Integrating Agentic Web Standards in Multi-Agent Systems Using Hypermedia

Jérémy Lemée[0000−0003−0828−1188]

Institute of Computer Science, University of St.Gallen, Switzerland
jeremy.lemee@unisg.ch

Abstract. The development of language agents, relying on Large Language Models (LLMs), is leading to the creation of new standards for multi-agent systems. Many such standards, including the Model Context Protocol (MCP), the Agent2Agent (A2A) Protocol, and the Universal Tool Calling Protocol (UTCP), enable agents to interact with Web-based systems, resulting in the creation of an Agentic Web. Hypermedia is a core feature of the Web that enables users to discover information at run time. Hypermedia standards, such as the Web of Things (WoT) Thing Description, allow agents to interact with Web services and physical devices in open Web environments. However, MCP and A2A do not natively define hypermedia controls, which introduces coupling between agents and their environments, limiting their independent evolution in open environments.

We develop a uniform hypermedia interface applicable to different Agentic Web standards and usable by language agents through tools provided using the agents' own supported standards. This interface enables agents to discover and interact with entities of different types. This interface relies on signifiers, which are hypermedia descriptions of affordances. We provide automatic generation of signifiers for MCP tools, A2A agents, UTCP tools, and WoT Things. We propose an LLM-based selection method for such signifiers so that agents perceive only contextually relevant signifiers. We expose each signifier perceived by an agent as an MCP tool that performs the action described by the signifier. Finally, we demonstrate and evaluate these concepts in a robotic lab environment.

Keywords: Hypermedia Multi-Agent Systems · Agentic Web · Signifiers · Web of Things · Model Context Protocol

1 Introduction

Language agents are agents that rely on Large Language Models (LLMs) to interact with the world [28]. They are increasingly interacting with Web services on behalf of human users, resulting in the development of an **Agentic Web**, designed for use by language agents in addition to human users [33,12,18]. One core protocol developed for the Agentic Web is the Model Context Protocol (MCP), which makes different applications and services, including Web services, available to language agents as tools [6,33]. Other protocols, such as the

Agent2Agent (A2A) protocol, enable communication between language agents on the Web [29,33]. However, while the Agentic Web develops Web environments for agents, it remains mostly disconnected from Multi-Agent Systems (MAS) research on such environments, such as hypermedia Multi-Agent Systems (hMAS) [17].

hMAS research considers that large-scale, open, interoperable, and dynamic Web-based MAS should be aligned with the architecture of the Web [17]. To do so, hMAS relies on hypermedia to decouple the development of agents from the development of their environments, enabling both to evolve independently [25,16]. Despite the potential of the hMAS vision for creating Web-based MAS, Fabien Gandon explains that this vision has yet to be applied at scale because there is not yet a critical use case requiring the development of large-scale hypermedia MAS [9,10]. Due to its widespread and increasing adoption, the Agentic Web could provide such a use case. As the scale and heterogeneity of the Agentic Web increase, hypermedia is likely to become necessary to enable language agents to discover and interact with Web services (including MCP-based services, A2A agents, etc.). Through a uniform hypermedia interface, language agents could achieve their goals by discovering information about how to interact with Web services, even if these services implement protocols that the agents are not familiar with. Designing a uniform hypermedia interface that can be applied over services implementing current, and potentially future, Agentic Web standards could thus enable their integration into a common Agentic Web and realize the hMAS vision.

Our research question is: "How can we build a general uniform hypermedia interface that provides language agents with contextually relevant information about affordances exposed by services implementing established and emerging Agentic Web standards?"

To address this question, we propose the definition of an extensible interface that agents can interact with using any standard or framework for tool use they natively implement, while enabling them to interact with Web services implementing standards for tool use or simple agent to agent interactions even when such standards are not directly usable by the agents. Our proposed interface relies on the Signifier Exposure Mechanism (SEM), which has been proposed to enable agents to perceive only contextually relevant hypermedia information, represented as signifiers, about the actions they can perform in the environment (e.g., interacting with a Web service) [30,24]. A signifier is contextually relevant for an agent if the action possibility it describes can actually be performed by that agent and contributes to achieving the agent's goal [30,24]. We apply the SEM to services that implement different standards for the Agentic Web. To do so, we present a model for signifiers that is applicable to these different types of services. Our model includes both formal hypermedia controls and natural language metadata that can be used by language agents. Using signifiers based on this model, the SEM provides a hypermedia interface for agents to interact with Agentic Web technologies, therefore bridging the gap between the Agentic Web and hMAS. The selection mechanism of our proposed SEM relies on

LLMs to determine whether a signifier is contextually relevant to an agent by processing the context, described in natural language, provided by the agent and the recommended context, also described in natural language, provided by the signifier. Tool are created from selected signifiers and exposed using standards the agents are capable of using. These tools enable agents to interact with the services described by the signifiers.

In Section 2, we present related work on the recent development of language agents and standards that enable them to interact with Web environments, as well as corresponding research in the MAS community to enable agents to perceive contextually relevant information in their environment. In Section 3, we present our model for signifiers for language agents and show how it can be applied to different protocols and standards for the Agentic Web. We integrate LLMs within the Signifier Exposure Mechanism (SEM) to present contextually relevant information to language agents, and we explain how the information provided by signifiers can be made actionable by language agents using tools. In Section 4, we present our implementation, scenario, and evaluation of our proposed SEM for an agent that controls a robot and needs to interact with services implementing different Agentic Web standards to achieve its goal. We discuss their implications for the development of the Agentic Web as well as their limitations in Section 5. Finally, we present our conclusions regarding the implications of the Agentic Web for MAS in Section 6.

2 Background and Related Work

Research concerning language agents on the Web and research concerning Multi-Agent Systems (MAS) have both been dealing with common issues about enabling agents to discover at run time the information they need to interact with their environments. In Section 2.1, we present the current developments of standards for the Agentic Web; then in Section 2.2, we present how hypermedia Multi-Agent Systems (hMAS) enable agents in Web environments to perceive contextualized information about how to interact with their environments.

2.1 Language Agents in Web-based Environments

Large Language Models (LLMs) are used to design *language agents*, capable of interaction with and reasoning about their environments in natural language [28]. Language agents need tools to perceive and act in their environments and to perform computations [26,32]. Although tools have been implemented directly as part of agent frameworks (e.g., Langchain [1]), framework-independent tools are usable across agent frameworks, creating a common ecosystem of tools for agents. Standards are being developed to enable heterogeneous agents to discover and interact with tools of any type.

Model Context Protocol (MCP) is a protocol to bring external tools and resources into LLM-based applications (e.g., chatbots, agents) [6]. MCP is becoming a

core technology of the Agentic Web to connect agents with Web services [33,18]. A core strength of MCP is that it enables a separation of concerns between designers of language agents and designers of tools because any tool available through an MCP server can be used by any LLM application implementing an MCP client. An MCP client can interact with an MCP server on the Web using the JSON-RPC 2.0 protocol [1] and the streamable HTTP transport.

Universal Tool Calling Protocol (UTCP) is another protocol to enable agents to interact with tools [3]. Unlike MCP that requires each tool to be provided through a server implementing the MCP protocol, UTCP provides manuals to allow agents to directly interact with native APIs for different protocols (e.g., HTTP APIs, gRPC). A manual lists the tools available and provides all the information that an agent needs to interact with them, including the requests (e.g., HTTP requests) to perform to interact with the tool, and natural language descriptions of the tool and its parameters.

Agent2Agent (A2A) is a protocol that allows users to send messages to agents that perform tasks on their behalf and return responses [29]. The A2A protocol uses agent cards to identify and describe agents, including their skills, so that a user knows how and for what purpose to initiate communication with an agent.

Web of Things (WoT) Thing Description (TD) is a standard for the WoT that defines how to interact with a given Thing through interaction affordances [23]. This standard relies on Semantic Web technologies, such as JSON-LD and the WoT TD Ontology [14], which enable extensibility to new use cases and support for a variety of protocols (e.g., HTTP, CoAP) [23]. The Thing Description standard predates language agents, but it has been used in the context of the Agentic Web. In particular, the Eclipse Language Model Operating System (LMOS) uses TDs to describe language agents and their tools [4], and thingctx uses TDs to describe tools and Web services for language agents [5].

Limitations of the Agentic Web Standards These different standards enable language agents to interact with external systems, often using the HTTP protocol. However, MCP and A2A do not natively support hypermedia-driven interaction and thus rely on out-of-band information to enable interactions between agents and tools or other agents. Additionally, the information that these standards provide to language agents is not adapted to the agents' goals. As a result, agents may perceive all available information (e.g., all tools described by a UTCP manual) instead of the ones that are actually relevant to the agent.

2.2 Providing Contextualized Information to Agents Using Hypermedia

On the Web, hypermedia associates information about the actions that clients can perform with the action controls [19]. Hypermedia controls presented to

[1] https://modelcontextprotocol.io/docs/learn/architecture. Accessed: 19/01/2026

agents are often contextualized to agents. For example, search engines present hyperlinks (a type of hypermedia controls) that are relevant to a user query [11]. The HATEOAS (Hypermedia As The Engine Of Application State) constraint ensures, when applied, that agents perceive the hypermedia controls they need to change the application state (e.g., moving to the next page of an ordered website) [19].

Hypermedia Multi-Agent Systems (hMAS) are Web-based MAS where hypermedia information and controls enable agents to discover, at run time, and use the resources present in the environment [17]. Hypermedia decouples agents from their environment, allowing both agents and environments to be designed independently [16], and enabling the integration of different information management systems [8,7]. A hypermedia interface exposing contextualized information for agents to let them interact with different types of services would thus help realize the Agentic Web as an integrated environment for agents. Both *hMAS* and *Agentic Web* are thus complementary approaches towards developing Web-based MAS [22].

Signifiers for software agents on the Web In User-Centered Design and Human-Computer Interaction, the concept of signifier refers to perceptual information about what an object (physical or virtual) can do and how to use it [27,24]. The hMAS ontology[2] relies on hypermedia signifiers to create hypermedia semantic representations of an hMAS. The Signifier Exposure Mechanism (SEM) relies on the profiles of the agents and the resource profiles present in the environment [30,24] to expose contextually relevant signifiers for the agents. Although so far the development of agents using signifiers has mainly focused on BDI agents [31], signifiers can be used by heterogeneous types of agents [30], including language agents. Besides being usable by heterogeneous agents, hypermedia signifiers can also be designed for any Web service, including ones that implement the standards presented in Section 2.1.

3 Enabling Language Agents to Perceive and Use Contextual Hypermedia Information about Agentic Web Environments

We define a uniform model for resource profiles that expose signifiers that language agents can use. This model is meant to be general enough to be applicable to resources that are implemented using different standards, potentially including future standards. In Section 3.1, we explain our ontological model for signifiers, and then we focus on creating such signifiers for MCP servers, UTCP tools, A2A agents, and WoT Things (presented in Section 2.1). In Section 3.2, we present our extension of the Signifier Exposure Mechanism, described in [30], to support contextual exposure of signifiers for language agents, and we explain how the

[2] https://purl.org/hmas/. Accessed February 14, 2026.

affordances described by signifiers can be used by language agents, through the presentation of tools for each perceived signifier.

3.1 Resource Profiles and Signifiers for the Agentic Web

We propose to automatically create resource profiles described using the hMAS ontology from descriptions provided by existing standards (MCP tool descriptions, UTCP manuals, A2A Agent Cards, and WoT Thing Descriptions).

An Ontological Model for Signifiers for Language Agents We present an ontological model for signifiers as an extension of the model presented in [30,31] (see Section 2.2) and the corresponding hMAS ontology. The ontological model also relies on the concept of *interaction affordance*, associated with the Thing Description model (see Section 2.1) to provide a general hypermedia representation of affordances to interact with different HTTP-based Agentic Web standards. Natural language descriptions integrated within signifiers enable language agents to understand the meaning of the described affordances. The hMAS ontology and other ontologies, listed in Listing 1 (all prefixes for the ontologies that we use in the article are defined there), are used to formalize the proposed model for signifiers. This model is developed to let agents discover how to use tools defined using different standards and how to send messages to other agents. The model identifies each signifier with a unique *label*, introduced with the property *rdfs:label*. The purpose of the label is to provide a name that identifies the action (e.g., the tool that the agent calls) conveyed by the signifier for the language agent using that signifier. The *recommended context* is introduced with the property *hmas:recommendsContext*. It is given a natural language description using the *rdfs:comment* property. This description is used by language agents to decide whether to use the action possibility described by the signifier. The signified action possibility, introduced with *hmas:signifies*, is an interaction affordance, as defined by the Thing Description (TD) (see Section 2.1). The HTTP protocol is common to the different standards that we consider. Therefore, we focus on affordances that can be interacted with using this protocol. However, the approach is generalizable to other protocols through protocol bindings [23]. The interaction affordance can include a JSON schema, which we call *schema*, introduced with the property *td:hasInputSchema*. This schema can be used to define the tool parameters when calling a tool for a tool-based standard (MCP, UTCP), or a message to send for A2A. If a JSON schema, describing a JSON body of an HTTP request, has parameters whose values are not known and need to be determined by the agent, these parameters can be given a description, introduced with the property *js:description*. The interaction affordance also provides hypermedia controls through a *form*, introduced with the property *td:hasForm*. The form defines the *content type* and the *target* URL. The form can also define the HTTP *headers* using *http:headers* and the *method* using *http:methodName*. Finally, even if not used in this article, the model supports the optional recommended *abilities* that an agent needs to use a signifier

(hmas:recommendsAbility) from [30]. We use the Shapes Constraint Language (SHACL) [21] to provide a formal representation of the shape we propose for signifiers in Listing 1. This shape is meant to be general enough to be applicable to several protocols and standards for the Agentic Web supporting HTTP-based interactions. We construct signifiers using this ontological model for some of these protocols and standards (MCP, A2A, UTCP, WoT; see Section 2.1).

```
@prefix sh:   <http://www.w3.org/ns/shacl#> .
@prefix xsd:  <http://www.w3.org/2001/XMLSchema#> .
@prefix ex:   <http://example.com/#> .
@prefix hmas: <https://purl.org/hmas/>.
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>.
@prefix hctl: <https://www.w3.org/2019/wot/hypermedia#>.
@prefix http: <http://www.w3.org/2011/http#>.
@prefix td: <https://www.w3.org/2019/wot/td#>.
@prefix js: <https://www.w3.org/2019/wot/json-schema#>.
ex:SignifierShape a sh:NodeShape ; sh:targetClass hmas:Signifier ;
  sh:property [ sh:path rdfs:label ;  sh:datatype xsd:string ;
  sh:minCount 1 ; sh:maxCount 1;] ;
    sh:property [ sh:path hmas:recommendsAbility] ;
    sh:property [ sh:path hmas:recommendsContext ; sh:minCount 1 ; sh:maxCount 1;
        sh:property [ sh:path rdfs:comment ; sh:minCount 1 ;  sh:maxCount 1; ] ; ] ;
    sh:property [  sh:path hmas:signifies ; sh:minCount 1 ; sh:maxCount 1;
        sh:node ex:AffordanceShape ].

ex:AffordanceShape a sh:NodeShape; sh:targetClass td:InteractionAffordance;
        sh:property [ sh:path td:hasForm ; sh:minCount 1; sh:maxCount 1;
        sh:node ex:FormShape] ;
    sh:property [ sh:path td:hasInputSchema ; sh:minCount 0 ; sh:maxCount 1; ].

ex:FormShape a sh:NodeShape;
sh:property [ sh:path http:methodName ; sh:minCount 1 ; sh:maxCount 1; ] ;
            sh:property [ sh:path hctl:hasTarget ; sh:minCount 1 ; sh:maxCount 1; ] ;
            sh:property [
                sh:path hctl:forContentType ; sh:minCount 0 ; sh:maxCount 1; ] ;
            sh:property [ sh:path http:headers ; sh:minCount 0 ; sh:maxCount 1; ] .
```

Listing 1: General model of a signifier

Signifiers for MCP A server implementing the Model Context Protocol (MCP) provides both tools and resources. Each tool or resource is described using JSON. The JSON description of a tool contains a *name*, a natural language *description* to help agents decide whether to use the tool, and an *inputSchema*, which is a JSON schema that describes the parameters of the tool and potentially includes a natural language description of each parameter. See Listing 4 in Appendix A for an example of a tool description. We explain how to automatically create signifiers to indicate to agents how to use a given MCP tool. This creation uses both the JSON description of the tool and information concerning the MCP protocol. We assume that the streamable HTTP transport is used to interact with the MCP server. The signifier constructed for a specific tool contains a *label*, which has the following form: {server_name}_{name}, where {server_name} is a name identifying the MCP server, and {name} is the name of a tool on that

server. The signifier's *recommended context* is the natural language description provided by the tool. The signifier's *form* includes a *header* called *Mcp-Session-Id*. When the signifier is created, the token of this header is generated by the MCP server. The *target* URL used for interaction is `{SERVER_BASE_URL}/mcp`, where {SERVER_BASE_URL} is the main URL of the server implementing MCP with the streamable HTTP transport. The HTTP *method* indicated in the signifier is "POST", and the *content type* is "application/json" for sending the JSON-RPC 2.0 payload. The *schema* relies on JSON-RPC 2.0 and adds elements specific to MCP, including the JSON-RPC method "tools/call". The parameters used to call the tool are defined by the *inputSchema* of the tool description. See Listing 2 for a signifier corresponding to the tool described in Listing 4. The construction of a signifier to read an MCP resource is similar (see Listing 8 in Appendix B for an example).

```

<#sig> a hmas:Signifier ;
    rdfs:label "goal_mcp_provide_feedback" ;
    hmas:recommendsContext [ rdfs:comment "Provide feedback for the current goal. If achieved is true, the current goal is cleared." ] ;
    hmas:signifies [ td:hasForm [ http:headers ( [ http:fieldName "Accept" ;
                                  http:fieldValue "application/json, text/event-stream" ] [ http:fieldName "MCP-Protocol-Version" ;
                                  http:fieldValue "2025-06-18" ] [ http:fieldName "Mcp-Session-Id" ;
                                  http:fieldValue "b4f1b002aac149869708507a1168ae1f" ] ) ;
                     http:methodName "POST" ; hctl:forContentType "application/json" ;
                     hctl:hasTarget <http://localhost:9996/mcp> ] ;
             td:hasInputSchema [ a js:ObjectSchema ;
                     js:properties [ a js:IntegerSchema ;js:propertyName "id" ],
                         [ a js:StringSchema ; js:const "2.0" ;js:propertyName "jsonrpc" ],
                         [ a js:StringSchema ;
                             js:const "tools/call" ;
                             js:propertyName "method" ],
                         [ a js:ObjectSchema ;
                             js:properties [ a js:ObjectSchema ;
                                     rdfs:comment "Provide feedback for the current goal." ;
                                     js:properties [ a js:StringSchema ;
                                             js:description "The feedback to provide" ;
                                             js:propertyName "feedback" ],
                                         [ a js:BooleanSchema ;
                                             js:description "Whether the goal has been achieved" ;
                                             js:propertyName "achieved" ] ;
                                     js:propertyName "arguments" ;
                                     js:required "achieved",
                                         "feedback" ],
                                 [ a js:StringSchema ;
                                     js:const "provide_feedback" ;
                                     js:propertyName "name" ] ;
                             js:propertyName "params" ;
                             js:required "arguments",
                                 "name" ] ;
                     js:required "id", "jsonrpc", "method", "params" ] ] .
```

Listing 2: Signifier for MCP tool

Signifiers for UTCP The Universal Tool Calling Protocol (UTCP) describes Web services for language agents using manuals [3]. A manual describes the functionalities of a Web service as tools that an agent can use. The UTCP manual provides all the information the agent needs to interact with the tools described by the manual, including a natural language description that indicates to language agents the meaning of the tool, and a formal description that is used on the client to interact with the tool. In Listing 6, we present a UTCP tool as described by a UTCP manual.

Using our formalism for signifiers, we can automatically create signifiers for each tool that uses the HTTP transport and provides its tool parameters in the request body. Each signifier has a *label*: {manual_name}_{tool_name}, where {manual_name} is a name that identifies the manual, while {tool_name} is the name of the tool in the UTCP manual. The *recommended context* of the tool is the *description* of the tool in the UTCP manual. The *form* is constructed from the HTTP *method*, the *content type*, and the *target* URL present in the *tool_call_template* of the tool in the manual. The *schema* of the signifier is constructed from the JSON schema *inputs* of the UTCP tool description. See Listing 10 for a signifier corresponding to the tool described in Listing 6.

Signifiers for A2A A2A agents provide information about themselves through a JSON agent card. This agent card is provided at the URL: `{BASE_URL}/.well-known/agent-card.json`, where {BASE_URL} is the base URL of the server running the A2A agent. We give a partial example of an agent card in Listing 5. An agent card describes the agent's skills. A skill includes a *description*, which indicates whether the A2A agent can perform a certain task, and input modes, which are the types of information the agent is able to process (e.g., text for textual media types, or data for JSON objects). A2A agents expose an HTTP interface that relies on HTTP+JSON-RPC 2.0, gRPC, or HTTP+JSON/REST transports. We consider the JSON-RPC 2.0 implementation, a common implementation of the protocol. The A2A protocol offers many potential affordances to users. In this article, we design signifiers for sending a message to an A2A agent and receiving a response from that agent. Other features that require long-term interactions between the agents, such as tasks, are not supported by our current model for signifiers.

From the proposed agent card, we create signifiers for each skill. The *label* given to the created signifier for an A2A agent is {agent_name}_{skill_id}, where {agent_name} is a name associated with the agent, and {skill_id} is the identifier of the skill. The *recommended context* of the signifier is the natural language *description* of the skill in the agent card. The signified affordance is a JSON-RPC 2.0 message transported over HTTP. The HTTP *method* is "POST", the *target* is the URL of the A2A server, the *content type* is "application/json" and the JSON Schema describes a JSON-RPC payload with the JSON-RPC method being "message/send". The "parts" parameter contains the various elements of the message. This parameter is a JSON array, with a corresponding JSON schema. Its description is constructed from the input modes of the skill

and it is represented in natural language (English in our implementation). These input modes indicate the types of data that the A2A agent skill can process. See Listing 9 for a signifier corresponding to the A2A agent described in Listing 5.

Signifiers for WoT Things WoT Things have Thing Descriptions (TDs) that indicate the interaction affordances (property, action, or event affordances) that they provided. We create signifiers for affordances described by TDs. See Listing 7 for an example of an action affordance in a TD. We rely on the Thing Description Ontology [15] (prefix: *td*) to create these signifiers. The *label* of the signifier is {thing_name}_{affordance_name}, where {thing_name} is a name associated with the Thing, while {affordance_name} is the *td:name* of the affordance. The *recommended context* of the signifier is the *td:description* of the affordance. If this natural language description of the context is not present, no signifier for the affordance is created in the resource profile of the Thing because this description is a required component in our model. The signified action possibility of the signifier in our model is of type *td:InteractionAffordance*, so it corresponds directly to the description of the affordance in the TD, and it contains a *form* that defines the method, target URL, and content type to use. Our proposed ontological model requires that only one form is defined in the signifier. If many forms are available in the description of the affordance in the TD, only one is integrated within the generated signifier to ensure that the described action can be unambiguously resolved as a tool call for the agent. The affordance may also contain an *input* schema, using *js:description* to provide natural language descriptions for the parameters whose values need to be set by the agent using the signifier. See Listing 11 for a signifier corresponding to the TD interaction affordance described in Listing 7.

3.2 Exposing Signifiers to Language Agents

We present how language agents can perceive signifiers that are relevant to their goals and how these agents can use tools created from information provided by signifiers.

Selection of Signifiers Using Natural Language Context The Signifier Exposure Mechanism (SEM) enables agents to perceive contextual information about affordances provided through hypermedia signifiers [30,24]. We enhance the SEM selection mechanism using an LLM-based assessment of contextual relevance. To do so, we create a function that computes a number between 0 and 1 estimating how relevant the signifier is given the agent profile. This function relies on an LLM call, with the LLM temperature set to 0 to reduce the variability of the results and thus increase their reproducibility. We construct a parametric prompt, presented in Listing 3 using Python, which lets the LLM determine whether the signifier's *recommended context* matches the natural language description of the agent's *context* in the agent profile. Our prompt lets the LLM

choose between 5 potential output values (0, 0.25, 0.5, 0.75, 1), expressing different relevance levels, from least to most relevant, of the signifier for the agent. An agent can indicate in its profile the minimum estimated relevance value it considers admissible (if not provided, the SEM uses 0.5) and the maximum number of signifiers it wishes to perceive (if not provided, the SEM returns all signifiers whose estimated relevance value is above the threshold). The SEM selects the signifiers that satisfy the agent's constraints while prioritizing signifiers with the highest estimated relevance. This evaluation of contextual relevance can be combined with other selection mechanisms (e.g., ability matching) to determine whether to expose a given signifier to an agent.

```
prompt = (
         "Determine whether a signifier defines an affordance that could be used by an agent to achieve its stated goal, according to its context.\n"
        "To do so, use the information from the profile context and the signifier context.\n"
        "The answer should be 0 if the signifier is not relevant at all to the agent goal.\n"
        "The answer should be 0.25 if the signifier is unlikely to be relevant to the agent goal but might be.\n"
        "The answer should be 0.5 if you do not know whether the signifier is relevant to the agent or not.\n"
        "The answer should be 0.75 if the signifier is likely to be relevant to the agent.\n"
        "The answer should be 1 if the signifier is extremely likely or definitely relevant to the agent.\n"
        "Respond with exactly one number among [0, 0.25, 0.5, 0.75, 1].\n"
        f"Profile context: {profile_context}\n"
        f"Signifier context: {signifier_context}"
    )
```

Listing 3: Prompt used by the SEM

Language Agents Using Signifiers as Tools Using the SEM, a language agent can perceive signifiers that are relevant to its goal. However, the agent must also be able to use the action possibilities described by the signifiers it perceives. To do so, we make the information provided by signifiers actionable by the agent through automatically generated tools. These tools could be implemented using many different frameworks (e.g., Langchain [1]) or exposed using different standards or protocols (e.g., MCP tools, UTCP tools). To make the proposed solution independent of the agent framework used to develop the agent, we propose to expose tools using known protocols or standards. We explain the specific process to create an MCP tool from a signifier. In our proposed mapping, the *label* of the signifier is used as the *name* of the created MCP tool, the *context* of the signifier is used as the *description* of the MCP tool, and the *schema* determines the parameters of the tool. Each non-constant field of the *schema* is used as a tool parameter. If one such field has a description, introduced with *js:description*, this description is associated with the tool parameter. The

execution of the tool corresponding to a signifier performs the HTTP request described by the signifier by setting the *method*, the *target*, the *content type*, the other *headers*, and the *body*, created from the *schema* and the parameters provided during the execution of the tool.

This mapping enables language agents with native support for MCP to use the information provided by signifiers even if they cannot directly use the other standards (e.g., A2A, UTCP, WoT TD) for which signifiers are created. Our mapping for MCP only depends on being able to create a tool capable of executing the request described by the signifier and exposing the relevant information from the signifier in its description. Therefore, similar mappings can be created for other standards that provide both a description of the tools and a formal description of their parameters using a JSON schema with associated natural language descriptions for the parameters. In particular, we can define a similar mapping for describing tools created from signifiers with an UTCP manual or a Thing Description. If an agent is natively able to use a certain standard and it has a corresponding interface for signifiers using that standard, it is able to use any signifier as a tool. A drawback of this approach is that the language agent may not be able to use all the features of the original standard from which a signifier is constructed. For example, an agent could only use the A2A send message functionality and not the management of A2A tasks because this feature is not currently supported by signifiers for A2A agents.

A language agent can use a special tool to query signifiers. The execution of this tool creates new tools from the retrieved signifiers. Finally, the agent interacts with the new tools to use the corresponding Web services. Figure 1 illustrates this workflow.

4 Implementation and Evaluation

We describe our implementation of the model proposed for signifiers and the Signifier Exposure Mechanism (SEM) in Section 4.1. Then, we describe the scenario we use to evaluate the SEM in Section 4.2, and present the evaluation itself in Section 4.3.

4.1 Implementation of the Signifier Exposure Mechanism for Language Agents

We implement the Signifier Exposure Mechanism (SEM) platform as a Python Flask application[3]. If `profile` is the URL of the agent profile, then an HTTP `GET` request to `/signifiers?profile={profile}` returns the signifiers that are contextually relevant to the agent with this profile. We define functions to register MCP servers, A2A agents, UTCP manuals, and WoT TDs. These functions accept two parameters: an identifier for the service (called `{artifact_id}`) and an entry point URL for retrieving relevant information about the service. They

[3] Our code is available here: https://github.com/JeremyLemee/agentic-web-hmas

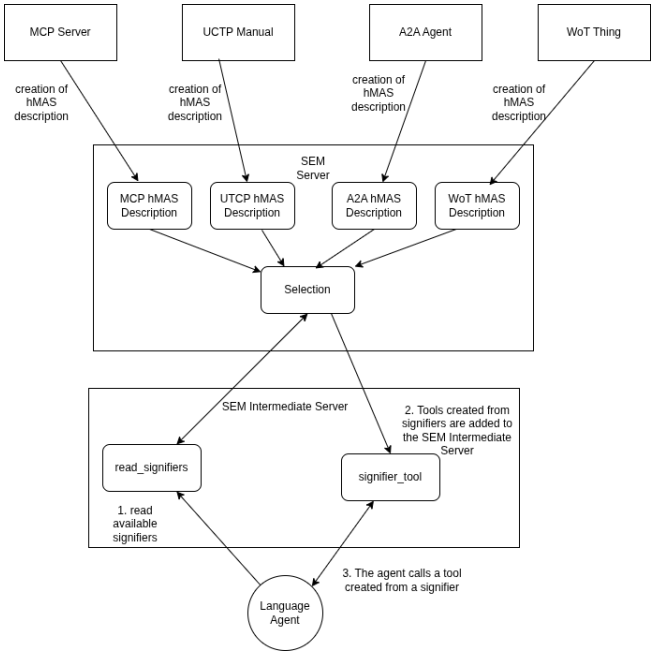

Fig. 1. Workflow of a language agent querying and using signifiers

retrieve information about the instance (e.g., an MCP list of tools) and construct the resource profile for the instance (see Section 3.1), where the identifier of the artifact is used to identify it in the label created for each signifier created for the artifact. The endpoint `/artifacts/{artifact_id}` exposes the resource profile of an artifact registered with the given `{artifact_id}`. To implement the LLM-based selection mechanism for signifiers (see Section 3.2), the SEM relies on an LLM backend that supports models from the OpenAI API and models from a local Ollama server [2].

We implement an MCP server to allow language agents to interact with the SEM platform. The server provides tools to query signifiers, to register a profile, to update its natural language context, to set the maximum number of signifiers, and to set the minimum expected relevance threshold for signifiers.

We use the Langchain framework to implement the language agents used in our scenario [1].

4.2 Scenario

Our proposed system enables a language agent to interact with Web services relying on different standards and protocols with which the agent is not natively able to use. We define a scenario that demonstrates an agent using a Web-enabled robotic arm and other Web services. Our full scenario is presented in Figure 2.

Scenario Components We use a laboratory environment that contains a robotic arm and we provide a proxy that handles interaction with the robotic arm through the proxy's own acquired credentials. The proxy executes operations corresponding to a formal goal (e.g., "move d", where d is the distance in cm, negative for backward movement, or "rotate a", where a is the angle of the rotation). The proxy exposes both a TD and a UTCP manual and either one, but not both, is registered in the SEM platform.

The Goal MCP server exposes a resource that provides the user goal in natural language and a tool for the agent to provide feedback to the user.

The Formalizer Agent is an A2A language agent that can return a formal description of the goal from an informal description. This agent uses private tools to compute the value of the operation with the correct units.

The Executor Agent is an agent dedicated to achieving the user goal. It can only interact directly with the SEM MCP server presented in Section 4.1.

Scenario Workflow At the start of the scenario, the Executor Agent registers its profile in the SEM profile registry. The agent's goal is to achieve the user goal, but the agent does not initially know the user goal. The agent writes in its profile that it is looking for the user goal. Then, the agent reads signifiers corresponding to its profile, finds the tool to read the goal, and retrieves the goal (e.g., "Move the robot by 10 centimeters"). The agent updates its profile to indicate that it wants to provide feedback to the user and reads the new signifiers available. Using the created tool suitable for the task, the agent provides feedback to the user stating that the agent has started working on the goal.

To achieve the goal, the agent first needs a formal representation of the goal that it can use to control the robot. To do so, the agent indicates in its profile that it wants a formal representation of the goal to control the robot. The agent reads the new signifiers and discovers a tool to interact with the Formalizer Agent. The agent uses this tool to get the formalized version of the goal (e.g., "move(10)").

The Executor Agent indicates in its profile that it wants to perform a formal operation on the robot. The agent queries signifiers and a new tool is created, from a perceived signifier, to control the robot. The agent uses this tool by providing the formal goal as input. The Executor Agent then provides feedback to the user, following the process previously described, to indicate that the goal has been achieved.

4.3 Evaluation

We evaluate the proposed concepts and their implementation using the scenario defined in Section 4.2. The scenario can only be run successfully if the Executor Agent perceives the signifiers it needs to perform the different subtasks of the scenario. We first consider the evaluation of the LLM-based selection mechanism of signifiers by measuring the precision and recall for the signifiers selected by the SEM given the agent profile, which vary for each subtask of the scenario, and for

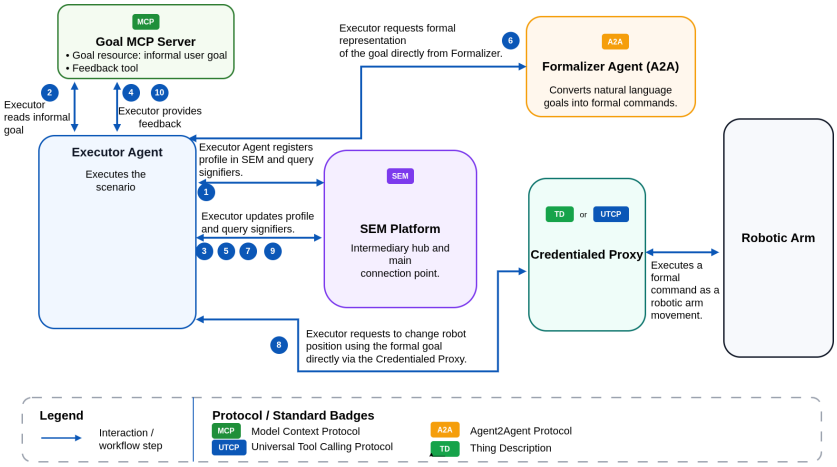

Fig. 2. Scenario

the underlying LLM being used. To perform this evaluation, we use local open-source models: *gemma3:1b* (0), *ministral-3:8b* (1), *ministral-3:14b* (2) through Ollama. We also use *gpt-4.1-mini* (3) and *gpt-5.4-mini* (with no reasoning) (4), and *gpt-5.4-mini* (with medium reasoning) (5) through the OpenAI API. For each task of perceiving signifiers present in the scenario, we define an agent profile that will be used in the signifier selection process and the expected signifiers. The agent profile contains the agent's possible goals. These goals are described for each task in the Appendix C. Although task "Goal/FB" does not appear in the scenario, we introduce it to evaluate the SEM on a task where many tools are relevant, unlike the other tasks considered. We also introduce different potential values for the maximum number of signifiers (1, 10) or the minimum value for acceptable signifiers (0.75, 1). We measure the precision (P) and recall (R) of each task for each LLM, with a temperature of 0 to obtain close to deterministic results for each model/task, and we run the models 25 times for each task. We present the results in Table 1, Table 2, and Table 3 for different values of maximum signifiers and minimum estimated relevance value. In each table, the numbers introduced for the different LLMs correspond to the ones previously introduced.

Task/LLM	LLM 0	LLM 1	LLM 2	LLM 3	LLM 4	LLM 5
Goal	P:0,R:0	P:1,R:1	P:1,R:1	P:0,R:0	P:0.92,R:0.92	P:1,R:1
Feedback	P:1,R:1	P:1 R:1	P:1,R:1	P:1,R:1	P:0.96,R:096	P:0.96,R:0.96
Formalizer	P:0,R:0	P:0,R:0	P:1,R:1	P:1,R:1	P:0,R:0	P:0.44,R:0.44
Goal/FB	P:0,R:0	P:1,R:0.50	P:1,R:0.50	P:0.24,R:0.12	P:1,R:0.50	P:1,R:0.50
Robot	P:1,R:1	P:0,R:0	P:1,R:1	P:0,R:0	P:0.12,R:0.12	P:0.32,R:0.32

Table 1. Precision and recall for different LLMs with maximum signifiers=1 and minimum estimated relevance value=0.75

Task/LLM	LLM 0	LLM 1	LLM 2	LLM 3	LLM 4	LLM 5
Goal	P:0.33,R:1	P:0.33,R:1	P:1,R:1	P:0.33,R:1	P:0.48,R:1	P:0.90,R:1
Feedback	P:1,R:1	P:1,R:1	P:1,R:1	P:0.35,R:1	P:0.25,R:1	P:0.50,R:1
Formalizer	P:0.50,R:1	P:0.33,R:1	P:1,R:1	P:0.29,R:1	P:0.25,R:1	P:0.31,R:1
Goal/FB	P:0.50,R:0.50	P:1,R:1	P:1,R:1	P:0.67,R:1	P:0.53,R:1	P:0.83,R:1
Robot	P:0.50,R:1	P:0.33,R:1	P:0.50,R:1	P:0.20,R:1	P:0.33,R:1	P:0.33,R:1

Table 2. Precision and recall for different LLMs with maximum signifiers=10 and minimum estimated relevance value=0.75

Task/LLM	LLM 0	LLM 1	LLM 2	LLM 3	LLM 4	LLM 5
Goal	P:0,R:0	P:1,R:1	P:1,R:1	P:0.50,R:1	P:0.96,R:1	P:1,R:1
Feedback	P:0,R:0	P:0.96,R:0.96	P:1,R:1	P:1,R:1	P:0.98,R:1	P:0.98,R:1
Formalizer	P:0,R:0	P:0,R:0	P:1,R:1	P:0.50,R:1	P:0.33,R:1	P:0.43,R:1
Goal/FB	P:0,R:0	P:1,R:0.50	P:1,R:1	P:0.75,R:1	P:1,R:1	P:1,R:1
Robot	P:0,R:0	P:0,R:0	P:0.50,R:1	P:0.33,R:1	P:0.35,R:1	P:0.39,R:1

Table 3. Precision and recall for different LLMs with maximum signifiers=10 and minimum estimated relevance value=1

We use the *gpt-4.1-mini* LLM to run both the Executor Agent and the Formalizer Agent. We evaluate whether the Executor Agent can perform the task successfully if the interface to the robotic arm proxy is registered from either a Thing Description or a UTCP manual. Using the LLM *ministral-3:14b* for the SEM, and a simulation of the robotic arm for repeated tests, we launch agents to achieve the goal "Rotate the robot by 12 degrees" 15 times for the UTCP interface and we launch agents to achieve the goal "Move the robot by 10 centimeters" 15 times for the WoT TD interface to the robot simulation proxy. The agent achieves the goal successfully for all iterations.

5 Discussion

Our evaluation demonstrates that signifiers enable agents to use Web services with which they cannot interoperate by default. Specifically, in the defined scenario, agents can accomplish their tasks by perceiving only the signifiers relevant to their goals. Our approach allows agents to adapt their perception of signifiers by controlling both the number of signifiers perceived and the minimum estimated relevance threshold. This selective perception of signifiers is expected to scale better in open Web environments with many signifiers.

Comparison of LLMs used for selecting signifiers Our evaluation shows that different LLMs estimate differently whether a signifier is relevant to an agent with a given profile. With medium reasoning, *gpt-5.4-mini* has a strictly positive recall in Table 1 while the same model with no reasoning has a recall of 0.

In addition, for a recall of 1 in Table 2, the model with reasoning has higher precision than the same model with no reasoning. However, the time taken by the model with medium reasoning is more than double compared when using the same model with no reasoning (e.g., 6.39s vs 2.75s for the Goal task). In Table 1, *ministral-3:14b* has a precision of 1 for all tasks, and a recall as high as possible for a maximum number of signifiers of 1. Therefore, setting a low value for maximum perceived signifiers and a high value for minimum estimated relevance value can lead agents to being able to retrieve the signifiers they need with maximum precision when the LLM used allows it. In the case of In Table 2, where the maximum number of signifiers (10) is higher than the total number of available signifiers (5) and the relevance threshold is 0.75, all models have a recall of 1 on all tasks, except for *gemma3:1b*, which has a recall of 0.50 on the Goal/FB task, and 1 for the other tasks. In Table 3, where the relevance threshold is changed to 1, the recall remains 1 for all tasks for the ministral models and the OpenAI models, and the precision on all tasks for these models is higher than or equal to the precision obtained when the relevance threshold is 0.75. This shows that weaker models benefit more from lowering the estimated relevance threshold, while stronger models benefit from a higher threshold. In general, the agent should select a lower relevance threshold when the selection does not succeed with a higher value or when the agent can afford to perceive all signifiers. The agent should select a high maximum for signifiers when the model is not good enough to have a strictly positive recall with a lower maximum number of signifiers. The agent should select a high maximum for signifiers when many signifiers are relevant to the agent's goal.

Limitations of the proposed approach While our model for signifiers enables agents to interact with services implementing many different Agentic Web standards, our signifiers cannot describe multi-step processes. For this reason, our proposed model for MCP signifiers needs to hardcode the Mcp-Session-Id token, making the current SEM only usable by mutually trusted agents because they share the same MCP session. A potential solution, better suited for open environments, would be to regenerate signifiers for each agent. More generally, we propose, as future work, to extend our model for signifiers to include support for multi-step processes in order to implement features such as authentication. While our approach enables agents to interact with new services without having any requirement on their architecture or abilities, this implies that signifiers cannot be customized to the agent's abilities. For example, an agent able to use MCP can interact with an MCP server discovered through signifiers using our proposed SEM MCP server but it cannot interact directly with that discovered server. We propose, as future work, to modify the architecture of language agents to add native support for signifier resolution combined with protocol-specific signifiers. This would enable the agent to interact with a service using its native protocol (e.g., MCP) if the protocol is also supported by the agent. The scenario described in Section 4.2 demonstrates that our current approach requires the agents to query signifiers the agents have already queried before because the tools created from signifiers are overwritten after a new signifier query. To

address this limitation, we propose to extend our resolution of signifiers as tools so that an agent could consider a tool to be long-lived. We also plan to extend our model for signifiers to manage long-term processes, such as A2A tasks.

Limitations of the evaluation We design the agent goals specified in the agent profiles and the tool descriptions to be similar enough that one LLM (ministral-3:14b) can correctly determine whether a tool matches an agent's goal even when the maximum number of signifiers selected by the agent is 1 and the relevance threshold is 0.75. In practice, an LLM may not be able to correctly identify the best signifier or many signifiers may be equally relevant to the agent. The choices by the agent of the maximum number of signifiers and the relevance threshold mitigate this issue when LLMs fail to identify a single best relevant signifier.

6 Conclusion

We integrated some of the standards (MCP, A2A, UTCP, WoT TD) used to create the **Agentic Web** within **hypermedia Multi-Agent Systems (hMAS)**. This integration relies on creating a uniform hypermedia interface by applying a model for hypermedia signifiers derived from [30] to the standards we considered. We showed that such signifiers can be automatically created from the standards themselves and the information provided by the servers implementing these standards. We extended the Signifier Exposure Mechanism (SEM) with LLMs to enable the selection of signifiers using their natural language recommended contexts and information provided by the agent in its profile, including its natural language context and additional parameters (maximum signifiers, relevance threshold). We showed how agents can use tools that are created from signifiers and that can be exposed using any protocol or standard the agents are able to use to interact with tools. Through the uniform interface provided by signifiers, any language agent becomes able to interact with any service implementing tools or simple agent interaction through its preferred interface. We implemented these ideas in the lab environment to show how they could be used in practice and validate that signifiers enable agents to discover and interact with relevant Agentic Web services that may not be natively able to interact with. Our evaluation shows that LLMs can enable the effective filtering of signifiers depending on the user's goal, but not all LLMs are capable of doing this task.

In future work, we plan to address the limitations mentioned in Section 5, to provide support for other Agentic Web protocols (e.g., Agent Network Protocol [13]) and to integrate other types of agents (e.g. PRS agents [20]) as part of the same hMAS.

Acknowledgements I thank my PhD supervisor Simon Mayer, and co-supervisor Andrei Ciortea for their guidance as well as Danai Vachtsevanou for her support.

Declaration on Generative AI Generative AI was used to implement the proposed model and its evaluation, to edit the article, and to contribute to the design and creation of Figures 1 and 2.

References

1. Langchain, https://www.langchain.com/
2. Ollama, https://ollama.com/
3. Universal tool calling protocol (utcp), https://www.utcp.io/
4. Eclipse Language Model Operating System (LMOS) (2025), https://eclipse.dev/lmos/
5. thingctx (2026), https://github.com/thingctx/thingctx
6. Anthropic: Introducing the Model Context Protocol. https://www.anthropic.com/news/model-context-protocol (2024)
7. Berners-Lee, T., Cailliau, R., Groff, J.F., Pollermann, B.: World-wide web: the information universe. Internet Research **2**(1), 52–58 (1992)
8. Berners-Lee, T.J.: Information management: A proposal. Tech. rep. (1989)
9. Boissier, O., Ciortea, A., Harth, A., Ricci, A.: Autonomous agents on the web. In: Dagstuhl-Seminar 21072: Autonomous Agents on the Web. p. 100p (2021)
10. Boissier, O., Ciortea, A., Harth, A., Ricci, A., Vachtsevanou, D.: Agents on the web (dagstuhl seminar 23081). Dagstuhl Reports **13**(2), 71–162 (2023)
11. Brin, S., Page, L.: The anatomy of a large-scale hypertextual web search engine. Computer networks and ISDN systems **30**(1-7), 107–117 (1998)
12. Chaffer, T.J.: On the institutional origins of the agentic web. Harvard Journal of Law and Technology Digest, Forthcoming (2025)
13. Chang, G., Lin, E., Yuan, C., Cai, R., Chen, B., Xie, X., Zhang, Y.: Agent network protocol technical white paper. arXiv preprint arXiv:2508.00007 (2025)
14. Charpenay, V., Käbisch, S.: On modeling the physical world as a collection of things: The W3C thing description ontology. In: European Semantic Web Conference. pp. 599–615. Springer (2020)
15. Charpenay, V., Käbisch, S., Kosch, H.: Introducing thing descriptions and interactions: An ontology for the web of things. In: SR+ SWIT@ ISWC. pp. 55–66 (2016)
16. Charpenay, V., Käfer, T., Harth, A.: A unifying framework for agency in hypermedia environments. In: International Workshop on Engineering Multi-Agent Systems. pp. 42–61. Springer (2021)
17. Ciortea, A., Mayer, S., Gandon, F., Boissier, O., Ricci, A., Zimmermann, A.: A decade in hindsight: the missing bridge between multi-agent systems and the world wide web. In: Proceedings of the International Conference on Autonomous Agents and Multiagent Systems (2019)
18. Duan, Q., Lu, Z.: AI Agent Communications in the Future Internet—Paving a Path Toward the Agentic Web. Future Internet **18**(3), 171 (2026)
19. Fielding, R.T., Taylor, R.N.: Principled design of the modern web architecture. ACM Transactions on Internet Technology (TOIT) **2**(2), 115–150 (2002)
20. Georgeff, M.P., Lansky, A.L.: Reactive reasoning and planning. In: AAAI. vol. 87, pp. 677–682 (1987)
21. Group, W.R.D.S.W.: Shapes Constraint Language (SHACL). World Wide Web Consortium (W3C) (2017), https://www.w3.org/TR/shacl/, w3C Recommendation, 20 July 2017
22. Ilin, M.: Converging hypermedia, protocols, and knowledge architectures: A new paradigm for grounded and interoperable llm-agent systems (2025)
23. Käbisch, S., Korkan, E., McCool, M.: Web of things (wot) thing description 1.1. W3C proposed recommendation, W3C (Jul 2023), https://www.w3.org/TR/2023/PR-wot-thing-description11-20230711/

24. Lemée, J., Vachtsevanou, D., Mayer, S., Ciortea, A.: Signifiers for conveying and exploiting affordances: from human-computer interaction to multi-agent systems. Annals of Mathematics and Artificial Intelligence (2024)
25. Mayer, S., Ciortea, A., Ricci, A., Robles, M.I., Kovatsch, M., Croatti, A.: Hypermedia to connect them all: Autonomous hypermedia agents and socio-technical interactions. Internet Technology Letters **1**(4), e50 (2018)
26. Mialon, G., Dessi, R., Lomeli, M., Nalmpantis, C., Pasunuru, R., Raileanu, R., Roziere, B., Schick, T., Dwivedi-Yu, J., Celikyilmaz, A., et al.: Augmented language models: a survey. Transactions on Machine Learning Research (2023)
27. Norman, D.: The design of everyday things: Revised and expanded edition. Basic books (2013)
28. Sumers, T., Yao, S., Narasimhan, K., Griffiths, T.: Cognitive architectures for language agents. Transactions on Machine Learning Research (2023)
29. Surapaneni, R., Jha, M., Vakoc, M., Segal, T.: Announcing the Agent2Agent protocol (A2A). https://developers.googleblog.com/en/a2a-a-new-era-of-agent-interoperability/ (2025)
30. Vachtsevanou, D., Ciortea, A., Mayer, S., Lemée, J.: Signifiers as a first-class abstraction in hypermedia multi-agent systems. In: Proceedings of the 2023 International Conference on Autonomous Agents and Multiagent Systems (2023)
31. Vachtsevanou, D., de Lima, B., Ciortea, A., Hübner, J.F., Mayer, S., Lemée, J.: Enabling BDI agents to reason on a dynamic action repertoire in hypermedia environments. In: AAMAS (2024)
32. Wang, Z., Cheng, Z., Zhu, H., Fried, D., Neubig, G.: What are tools anyway? a survey from the language model perspective. In: First Conference on Language Modeling (2024)
33. Yang, Y., Ma, M., Huang, Y., Chai, H., Gong, C., Geng, H., Zhou, Y., Wen, Y., Fang, M., Chen, M., et al.: Agentic Web: Weaving the Next Web with AI agents. arXiv preprint arXiv:2507.21206 (2025)

A Descriptions for Agentic Web services

Listing 4 is a description of an MCP tool.
Listing 5 is an example of an A2A agent card.
Listing 6 is a description of a tool from a UTCP manual.
Listing 7 is an action affordance as described in a WoT Thing Description.

B Signifiers for Agentic Web Standards

Listing 8 presents a signifier created to read an MCP resource.

Listing 9 presents a signifier constructed for the A2A agent described by Listing 5.

Listing 10 presents a signifier constructed for the UTCP tool described by Listing 6.

Listing 11 presents a signifier constructed for the WoT TD action affordance described by Listing 7.

```
{
  "name": "provide_feedback",
  "title": null,
  "description": "Provide feedback for the current goal. If achieved is true, the
  ↪  current goal is cleared.",
  "inputSchema": {
    "properties": {
      "achieved": {
        "description": "Whether the goal has been achieved",
        "title": "Achieved",
        "type": "boolean"
      },
      "feedback": {
        "description": "The feedback to provide",
        "title": "Feedback",
        "type": "string"
      }
    },
    "required": [
      "achieved",
      "feedback"
    ],
    "title": "provide_feedbackArguments",
    "type": "object"
  },
  "outputSchema": null,
  "icons": null,
  "annotations": null,
  "meta": null,
  "execution": null
}

```

Listing 4: MCP tool description for providing feedback to a user about goal achievement

```
{
  "capabilities": {
    "streaming": true
  },
  "defaultInputModes": [
    "text"
  ],
  "defaultOutputModes": [
    "text"
  ],
  "description": "Formalizes goal descriptions into move/rotate commands.",
  "name": "Formalizer Agent",
  "preferredTransport": "JSONRPC",
  "protocolVersion": "0.3.0",
  "skills": [
    {
      "description": "Converts a natural language goal into move(d) or rotate(a).",
      "examples": [
        "move forward 2 meters",
        "rotate 90 degrees",
        "turn left 45 degrees"
      ],
      "id": "formalize_goal",
      "name": "Formalize goal",
      "tags": [
        "formalizer",
        "goal"
      ]
    }
  ],
  "supportsAuthenticatedExtendedCard": false,
  "url": "http://localhost:9997/",
  "version": "1.0.0"
}
```

Listing 5: A2A Agent Card for an agent converting a goal in natural language into a formal representation

```
{
      "description": "Perform a move(d) or rotate(a) operation on the robot. Request
      ↪  body must be plain text (MIME type text/plain, no JSON), of the form
      ↪  operation_name(param) e.g. move(1) or rotate(1)",
      "inputs": {
        "description": "Operation command as plain text, e.g. move(10).",
        "type": "string"
      },
      "name": "operation",
      "outputs": {},
      "tags": [],
      "tool_call_template": {
        "call_template_type": "http",
        "content_type": "text/plain",
        "http_method": "POST",
        "url": "http://localhost:8086/operation"
      }
    }
```

Listing 6: UTCP tool description for a tool to control a robot

```
<#aff> a td:ActionAffordance ;
            td:description "Perform a move(d) or rotate(a) operation. Request body must be plain text (MIME type text/plain, no JSON), of the form operation_name(param) e.g. move(1) or rotate(1)" ;
            td:hasForm [ htv:methodName "POST" ;
                    hctl:forContentType "text/plain" ;
                    hctl:hasOperationType td:invokeAction ;
                    hctl:hasTarget <http://localhost:8086/operation> ] ;
            td:hasInputSchema [ a js:StringSchema ] ;
            td:name "operation" ]
```

Listing 7: TD action affordance

```


<#sig> a hmas:Signifier ;
    rdfs:label "goal_mcp_CurrentGoal" ;
    hmas:recommendsContext [ rdfs:comment "Returns the current goal defined by the user." ] ;
    hmas:signifies [ td:hasForm [ http:headers ( [ http:fieldName "Accept" ;
                                http:fieldValue "application/json, text/event-stream" ] [ http:fieldName "MCP-Protocol-Version" ;
                                http:fieldValue "2025-06-18" ] [ http:fieldName "Mcp-Session-Id" ;
                                http:fieldValue "fdfccc0a27f5477893dbea501d0834fa" ] ) ;
                    http:methodName "POST" ;
                    hctl:forContentType "application/json" ;
                    hctl:hasTarget <http://localhost:9996/mcp> ] ;
            td:hasInputSchema [ a js:ObjectSchema ;
                    js:properties [ a js:StringSchema ;
                            js:const "2.0" ;
                            js:propertyName "jsonrpc" ],
                        [ a js:ObjectSchema ;
                            js:properties [ a js:StringSchema ;
                                    js:const "goal://current" ;
                                    js:propertyName "uri" ] ;
                            js:propertyName "params" ;
                            js:required "uri" ],
                        [ a js:StringSchema ;
                            js:const "resources/read" ;
                            js:propertyName "method" ],
                        [ a js:IntegerSchema ;
                            js:const 1 ;
                            js:propertyName "id" ] ;
                    js:required "id",
                        "jsonrpc",
                        "method",
                        "params" ] ] .
```

Listing 8: Signifier for MCP resource

```rdf-turtle


<#sig> a hmas:Signifier ;
    rdfs:label "formalizer_Formalize goal" ;
    hmas:recommendsContext [ rdfs:comment "Converts a natural language goal into move(d) or rotate(a)." ] ;
    hmas:signifies [ td:hasForm [ http:methodName "POST" ;
                    hctl:forContentType "application/json" ;
                    hctl:hasTarget <http://localhost:9997> ] ;
            td:hasInputSchema [ a js:ObjectSchema ;
                    js:additionalProperties false ;
                    js:properties [ a js:NullSchema,
                                js:NumberSchema,
                                js:StringSchema ;
                            js:propertyName "id" ],
                        [ a js:ObjectSchema ;
                            js:additionalProperties false ;
                            js:properties [ a js:ObjectSchema ;
                                    js:additionalProperties false ;
                                    js:properties [ a js:StringSchema ;
                                            rdfs:comment "A string that identifies the message" ;
                                            js:propertyName "messageId" ],
                                        [ a js:ArraySchema ;
                                            js:description "A JSON array. Each element is a JSON object whose first property is called \"kind\". Allowed \"kind\" values from input modes: \"text\". If \"kind\" is \"text\", the other field is \"text\" and its value is a string. If \"kind\" is \"data\", the other field is \"data\" and its value is a JSON object." ;
                                            js:maxItems 1 ;
                                            js:minItems 1 ;
                                            js:propertyName "parts" ],
                                        [ a js:StringSchema ;
                                            js:const "user" ;
                                            js:propertyName "role" ] ;
                                    js:propertyName "message" ;
                                    js:required "parts",
                                        "role" ] ;
                            js:propertyName "params" ;
                            js:required "message" ],
                        [ a js:StringSchema ;
                            js:const "2.0" ;
                            js:propertyName "jsonrpc" ],
                        [ a js:StringSchema ;
                            js:const "message/send" ;
                            js:propertyName "method" ] ;
                    js:required "id",
                        "jsonrpc",
                        "method",
                        "params" ] ] .

```

Listing 9: Signifier for A2A agent

```
<#sig> a hmas:Signifier ;
    rdfs:label "cherrybot_utcp_operation" ;
    hmas:recommendsContext [ rdfs:comment "Perform a formal operation on the robot.
    ↪  Request body must be plain text (MIME type text/plain, no JSON), of the form
    ↪  operation_name(param) e.g. move(1) or rotate(1)" ] ;
    hmas:signifies [ td:hasForm [ http:methodName "POST" ;
                     hctl:forContentType "text/plain" ;
                     hctl:hasTarget <http://localhost:8086/operation> ] ;
            td:hasInputSchema [ a js:StringSchema ;
                     js:description "Operation command as plain text, e.g. move(10)." ]
                     ↪  ] .
```

Listing 10: Signifier for UTCP tool

```
<#sig> a hmas:Signifier ;
    rdfs:label "cherrybot_proxy_operation" ;
    hmas:recommendsContext [ rdfs:comment "Perform a formal operation on the robot.
    ↪  Request body must be plain text (MIME type text/plain, no JSON), of the form
    ↪  operation_name(param) e.g. move(1) or rotate(1))" ] ;
    hmas:signifies [ td:hasForm [ http:methodName "POST" ;
                     hctl:forContentType "text/plain" ;
                     hctl:hasOperationType "invokeaction" ;
                     hctl:hasTarget <http://localhost:8086/operation> ] ;
            td:hasInputSchema [ a js:StringSchema ] ] .
```

Listing 11: Signifier for WoT TD Action Affordance

C Evaluation Tasks

For the evaluation, we design a few tasks in which different LLMs are evaluated. A task is characterized by a natural language description of a *context* in an agent profile and a set of tools to select among a broader set of tools. The LLMs are evaluated by their ability to identify which tools should be used given the *context*. The available tools are: “goal_mcp_CurrentGoal”, “goal_mcp_provide_feedback”, “formalizer_Formalize goal”, “cherrybot_utcp_initialize”, and “cherrybot_utcp_operation”. The task "Goal or Feedback" does not correspond to any step of the scenario but is added to provide an example where more than one tool needs to be selected.

Goal: The context is "The agent wants to read the user goal." and the tool to select is only “goal_mcp_CurrentGoal”.

Feedback: The context is "The agent wants to provide feedback to the user." and the tool to select is only: “goal_mcp_provide_feedback”.

Formalizer: The context is "The agent wants to convert the natural language goal into a formal representation." and the tool to select is only: “formalizer_Formalize goal”.

Goal/FB: The context is "The agent wants to read the goal or provide feedback to the user." and the tools to select are: “goal_mcp_CurrentGoal” and “goal_mcp_provide_feedback”.

Robot: The context is: "The agent wants to perform a formal operation on the robot." and the tool to select is only: “cherrybot_utcp_operation”. This task is only performed with the UTCP interface for the robot proxy as the WoT TD interface would provide the same *recommended context* for the signifier to select.