI agents and the authorization server, meanwhile maintaining full compatibility with existing OAuth 2.0 workflows.

## Requirements Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC2119 [RFC2119].

## Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at https://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on 9 May 2026.

Table of Contents



## 1. Introduction

AI agents are capable of handling tasks. They can integrate multiple technologies such as natural language understanding, data analysis, and logical reasoning to meet multi-step and cross-scenario requirements. However, a single AI agent has a limited knowledge scope and restricted functions, making it difficult to independently complete complex tasks that span multiple professional domains and require different capabilities.

Collaboration among a group of AI agents is an essential approach for such complex tasks. Taking the demand for "real-time health advice" as an example, this intent can be split into three tasks: "collect the user's health data", "predict the user's health status" and "provide health advice based on health status". These tasks need to be accomplished by corresponding professional sub-AI agents: the data collection agent is responsible for privacy-compliant data

collection, the health status prediction agent invokes medical models for analysis, and the advice generation agent outputs solutions based on health guidelines.

What’s more, accordingto some existing research[multi-agent-research-system], a leading agent is necessary for coordination. In the above example, the leading agent can receive the intent "give me real-time health advice" from user, understand it and split it into tasks. Additionally, the leading agent can distribute these tasks to sub-AI agents. Finally, the leading agent can integrate the results of sub-AI agents to form a complete health advice. Without the leading agent, tasks are prone to confusion and results are difficult to integrate.

In this collaborative scenario, if each sub-AI agent applies for permissions from the authorization server separately, the authorization server needs to verify the identity of each sub-AI agent one by one and generate access tokens for them. This will lead to problems such as frequent interactions, high computing resource consumption of the authorization server and low efficiency of the token issuance process.

Therefore, this draft puts forward an authorization method: configure "task scheduling" and "result integration" rights for the leading agent, and let the leading agent interface with the authorization server centrally. This method can greatly simplify the authorization process of sub-AI agents and avoid efficiency loss caused by repeated interactions.

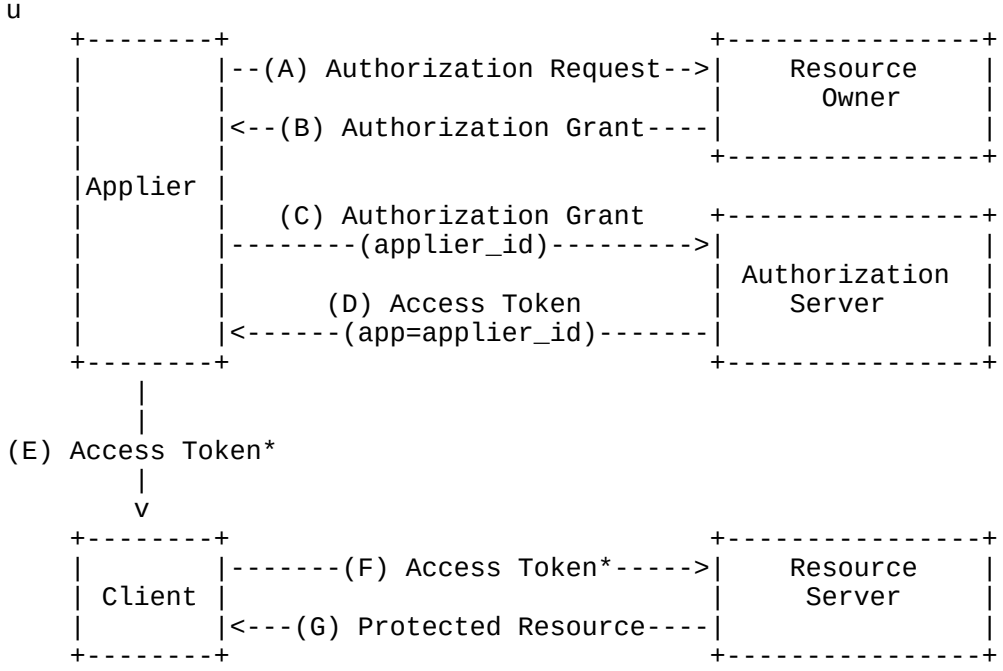
## 2. Terminology

Applier: a role that request the access token on be half of client(s), e.g. a leading AI agent.

This draft uses the terms "authorization server", "client", "resource server" defined by OAuth 2.0 [RFC6749]

## 3. Extention of Protocol Flow

This protocol flow is modeled after [RFC6749] and extends it with a *applier_id* field to enable authorization request by the applier.




## 3.1. Extension for Client Registration

Before initiating the protocol, the applier registers with the authorization server, as defined in [RFC6749].

In this draft, the applier’s role or capability is incorporated into its registration process. For example, the applier’s profile includes a "capability" parameter with the value "resolve intent and distribute tasks".

## 3.2. Authorization Request & Response

This section refers to step (A) and (B).

The applier receives a requirement from the user, resolves it and determines the task(s) and client(s) needed to fulfill the requirement. For example, when the user sends the intent "give me real-time health advice" to the leading AI agent, the leading AI agent resolves this intent and identifies three tasks: task 1 (collect the user’s health data), task 2 (predict the user’s health status) and task 3 (provide health advice based on health status). The leading agent then utilizes the agent discovery process to select sub-AI agent(s) capable of performing the respective tasks. For instance, task 1 assigned to sub-AI agent 1, task 2 to sub-AI agent

2, and task 3 to sub-AI agent 3. In the subsequent authorization process proposed in this draft, the leading AI agent is designated as the applier, and the selected sub-AI agent(s) are designated as the client(s).

The applier requests authorization from the resource owner, as defined in [RFC6749].

### 3.3. Access Token Request & Response

This section refers to step (C) and D).

The applier requests an access token by authenticating with the authorization server and presenting the authorization grant, as defined in [RFC6749]. The message may also include the following parameter:

applier_id

OPTIONAL. The identifier of the applier which is requesting access token on be half of the client(s).

The authorization server authenticates the applier and validates that the applier is capable of requesting access tokens on behalf of client(s). For example, it checks whether the applier’s profile includes a "capability" parameter whose value includes "distribute tasks".

If the validation is successful, the authorization server may verify grants as defined in [RFC6749]. Then the authorization server generates access token and send access token response message to the applier. The access token is similar to OAuth2.0, except that it consists of an additional claim to specify the applier.

app

OPTIONAL. The identifier of the applier which is requesting access token on be half of the client(s).

The access token includes one *app* parameter and multiple *sbj*-*aud*-*scope* pairs. Below is an example:

```
{

"iss": "authorization server ID",

   "app": "leading agent ID",
```

```
        "sbj": "sub AI agent1 ID",
        "aud": "resource server 1 ID",
        "sbj": "sub AI agent 2 ID",
        "aud": "resource server 1 ID", "resource server 2 ID",
        "sbj": "sub AI agent 3 ID",
        "aud": "resource server 2 ID", "resource server 3 ID"
        ...
     }
```

If the validation fails, the access token response message may include the reason for failure. [RFC6749] has defined types of error response. In this case, the authorization server may use a new error message "unauthorized_applier" to indicate that the applier is not capable of requesting access tokens on be half of client(s).

### 3.4. Access Token Transmit

This section refers to step (E).

The applier sends the task ID and access token* to the client(s). Based on local policies and regulations, the applier may use privacy protection mechanisms to process the access token. Thus, the access token* may be the same as the access token, or generated from the access token using privacy protection mechanisms. For example, the applier uses a selective disclosure algorithm to generate access token 1, access token 2, and access token 3 from the access token received in step (D). Access token 1 may include the leading agent ID and sub-AI agent 1 ID; access token2 may include the leading agent ID and sub-AI agent 2 ID; access token3 may include the leading agent ID and sub-AI agent 3 ID. The applier may then send access token 1 to sub-AI agent 1, access token 2 to sub-AI agent 2, and access token 3 to sub-AI agent 3.

The client(s) validate the proof of the received access token* and check whether the applier ID refers to a trustworthy applier. For example, each AI agent is preconfigured with a list of trusted AI agents, and any one of the trusted AI agents that acts as a leading agent will pass the validation.

If the validation fails, the client(s) may send a response message to the applier with a failure indication of "unknown_applier".

### 3.5. Access Token Verify

This section refers to step (F) and (G).

The client(s) execute the task according to message received in step (E). When resources are needed, the client(s) request the protected resource from the resource server and present the access token*, as defined in [RFC6749]. The resource server validates the access token*, as defined in [RFC6749]. If the access token* includes multiple *sbj*-*aud* pairs, the resource server may verify that its own ID and the client’s ID are in the same pair. In previous example of the access token, "sub-AI agent 1 ID"and "resource server 1 ID" are in the same pair.

If the validation succeeds, the resource server may provide the resource to the client(s).

## 4. Security Considerations

TBD

## 5. References

### 5.1. References

### 5.2. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/rfc/rfc2119>.

[RFC6749] Hardt, Dick., "The OAuth 2.0 Authorization Framework", RFC 6749, DOI 10.17487/RFC6749, October 2012, <https://www.rfc-editor.org/rfc/rfc6749>.

### 5.3. Informative References

[multi-agent-research-system] Anthropic, "How we built our multi-agent research system", 13 June 2025.

## Authors’ Addresses

Yurong Song
Huawei
Email: songyurong1@huawei.com

Lun Li
Huawei
Email: lilun20@huawei.com

Faye Liu
Huawei Singapore
Email: liufei19@huawei.com