Independent Submission
Internet-Draft
Intended status: Experimental
Expires: 4 June 2026

M. Gaikwad

1 December 2025

The Web of Agents (WoA)
draft-gaikwad-woa-00

Abstract

This document defines the Web of Agents (WoA), a minimal JSON based description format and invocation convention that allows HTTP hosts to advertise AI agents and for clients to invoke those agents in a uniform way. A WoA document is typically served from a well known location on an HTTP origin and uses JSON Schema to describe agent inputs and outputs. WoA does not define a discovery protocol itself but is designed to be used as a host level primitive by higher level discovery systems.

Table of Contents

1. Introduction

AI agents are increasingly exposed as network reachable services, such as text based large language model interfaces, task specific tools, and composite workflows that call other services. Today these agents are typically documented using ad hoc JSON documents or provider specific formats. This makes it difficult for generic clients to discover which agents exist on a given origin and to invoke them in a predictable way.

The Web of Agents (WoA) defines:

* a JSON based document format, usually served from a well known path on an origin, that lists agents and their properties

* a JSON Schema based description of agent inputs and outputs

* a small set of transport configuration objects that define how a client constructs an invocation request

* a simple invocation envelope used by some transports

WoA is intentionally small in scope. It does not attempt to define a global discovery system or a capability vocabulary. It is designed to complement such systems by providing a host level primitive that discovery mechanisms can fetch, cache, and index. For example, a cross platform discovery system such as [CUI-AGENT-DISCOVERY] could use WoA documents as one of its inputs when building a search index.

This document is targeted at the Independent Submission Stream with category "exp". The intent is to provide a concrete and implementable format that can be experimented with by client and host implementers.

2. Conventions and Terminology

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.

This document uses the JSON data model defined in [RFC8259] and HTTP terminology from [RFC9110].

2.1. Terms

WoA document A JSON document served by an HTTP origin that describes one or more agents and their transport configurations according to this specification.

Agent A logical AI powered capability exposed by a host. An agent has an identifier, a human readable name and description, a JSON Schema based input and output description, and one or more transport options that describe how to invoke it.

Host An HTTP origin that serves a WoA document and optionally implements one or more transports for agents described in that document.

WoA Client A component that retrieves the WoA document and invokes agents. A WoA client may be a user facing application, an orchestration engine, or another independent agent, enabling agent to agent (A2A) interactions.

Transport A named mechanism that defines how a client constructs and sends an invocation request. This document defines the "rest" and "mcp" transports. Additional transports may be registered in the WoA Transport Registry.

3. Architecture

A typical deployment has an HTTP origin that serves a WoA document at a well known path, and one or more transports that accept invocation requests.

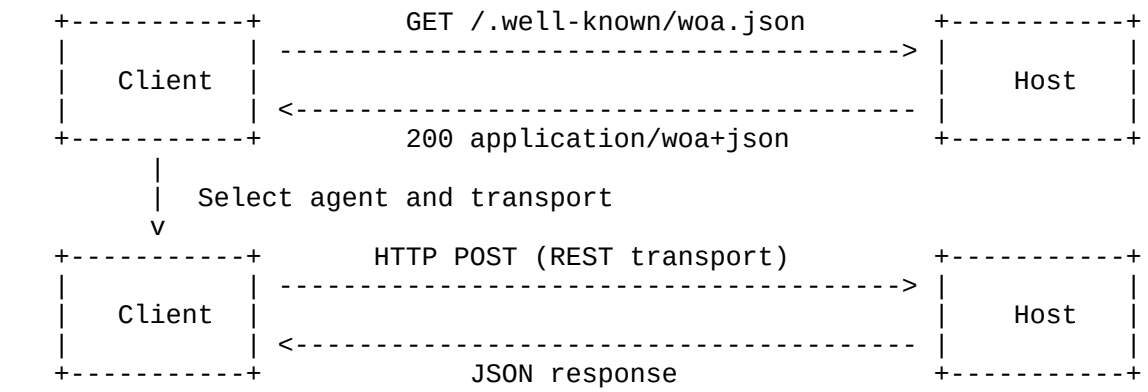

Figure 1: High Level WoA Architecture

A client fetches the WoA document from a host, selects an agent, chooses a transport supported by both client and host, constructs an invocation request according to the transport rules, and processes the response. Higher level discovery systems may crawl or index WoA documents but are out of scope for this specification.

4. WoA Document Schema

A WoA document is a single JSON object with the following top level fields:

* woa_version: A string identifying the WoA document version. For this specification the value MUST be the string "1".

* agents: An array of agent objects as defined in Section 4.1.

* transports: An object whose keys are transport names and whose values are transport configuration objects as defined in Section 4.3.

4.1. Agent Objects

Each element of the agents array is an object with the following fields:

* id (string, REQUIRED): A token that uniquely identifies the agent within the WoA document. The value MUST match the following ABNF (using [RFC5234]):

```
agent-id = 1*( ALPHA / DIGIT / "-" / "_" )
```

* name (string, REQUIRED): A short, human readable name for the agent.

* description (string, REQUIRED): A human readable description of what the agent does. Clients MUST treat this field as untrusted input for security reasons (see Section 8).

* version (string, OPTIONAL): A version identifier for the agent implementation, such as "1.0.0".

* capabilities (array of strings, OPTIONAL): A list of capability identifiers. This specification does not define a controlled vocabulary. Implementations MAY use URIs (for example, "https://example.com/capability/summarization") or URN like identifiers as long as they are stable.

* inputs (object, REQUIRED): A JSON Schema document that describes the expected request body when invoking the agent. See Section 4.2.

* outputs (object, REQUIRED): A JSON Schema document that describes the successful response body produced by the agent.

* transports (array of strings, REQUIRED): Names of transport configurations (keys in the top level transports object) that can be used to invoke this agent.

* operations (array of objects, OPTIONAL): A list of operation descriptors for agents that support multiple logical operations via a single transport endpoint. If operations is not present, the agent is considered to have a single unnamed operation.

Each operation object in the operations array has:

* name (string, REQUIRED): A token naming the operation.

* description (string, REQUIRED): Human readable description of the operation.

* inputs (object, OPTIONAL): JSON Schema overriding the agent level inputs for this operation.

* outputs (object, OPTIONAL): JSON Schema overriding the agent level outputs for this operation.

Clients MUST NOT assume that two agents with the same id on different origins have the same behavior or schema.

4.2. JSON Schema Usage for Inputs and Outputs

The inputs, outputs, and any operation level inputs or outputs fields MUST be valid JSON Schema documents compliant with the JSON Schema 2020-12 core and validation specifications [JSON-SCHEMA-CORE] [JSON-SCHEMA-VAL].

At minimum, implementations MUST support JSON Schema documents whose root is an object with:

* a type of "object"

* a properties object describing named properties

* an optional required array naming required properties

More advanced JSON Schema features such as oneOf, allOf, and format MAY be used by hosts and SHOULD be ignored by clients that do not understand them, as long as the client can still construct a conforming request.

Hosts SHOULD provide concise and accurate JSON Schema documents to enable static validation and tooling. Clients MAY validate request bodies locally against the inputs schema before sending them and MAY validate successful responses against the outputs schema.

4.3. Transport Configurations

The top level transports object maps transport names to configuration objects. Transport names are strings. This document defines the rest and mcp transports. Additional transports MAY be defined and registered in the WoA Transport Registry (see Section 9.3).

4.3.1. REST Transport

A rest transport configuration object MUST contain:

* base (string, REQUIRED): An absolute HTTPS URL that identifies the base of the REST API. The URL scheme MUST be "https" and the server SHOULD support TLS 1.3 [RFC8446].

* invoke_path (string, REQUIRED): An absolute path beginning with "/" that, when combined with base, yields the invocation URL template. The path MAY contain a query component. Within the path, the case sensitive substring "{agent_id}" MAY appear as a placeholder.

To construct the invocation URL, the client concatenates the base URL and the invoke_path value. If the invoke_path contains the case sensitive substring "{agent_id}", the client MUST replace it with the target agent’s id value.

Clients invoking an agent via the REST transport MUST use HTTP POST and MUST send a request body encoded as JSON with media type application/json. The request body for the REST transport is defined in Section 5. Hosts using the REST transport MUST respond with JSON and SHOULD use the application/problem+json media type for errors as specified in [RFC9457].

4.3.2. MCP Transport

A mcp transport configuration object describes how an agent is exposed via a Model Context Protocol (MCP) server [MCP]. The configuration object MUST contain:

* server (string, REQUIRED): An identifier or URL for the MCP server endpoint. The exact format is defined by the MCP specification.

* tool_namespace (string, REQUIRED): A string that identifies the logical namespace of tools on the MCP server.

* tool_field (string, REQUIRED): The name of the field in the invocation envelope that contains the tool name for this MCP server.

When using the MCP transport, the client translates the WoA invocation envelope into the appropriate MCP tool invocation as defined by the MCP specification [MCP]. For example, if tool_field is "agent", the client MAY pass the value of the envelope’s agent field as the tool name.

4.3.3. Private Transports

Hosts MAY define private transport names for experimental use. Private transport names MUST use a reverse DNS prefix, such as "com.example.mytransport", to avoid collisions with registered transports. Private transports MUST NOT be registered in the WoA Transport Registry.

5. Invocation

This section describes the invocation envelope and how it is used with the REST and MCP transports. Other transports MAY define their own invocation rules.

5.1. Invocation Envelope

When a transport uses a JSON request body that contains metadata about the invocation in addition to the agent specific input object, this document refers to that JSON object as the "invocation envelope".

For the REST transport defined in this document, the invocation envelope is an object with the following fields:

* agent (string, REQUIRED): The agent identifier as listed in the WoA document.

* operation (string, OPTIONAL): The operation name, if the agent defines an operations array. If omitted for an agent that defines operations, the name "default" is RECOMMENDED as the implied operation.

* input (object, REQUIRED): A JSON object that MUST conform to the JSON Schema specified in the selected operation’s inputs schema, or to the agent level inputs if there is no per operation override.

Transports MAY add additional fields to the invocation envelope as long as they do not conflict with the fields defined above.

5.2. REST Transport Invocation

When using the REST transport, the client constructs the invocation URL as described in Section 4.3.1 and sends an HTTP POST request with media type application/json and a request body containing the invocation envelope.

The agent field in the envelope MUST match the intended agent's id. If the REST invocation URL template includes an "{agent_id}" placeholder, the host can infer the agent from the URL. In that case:

* If the host derives an agent identifier from the URL and the agent field is absent in the envelope, the host MAY use the identifier from the URL.

* If the host derives an agent identifier from the URL and the envelope contains an agent field with a different value, the host MUST reject the request with HTTP status 400 (Bad Request). The host SHOULD return a application/problem+json body explaining the mismatch.

On success, the host MUST return a 2xx status code and a JSON response body that conforms to the JSON Schema specified in the selected operation's outputs schema, or the agent level outputs schema if there is no per operation override.

On error, the host SHOULD return a 4xx or 5xx status code and a response body using the problem details format defined in [RFC9457] with media type application/problem+json. Problem type URIs and error code taxonomies are deployment specific.

5.3. MCP Transport Invocation

When using the MCP transport, the client translates the invocation envelope into the appropriate MCP tool invocation. The exact mapping is defined by the MCP specification [MCP] and is not repeated here. As a typical pattern, the agent or operation fields of the envelope may map to tool names and the input object may map to MCP tool arguments.

WoA does not define the wire protocol for MCP and does not impose additional security requirements beyond those defined by the MCP specification. The security considerations in Section 8 still apply to the host that publishes the WoA document.

6. Authorization

WoA deliberately separates the question of "which agent exists and how can it be invoked" from "who is allowed to invoke this agent". The WoA document format does not contain access tokens and does not mandate a particular authorization protocol. Hosts MAY use OAuth 2.0 [RFC6749], mutual TLS, or other mechanisms to protect invocation endpoints.

A host MAY include additional metadata in the WoA document to help clients understand the authorization expectations for an agent. For example, a deployment using the SOUTH stochastic authorization protocol [I-D.gaikwad-south-authorization] might include an authz object with pointers to policy endpoints. Such metadata is deployment specific and out of scope for the core WoA specification.

7. Caching

WoA documents are likely to be fetched frequently by clients and discovery systems. Hosts SHOULD include HTTP caching headers, such as ETag, Last-Modified, and appropriate Cache-Control directives, on responses that carry a WoA document to reduce unnecessary network traffic. Clients SHOULD honor these headers when polling for updates.

8. Security Considerations

WoA is a descriptive format and does not execute code by itself, but it is used in contexts where agents may execute untrusted input or perform sensitive actions. Both hosts and clients need to consider the following security aspects.

8.1. Transport Security

Hosts that serve WoA documents and implement REST transports MUST use HTTPS. The underlying TLS version SHOULD be TLS 1.3 [RFC8446]. Clients MUST validate server certificates according to normal HTTPS rules. Plain HTTP MUST NOT be used for WoA documents or REST invocations on the open Internet.

8.2. Server Side Request Forgery and Origin Validation

A malicious WoA document could advertise transport URLs that point to internal or otherwise sensitive endpoints. Automated clients that unconditionally invoke agents based on WoA documents risk being used as a vector for server side request forgery.

Clients SHOULD validate that transport URLs provided in a WoA document resolve to allowed domains or IP address ranges before invoking agents. In particular, clients SHOULD NOT automatically send invocation requests to URLs that resolve to private or link local address ranges, such as those defined in [RFC1918], unless explicitly configured to do so by an administrator. Clients SHOULD also be cautious of HTTP redirects that change the authority component of the URL and SHOULD apply the same validation to the redirect target.

8.3. Indirect Prompt Injection

The name and description fields of an agent are free form strings controlled by the host. Clients that use a large language model to reason about or select agents based on these fields MUST treat them as untrusted input. A malicious host could embed text that attempts to influence the client side model in ways that bypass user intent, a pattern sometimes referred to as indirect prompt injection.

Implementers SHOULD design any agent selection logic that uses LLMs with appropriate input isolation, such as explicit system level instructions that describe which fields are trusted and which are untrusted, and with guardrails that restrict the actions that can be taken based on WoA content.

8.4. Agent Output Trust

WoA itself does not provide integrity or authenticity guarantees for agent outputs. Clients SHOULD treat agent responses as untrusted data and apply appropriate validation, sandboxing, and authorization checks before using them to take actions such as executing code, issuing network requests, or changing external state.

9. IANA Considerations

9.1. Well Known URI Registration

This document requests registration of a new well known URI suffix in the "Well Known URIs" registry as defined in [RFC8615].

The registration template is:

URI suffix: woa.json
Change controller: IETF
Specification document: This document
Status: permanent
Related information: none

9.2. Media Type Registration

This document requests registration of the application/woa+json media type in the "Media Types" registry according to [RFC6838].

The registration template is:

Type name: application
Subtype name: woa+json
Required parameters: none
Optional parameters: none
Encoding considerations: binary
  WoA documents are encoded as UTF-8 JSON text.
Security considerations: See Section 7 of this document.
Interoperability considerations: none
Published specification: This document.
Applications that use this media type:
  WoA hosts and clients that exchange WoA documents.
Fragment identifier considerations: none
Additional information: none
Person & email address to contact:
  Madhava Gaikwad <gaikwad.madhav@gmail.com>
Intended usage: COMMON
Restrictions on usage: none
Author: Madhava Gaikwad <gaikwad.madhav@gmail.com>
Change controller: IETF

9.3. WoA Transport Registry

This document requests the creation of a new registry titled "WoA Transport Registry". The registry records transport names used in the top level transports object of WoA documents.

Each entry in the registry has the following fields:

* Transport name: a short string identifying the transport.

* Reference: a stable specification that defines the transport.

Registration policy for the WoA Transport Registry is "Specification Required" as defined in [RFC8126].

This document registers the following initial entries:

Transport name: rest
Reference: This document

Transport name: mcp
Reference: Model Context Protocol specification [MCP]

10. 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/info/rfc2119>.

[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/info/rfc8174>.

[RFC8259] Bray, T., "The JavaScript Object Notation (JSON) Data Interchange Format", STD 90, RFC 8259, DOI 10.17487/RFC8259, December 2017, <https://www.rfc-editor.org/info/rfc8259>.

[RFC9110] Fielding, R., Nottingham, M., and J. Reschke, "HTTP Semantics", STD 97, RFC 9110, DOI 10.17487/RFC9110, June 2022, <https://www.rfc-editor.org/info/rfc9110>.

[RFC8446] Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", RFC 8446, DOI 10.17487/RFC8446, August 2018, <https://www.rfc-editor.org/info/rfc8446>.

[RFC8615] Nottingham, M., "Well-Known Uniform Resource Identifiers (URIs)", RFC 8615, DOI 10.17487/RFC8615, May 2019, <https://www.rfc-editor.org/info/rfc8615>.

[RFC6838] Freed, N., Klensin, J., and T. Hansen, "Media Type Specifications and Registration Procedures", BCP 13, RFC 6838, DOI 10.17487/RFC6838, January 2013, <https://www.rfc-editor.org/info/rfc6838>.

[RFC9457] Nottingham, M., Wilde, E., and S. Dalal, "Problem Details for HTTP APIs", RFC 9457, DOI 10.17487/RFC9457, July 2023, <https://www.rfc-editor.org/info/rfc9457>.

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

[RFC5234] Crocker, D. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", STD 68, RFC 5234, DOI 10.17487/RFC5234, January 2008, <https://www.rfc-editor.org/info/rfc5234>.

[RFC8126] Cotton, M., Leiba, B., and T. Narten, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 8126, DOI 10.17487/RFC8126, June 2017, <https://www.rfc-editor.org/info/rfc8126>.

[RFC1918] Rekhter, Y., Moskowitz, B., Karrenberg, D., de Groot, G., and E. Lear, "Address Allocation for Private Internets", BCP 5, RFC 1918, DOI 10.17487/RFC1918, February 1996, <https://www.rfc-editor.org/info/rfc1918>.

[JSON-SCHEMA-CORE] Wright, A., Andrews, H., Hutton, B., and G. Dennis, "JSON Schema: A Media Type for Describing JSON Documents", December 2020, <https://json-schema.org/draft/2020-12/json-schema-core>.

[JSON-SCHEMA-VAL] Wright, A., Andrews, H., and B. Hutton, "JSON Schema Validation: A Vocabulary for Structural Validation of JSON", December 2020, <https://json-schema.org/draft/2020-12/json-schema-validation>.

[I-D.gaikwad-south-authorization] Gaikwad, M., "SOUTH: Stochastic Authorization for Agent and Service Requests", Work in Progress, Internet-Draft, draft-gaikwad-south-authorization, October 2025, <https://datatracker.ietf.org/doc/draft-gaikwad-south-authorization/>.

11. Informative References

[CUI-AGENT-DISCOVERY] Cui, Y., Chao, Y., and C. Du, "HTTP-Based AI Agent Discovery and Invocation Protocol", Work in Progress, Internet-Draft, draft-cui-ai-agent-discovery-invocation, October 2025, <https://datatracker.ietf.org/doc/draft-cui-ai-agent-discovery-invocation/>.

[MCP] Anthropic, "Model Context Protocol Specification", November 2024, <https://modelcontextprotocol.io>.

Appendix A: Example Use with SOUTH (Non-Normative)

This appendix sketches how a deployment might combine WoA with the SOUTH stochastic authorization protocol [I-D.gaikwad-south-authorization]. It is non-normative and does not modify the core WoA format.

A host could add an authz object at the top level of the WoA document or within an agent object that points to a SOUTH policy endpoint. For example:

```
{
  "woa_version": "1",
  "agents": [
    {
      "id": "summarizer",
      "name": "Document Summarizer",
      "description": "Summarizes English text.",
      "version": "1.0.0",
      "capabilities": [
        "https://example.com/capability/summarization"
      ],
      "inputs": { "...": "JSON Schema elided" },
      "outputs": { "...": "JSON Schema elided" },
      "transports": ["rest"],
      "authz": {
        "scheme": "south",
        "policy_uri":
          "https://authz.example.com/south/policies/summarizer",
        "token_header": "Authorization"
      }
    }
  ],
  "transports": {
    "rest": {
      "base": "https://api.example.com",
      "invoke_path": "/agents/{agent_id}/invoke"
    }
  }
}
```

A client that understands SOUTH could first obtain a token from the SOUTH authorization server according to [I-D.gaikwad-south-authorization] and then include it as a SOUTH token in the HTTP request:

```
POST /agents/summarizer/invoke HTTP/1.1
Host: api.example.com
Content-Type: application/json
Authorization: SOUTH eyJhbGciOi...

{
  "agent": "summarizer",
  "operation": "default",
  "input": {
    "text": "Long document text here..."
  }
}
```

The details of how SOUTH evaluates policies and issues tokens are defined in [I-D.gaikwad-south-authorization] and are not repeated here.

Appendix B: Complete Example WoA Flow (Non-Normative)

This appendix provides a complete example showing a WoA document, a client fetching it, invoking an agent via the REST transport, and receiving a successful response.

Example WoA Document

Consider a host at https://api.example.com that serves the following WoA document at https://api.example.com/.well-known/woa.json using the application/woa+json media type.

```
{
  "woa_version": "1",
  "agents": [
    {
      "id": "summarizer",
      "name": "Document Summarizer",
      "description": "Summarizes English text.",
      "version": "1.0.0",
      "capabilities": [
        "https://example.com/capability/summarization"
      ],
      "inputs": {
        "$schema": "https://json-schema.org/draft/2020-12/schema",
        "type": "object",
        "properties": {
          "text": {
            "type": "string",
            "description": "Input document text in English."
          },
          "max_words": {
            "type": "integer",
            "minimum": 10,
            "maximum": 500,
            "description": "Maximum number of words in the summary."
          }
        },
        "required": ["text"]
      },
      "outputs": {
        "$schema": "https://json-schema.org/draft/2020-12/schema",
        "type": "object",
        "properties": {
```

```
            "summary": {
              "type": "string",
              "description": "The generated summary."
            }
          },
          "required": ["summary"]
        },
        "transports": ["rest"],
        "operations": [
          {
            "name": "default",
            "description": "Default summarization operation."
          }
        ]
      }
    ],
    "transports": {
      "rest": {
        "base": "https://api.example.com",
        "invoke_path": "/agents/{agent_id}/invoke"
      }
    }
  }
```

Client Fetches WoA Document

A client fetches the WoA document:

```
GET /.well-known/woa.json HTTP/1.1
Host: api.example.com
Accept: application/woa+json, application/json
```

The host responds:

```
HTTP/1.1 200 OK
Content-Type: application/woa+json
ETag: "abc123"
Cache-Control: max-age=300

{ ... JSON from previous section ... }
```

The client parses the JSON, locates the agent with id "summarizer", and notes that it supports the rest transport.

Client Invokes the Agent

The client constructs the invocation URL by concatenating the base and invoke_path values and substituting "{agent_id}" with "summarizer":

```
https://api.example.com/agents/summarizer/invoke
```

The client then builds an invocation envelope that conforms to the agent's inputs schema:

```
{
  "agent": "summarizer",
  "operation": "default",
  "input": {
    "text": "The IETF is an open community of designers.",
    "max_words": 40
  }
}
```

The client sends the HTTP request:

```
POST /agents/summarizer/invoke HTTP/1.1
Host: api.example.com
Content-Type: application/json
Accept: application/json

{
  "agent": "summarizer",
  "operation": "default",
  "input": {
    "text": "The IETF is an open community of designers.",
    "max_words": 40
  }
}
```

The host validates the request against the JSON Schema in inputs, performs the summarization, and returns a response that matches the outputs schema:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "summary": "The IETF is a community focused on Internet evolution."
}
```

The client MAY validate the response body against the outputs schema and then present the summary to the user or feed it into subsequent processing.

Author’s Address

Madhava Gaikwad
Email: gaikwad.madhav@gmail.com