Independent Submission
Internet-Draft
Intended status: Informational
Expires: 5 October 2026

M. Besozzi
TwoGenIdentity

3 April 2026

OAuth 2.0 Agents Native Authorization via Structured Elicitation
draft-embesozzi-oauth-agent-native-authorization-00

Abstract

This document defines a Structured Elicitation extension to the OAuth 2.0 First-Party Applications (FiPA) specification [FiPA], establishing a standard metadata format for FiPA authorization challenge responses. FiPA leaves the format for advertising available authenticators and their required inputs undefined, preventing interoperable implementation by AI Agents and other non-browser clients. This extension adds an elicitations array to the FiPA Authorization Challenge Response, providing a standard metadata format for authenticator challenges. Model Context Protocol (MCP) Elicitation [MCP-Elicitation] is the normative reference binding.

This specification covers the just-in-time (JIT) authorization for AI Agents executing on behalf of users in Human-to-Agent (H2A) flows.

About This Document

This note is to be removed before publishing as an RFC.

The latest revision of this draft can be found at https://embesozzi.github.io/draft-embesozzi-oauth-agent-native-authorization/draft-embesozzi-oauth-agent-native-authorization.html. Status information for this document may be found at https://datatracker.ietf.org/doc/draft-embesozzi-oauth-agent-native-authorization/.

Source for this draft and an issue tracker can be found at https://github.com/embesozzi/draft-embesozzi-oauth-agent-native-authorization.

Status of This Memo

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

Copyright Notice

Table of Contents

1. Introduction

1.1. Applicability

AI Agents operate autonomously and do not inherently distinguish routine operations from sensitive ones. When an agent encounters an operation requiring step-up authentication or explicit user approval, a just-in-time (JIT) authorization gate is needed: the agent must pause, surface a structured challenge to the human user, and obtain a cryptographic proof — an OAuth token — before proceeding. The FiPA specification [FiPA] provides the challenge/response wire protocol for this gate. This extension defines the metadata format that makes FiPA challenges machine-readable for agent runtimes.

This extension applies when the client is an AI Agent acting on behalf of a human user (see Section 1.2), the authorization server supports the FiPA challenge/response protocol, and the agent runtime supports a Structured Elicitation mechanism (Section 4), with MCP Elicitation [MCP-Elicitation] as the normative reference implementation.

1.2. Human-to-Agent (H2A) Communication Model

This extension addresses the Human-to-Agent (H2A) communication pattern. In this pattern, a human user interacts with an AI Agent that acts as a first-party OAuth client on their behalf. The AI Agent orchestrates the FiPA challenge/response cycle but cannot independently satisfy authentication challenges that require human interaction — such as entering a TOTP code or completing a WebAuthn ceremony.

The Structured Elicitation mechanism defined in Section 4 is the channel through which the authorization server communicates challenge requirements back to the human user via the agent runtime. The agent runtime presents the challenge to the human, collects the response, and forwards it to the authorization server as an Authorization Challenge Request.

This contrasts with Agent-to-Agent (A2A) patterns, in which an autonomous agent satisfies authorization challenges without human involvement. The present document is scoped to H2A interactions; A2A authorization is outside the scope of this specification.

2. Conventions and Definitions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 RFC2119 [RFC8174] when, and only when, they appear in all capitals, as shown here.

2.1. Terminology

This document uses terminology defined in [RFC6749], [FiPA], and [MCP-Elicitation]. The following terms are used as defined:

* *Authorization Server (AS):* As defined in [RFC6749].

* *Just-In-Time (JIT) Authorization:* An authorization pattern in which a fresh credential challenge is issued at the moment a sensitive action is attempted, rather than relying solely on the trust established at initial authentication. This extension enables JIT Authorization by providing the structured metadata needed for agent runtimes to surface mid-flow challenges to the human user.

* *Structured Elicitation:* The abstract mechanism by which an agent runtime requests structured user input mid-flow and returns a structured response. MCP Elicitation [MCP-Elicitation] is a conformant implementation of this mechanism and serves as the normative reference implementation for this specification.

* *Human-to-Agent (H2A) Communication:* The interaction pattern in which a human user delegates actions to an AI Agent that acts as a first-party OAuth client. The agent orchestrates protocol flows on behalf of the user.

* *First-Party AI Agent:* An AI agent whose client runtime is built and controlled by the same operator as the server-side components and authorization server. The agent MAY implement FiPA-specific elicitation extensions.

* *Third-Party AI Agent:* An AI agent whose client runtime is a product provided by a third party. The implementer cannot modify client elicitation handling.

3. Protocol Overview

This extension operates within the FiPA challenge/response cycle defined in [FiPA]. A FiPA authorization request that requires additional authentication receives an Authorization Challenge Response:

```
HTTP/1.1 400 Bad Request
Content-Type: application/json

{
  "error": "insufficient_authorization",
  "auth_session": "sess_abc123"
}
```

The auth_session binds all subsequent requests to this authentication context. FiPA supports multi-step flows via continued challenge/response cycles on the same auth_session.

This extension adds an elicitations array to the Authorization Challenge Response. Each entry in the array describes one authenticator challenge the human user must complete. The agent runtime translates each entry into a Structured Elicitation request directed at the human user, collects the response, and submits it in the next Authorization Challenge Request.

Upon completion of the challenge/response cycle, the authorization server issues an authorization code or token bound to the auth_session. This token serves as cryptographic proof that a human user satisfied the required authentication challenge at a specific point in time, enabling the MCP server to enforce JIT authorization before executing sensitive operations.

4. Elicitation Extension

4.1. Response Format

The elicitations array is added to the FiPA insufficient_authorization error response body. Each entry carries the parameters needed for the agent runtime to present a structured challenge to the human user.

Fields per entry:

Field	Required	Description
mode	REQUIRED	Interaction mode. "form" for in-band structured data collection using a JSON Schema.
message	REQUIRED	Human-readable prompt displayed to the user by the agent runtime.
requestedSchema	Form mode only	A restricted JSON Schema object describing the fields the user must fill in. Maps 1:1 to the Structured Elicitation requestedSchema.

Table 1

Note: A "url" mode (out-of-band browser interaction) is defined in [MCP-Elicitation] and may be used in future revisions of this specification for authenticator types that require a browser ceremony (e.g., Passkeys in runtimes that cannot invoke the WebAuthn API natively).

4.1.1. Request Content Type for Intermediate Requests

The initial Authorization Challenge Request MUST use application/x-www-form-urlencoded as required by [FiPA]. Intermediate requests — those that carry an auth_session obtained from a prior challenge response — MAY use application/json as the content type. This is consistent with [FiPA], which explicitly permits intermediate requests to use a different format than the initial request.

4.1.2. Elicitation Sequencing

The elicitations array typically contains one entry at a time. Because subsequent steps depend on prior user selections (e.g., choosing TOTP versus Passkey changes the next challenge), the authorization server issues each step’s elicitation in a new Authorization Challenge Response after the preceding credential is submitted.

4.1.3. Clients Without Structured Elicitation Support

For clients that do not support Structured Elicitation, the authorization server returns a standard OAuth error response per [RFC6749] §5.2. The WWW-Authenticate response header per [RFC6749] §5.2 is one example of how the error is surfaced at the HTTP level:

```
WWW-Authenticate: Bearer realm="authorization-server",
                  error="insufficient_scope",
                  auth_session="sess_abc123"
```

4.2. Runtime Bindings

The elicitations array is a transport-agnostic FiPA extension. The agent runtime is responsible for translating each entry into its native interaction mechanism. This section defines bindings for known runtimes.

4.2.1. MCP Elicitation Binding

When the agent runtime is MCP-based, the runtime (acting as the FiPA client) maps each elicitations entry directly to an elicitation/create params object per [MCP-Elicitation]. The jsonrpc, id, and method fields are MCP transport framing — the params content is identical to the elicitations entry.

The following example shows a form mode elicitation entry from the Authorization Challenge Response alongside the resulting elicitation/create call:

FiPA elicitations entry (from Authorization Challenge Response):

```
{
  "mode": "form",
  "message": "Additional verification is required. Select your authentication method."
,
  "requestedSchema": {
    "type": "object",
    "properties": {
      "authenticator": {
        "type": "string",
        "title": "Authentication Method",
        "oneOf": [
          { "const": "totp", "title": "Authenticator App (TOTP)" },
          { "const": "passkey", "title": "Passkey" }
        ]
      }
    },
    "required": ["authenticator"]
  }
}
```

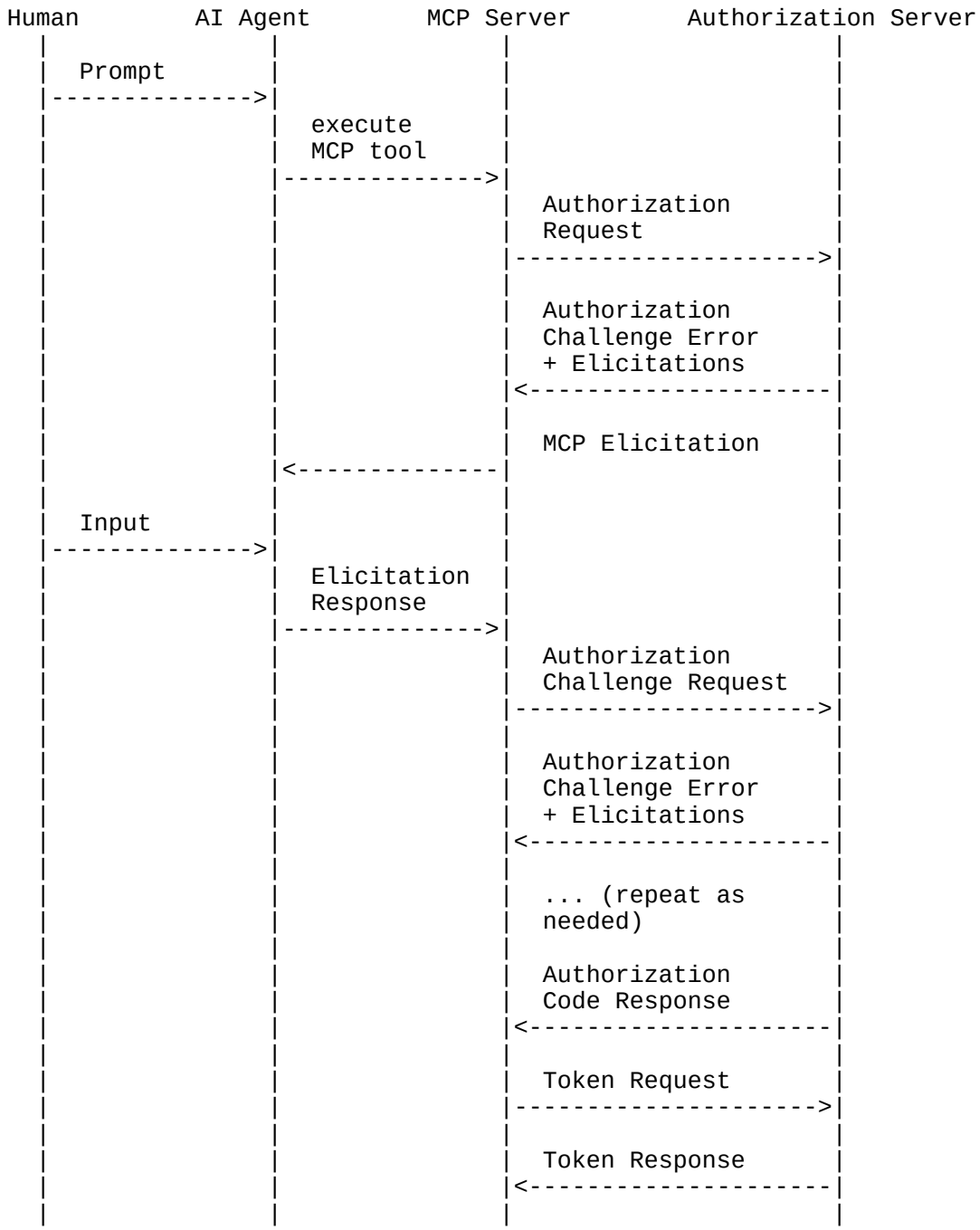
Resulting MCP elicitation/create call issued by the MCP server:

```
{
  "jsonrpc": "2.0",
  "id": "elicit-1",
  "method": "elicitation/create",
  "params": {
    "mode": "form",
    "message": "Additional verification is required. Select your authentication method
.",
    "requestedSchema": {
      "type": "object",
      "properties": {
        "authenticator": {
          "type": "string",
          "title": "Authentication Method",
          "oneOf": [
            { "const": "totp", "title": "Authenticator App (TOTP)" },
            { "const": "passkey", "title": "Passkey" }
          ]
        }
      },
      "required": ["authenticator"]
    }
  }
}
```

MCP client response:

```
{
  "jsonrpc": "2.0",
  "id": "elicit-1",
  "result": {
    "action": "accept",
    "content": { "authenticator": "totp" }
  }
}
```

4.2.1.1. MCP Elicitation Flow Example


Human
AI Agent
MCP Server
Authorization Server
Prompt
execute MCP tool
Authorization Request
Authorization Challenge Error + Elicitations
MCP Elicitation
Input
Elicitation Response
Authorization Challenge Request
Authorization Challenge Error + Elicitations
... (repeat as needed)
Authorization Code Response
Token Request
Token Response

4.2.2. HTTP API Binding

TODO: future section

5. Authorization Challenge Protocol

5.1. Authorization Challenge Response

When the authorization server requires additional authentication, it returns an insufficient_authorization error response containing an elicitations array. Each entry in elicitations describes one structured challenge the agent runtime MUST present to the human user.

The response includes the following parameters:

Parameter	Required	Description
error	REQUIRED	MUST be "insufficient_authorization".
auth_session	REQUIRED	Session identifier that MUST be included in all subsequent Authorization Challenge Requests for this context.
elicitations	REQUIRED	Array of elicitation objects. Each entry contains mode, message, and (for "form" mode) requestedSchema. See the Appendix for authenticator-specific schema examples.

Table 2

```
HTTP/1.1 400 Bad Request
Content-Type: application/json
Cache-Control: no-store

{
  "error": "insufficient_authorization",
  "auth_session": "<session-identifier>",
  "elicitations": [
    {
      "mode": "form",
      "message": "<human-readable prompt>",
      "requestedSchema": { ... }
    }
  ]
}
```

5.2. Authorization Challenge Request

After collecting user input via Structured Elicitation, the agent runtime submits an Authorization Challenge Request to the authorization challenge endpoint.

Parameter	Required	Description
auth_session	REQUIRED	The session identifier from the preceding Authorization Challenge Response.
response	REQUIRED	An object whose fields match the requestedSchema advertised in the corresponding elicitations entry.

Table 3

```
POST /as/authorization.oauth2
Content-Type: application/json

{
  "auth_session": "<session-identifier>",
  "response": { ... }
}
```

On success, the authorization server issues an OAuth token bound to the completed auth_session, per [FiPA].

6. Security Considerations

TBD

7. IANA Considerations

TBD

8. References

8.1. Normative References

[FiPA] IETF OAuth Working Group, "OAuth 2.0 First-Party Applications", n.d., <https://datatracker.ietf.org/doc/draft-ietf-oauth-first-party-apps/>.

[MCP-Elicitation] Anthropic, "Model Context Protocol Specification — Elicitation", n.d., <https://modelcontextprotocol.io/specification/>.

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

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/rfc/rfc8174>.

[RFC9700] Lodderstedt, T., Bradley, J., Labunets, A., and D. Fett, "Best Current Practice for OAuth 2.0 Security", BCP 240, RFC 9700, DOI 10.17487/RFC9700, January 2025, <https://www.rfc-editor.org/rfc/rfc9700>.

[WebAuthn] W3C, "Web Authentication: An API for accessing Public Key Credentials, Level 3", n.d., <https://www.w3.org/TR/webauthn/>.

8.2. Informative References

[EAP-ACR] OpenID Foundation, "OpenID Connect Extended Authentication Profile (EAP) ACR Values 1.0", n.d., <https://openid.net/specs/openid-connect-eap-acr-values-1_0.html>.

Appendix A. Appendix

A.1. Examples

A.1.1. Authenticator Selection

When additional authentication is required, the first elicitation typically presents the available authenticators. The following examples show the FiPA wire format for authenticator selection. This specification defines elicitation schemas for two authenticator types: Passkey (WebAuthn) and Authenticator App (TOTP). Password authentication is outside the scope of this document.

When multiple authenticators are supported, the authorization server returns a selection prompt:

```
   HTTP/1.1 400 Bad Request
   Content-Type: application/json
   Cache-Control: no-store

   {
     "error": "insufficient_authorization",
     "auth_session": "sess_abc123",
     "elicitations": [
       {
         "mode": "form",
         "message": "Additional verification is required. Select your authentication meth
od.",
         "requestedSchema": {
           "type": "object",
           "properties": {
             "authenticator": {
               "type": "string",
               "title": "Authentication Method",
               "oneOf": [
                 { "const": "totp", "title": "Authenticator App (TOTP)" },
                 { "const": "passkey", "title": "Passkey" }
               ]
             }
           },
           "required": ["authenticator"]
         }
       }
     ]
   }
```

The agent submits the user’s selection:

```
   POST /as/authorization.oauth2
   Content-Type: application/json

   {
     "auth_session": "sess_abc123",
     "response": { "authenticator": "totp" }
   }
```

A.1.2. TOTP Challenge

TOTP uses form mode: the challenge is a 6-digit numeric code, expressible as a plain string with pattern and length constraints.

Authorization Challenge Response:

```
HTTP/1.1 400 Bad Request
Content-Type: application/json
Cache-Control: no-store

{
  "error": "insufficient_authorization",
  "auth_session": "sess_abc123",
  "elicitations": [
    {
      "mode": "form",
      "message": "Enter the 6-digit code from your Authenticator App.",
      "requestedSchema": {
        "type": "object",
        "properties": {
          "otp": {
            "type": "string",
            "title": "One-Time Password",
            "minLength": 6,
            "maxLength": 6,
            "pattern": "^[0-9]{6}$"
          }
        },
        "required": ["otp"]
      }
    }
  ]
}
```

Authorization Challenge Request:

```
POST /as/authorization.oauth2
Content-Type: application/json

{
  "auth_session": "sess_abc123",
  "response": { "otp": "847291" }
}
```

On success, the authorization server issues an OAuth token with the authenticated ACR value, per [FiPA].

A.1.3. Passkey Challenge

In a Third-Party AI Agent deployment, the client runtime is a product the implementer does not control. It supports standard Structured Elicitation form mode and URL mode but cannot be extended with custom format handling or WebAuthn API invocation.

For Third-Party AI Agents, URL mode SHOULD be used to redirect the user to a browser-based WebAuthn flow. The out-of-band WebAuthn ceremony is outside the scope of this specification.

For in-band Passkey support, a First-Party AI Agent deployment is required. The implementer controls the agent code and MAY implement custom elicitation handling, including native WebAuthn API invocation using the OS or platform FIDO2 APIs. The WebAuthn ceremony is performed in-band: no browser, no redirect.

A.1.3.1. WebAuthn Challenge Delivery

This section defines how the authorization server delivers WebAuthn challenge parameters to the agent through Structured Elicitation form mode.

TBD

A.1.3.2. Authorization Challenge Response

TBD

Author’s Address

Martin Besozzi
TwoGenIdentity
Email: embesozzi@twogenidentity.com