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 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, and has been used to allow agents to interact with Web services and physical devices in open Web environments through hypermedia standards, such as the Web of Things (WoT) Thing Description. 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.
Building on prior work about hypermedia multi-agent systems, we develop a uniform hypermedia interface applicable to different Agentic Web 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 a natural language-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, which rely on Large Language Models (LLMs) to interact with the world [26], are increasingly interacting with Web services on behalf of human users. This evolution results in the development of an **Agentic Web**, designed for use by language agents in addition to human users [33]. 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 [27,33].

Agent environments [32], such as Web environments [16], are core components of Multi-Agent Systems (MAS). However, while the Agentic Web develops Web environments for agents, it remains mostly disconnected from Multi-Agent Systems (MAS) research on such environments.

Research on **hypermedia Multi-Agent Systems (hMAS)** suggests that approaches to creating Multi-Agent Systems on the Web that are large-scale, open, interoperable, and dynamic should be aligned with the architecture of the Web [17]. To do so, hMAS rely on hypermedia to decouple the development of agents from the development of their environments, enabling both to evolve independently [23,14]. Despite the potential of the hMAS vision for creating Web-based MAS, Fabien Gandon explains that it has yet to be applied at scale because hMAS is facing a cold-start problem, due to missing a critical use case [9,10]. We consider that 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 conceived for the Agentic Web. 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. Applying hypermedia to the emerging standards for the Agentic Web, could thus enable their integration into a common Agentic Web and realize the hMAS vision by providing contextually relevant hypermedia information for agents about service implementing these standards.

Our research question is: "How can we build a general uniform hypermedia interface for established and emerging standards that enables Web environments to provide language agents with contextually relevant information?".

Our approach to address this research question 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 [28,22]. 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 [28,22]. We apply the SEM for 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 information that can be exploited 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 using natural language information introduced in both the signifier and the agent's profile.

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 MCP. 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 associated and 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 software agents [15], called *language agents* [26]. Language agents perceive and act in the environment using the ReAct prompting technique [34], combined with the use of tools [24,30]. Although tools have been implemented directly as part of agent frameworks (e.g., Langchain [3]), framework-independent tools are usable across agent frameworks, creating a common ecosystem of tools for agents. This is why 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], and is becoming a core technology of the Agentic Web to connect agents with Web services [33]. MCP enables a separation of concerns between designers of language agents and designers of tools. An MCP client interacts with an MCP server using the JSON-RPC 2.0 protocol [1] and the streamable HTTP transport for MCP enables LLM applications and agents to interact with Web-based MCP servers. MCP requires tool developers to create specific MCP servers to enable external users to interact with their tools.

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

Universal Tool Calling Protocol (UTCP) is another protocol to enable agents to interact with tools [4]. Unlike MCP that requires servers to implement an MCP interface for tools, UTCP provides UTCP manuals to allow agents to directly interact with native APIs (e.g., HTTP APIs). A manual provides all the information that an agent needs to interact with a tool, including the requests (e.g., HTTP requests) to perform to interact with the tool, and natural language descriptions of the tool and its parameters. UTCP is a versatile protocol that supports manuals for different protocols (e.g., HTTP, gRPC).

Agent2Agent (A2A) is a protocol that allows users to send messages to agents that perform tasks on their behalf and return responses [27]. 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. A2A relies on JSON-RPC 2.0 to exchange messages between users and agents, and the HTTP protocol is used to convey these messages.

Web of Thing (WoT) Thing Description (TD) is a standard for the WoT that defines interaction affordances that indicate how to interact with a given Thing [21]. This standard relies on Semantic Web technologies, such as JSON-LD and the WoT TD Ontology [12], which enable extensibility to new use cases and support for a variety of protocols [21]. The Eclipse Language Model Operating System (LMOS) uses TDs to describe language agents and their tools [1].

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, none of these standards enables the contextual exposure of the information they provide to agents. As a result, agents may perceive all available information (e.g., all tools described by an UTCP manual) instead of the ones that are actually relevant to the agent.

2.2 Providing Contextualized Information to Agents Using Hypermedia

Situated agents make decisions based on the local information they discover in their environments [31]. On the Web, situatedness is implemented through hypermedia, which provides information to clients, such as agents, about an action that they can perform associated with hypermedia controls to perform the action.

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 [14]. The vision of hMAS emphasizes the need for hypermedia in

Web-based MAS, while the vision of the Agentic Web emphasizes the need for generative AI in such MAS. Both visions are complementary approaches towards developing Web-based MAS [20]. The Web was initially developed to provide a uniform hypermedia interface for different information management systems [8,7], and therefore a uniform hypermedia interface can be developed for agents to interact with protocols developed for the Agentic Web.

Signifiers for software agents on the Web The concept of signifier, as defined in User-Centered Design and applied in Human-Computer Interaction, refers to information about what an object (physical or virtual) could be used for and how to use it [25]. 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 [28,22] to expose contextually relevant signifiers for the agents. Although so far the development of agents using signifiers has mainly focused on BDI agents [29], signifiers can be exploited by heterogeneous types of agents [28], including language agents. Additionally, hypermedia signifiers can 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. We first explain our ontological models for signifiers and then we focus on creating such signifiers for MCP servers, A2A agents, UTCP tools, and WoT Things (explained in Section 2.1), in Section 3.1. Signifiers constructed using this model enable language agents to interact with any service that implements the standard. In Section 3.2, we present our extension of the Signifier Exposure Mechanism, described in [28], to support contextual exposure of signifiers for language agents. Finally, in Section 3.3, we explain how the affordances described by signifiers can be exploited 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, A2A Agent Cards, UTCP manuals, and WoT Thing Descriptions). A resource profile exposes signifiers, using the concept *hmas:exposesSignifier*, to indicate how an agent can interact with the resource described by the profile.

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

An Ontological Model for Signifiers for Language Agents We present an ontological model for signifiers as an extension of the model presented in [28,29] (see Section 2.2) and the corresponding hMAS ontology. This ontological model also relies on other ontologies, listed in Listing 1 (all prefixes for ontologies that we used in the article are defined there). The model reuses the optional recommended *abilities* that an agent needs to use a signifier (hmas:recommendsAbility). We give the *recommended context* (hmas:recommendsContext) a natural language description using the *rdfs:comment* property. This description is used by language agents to decide whether to exploit or not the action possibility described by the signifier. The signified action possibility (hmas:signifies) is an interaction affordance, as defined by the Thing Description (TD) (See Section 2.1). We focus on affordances that can be exploited using the HTTP protocol because support for the HTTP protocol is common to the different standards that we consider. 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, called *schema* using *js:description*. The hypermedia controls also define the *content type*, and the *target* URL. The form can also define the HTTP *headers*, using *http:headers*, and the *method* using *http:methodName*. We use the Shapes Constraint Language (SHACL) [19] to provide a formal representation of the general form we propose for signifiers in Listing 1. This 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).

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 for a partial example of such a tool description. We explain how to automatically create signifiers to indicate to agents how to use a given MCP tool, when the streamable HTTP transport is used for MCP,using both the JSON description of the tool and information concerning the MCP protocol. The signifier constructed for a specific tool contains a *label*, which has the following form: {server_name}_{name}, where {server_name} is a user-defined name that name that is associated with 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 formal specification on how to interact with the tool includes a *header* called *MCP-Session-Id*. When the signifier is created, the token of this header is generated by the MCP server and will be used for future interaction. 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 streamable HTTP transport. The *method* indicated in the signifier is

```
@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

"POST", and the *content type* is "application/json' for the JSON-RPC 2.0 payload. The *schema* relies on JSON-RPC 2.0 and adds elements specific to MCP, including a method, which is "tools/call" to call a tool. The parameters to specify 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 for an example).

```
1
2  <#sig> a hmas:Signifier ;
3      rdfs:label "goal_mcp_provide_feedback" ;
4      hmas:recommendsContext [ rdfs:comment "Provide feedback for the current goal." ] ;
5      hmas:signifies [ td:hasForm [ http:headers ( [ http:fieldName "Accept" ;
6                                   http:fieldValue "application/json, text/event-stream"
                                    ↪  ] [ http:fieldName "MCP-Protocol-Version" ;
7                                   http:fieldValue "2025-06-18" ] [ http:fieldName
                                    ↪  "Mcp-Session-Id" ;
8                                   http:fieldValue "b4f1b002aac149869708507a1168ae1f" ] )
                                    ↪  ;
9                      http:methodName "POST" ;
10                     hctl:forContentType "application/json" ;
11                     hctl:hasTarget <http://localhost:9996/mcp> ] ;
12             td:hasInputSchema [ a js:ObjectSchema ;
13                     js:properties [ a js:IntegerSchema ;
14                             js:propertyName "id" ],
15                         [ a js:StringSchema ;
16                             js:const "2.0" ;
17                             js:propertyName "jsonrpc" ],
18                         [ a js:StringSchema ;
19                             js:const "tools/call" ;
20                             js:propertyName "method" ],
21                         [ a js:ObjectSchema ;
22                             js:properties [ a js:ObjectSchema ;
23                                     rdfs:comment "Provide feedback for the current
                                        ↪  goal." ;
24                                     js:properties [ a js:StringSchema ;
25                                             js:description "The feedback to provide" ;
26                                             js:propertyName "feedback" ],
27                                         [ a js:BooleanSchema ;
28                                             js:description "Whether the goal has been
                                                ↪  achieved" ;
29                                             js:propertyName "achieved" ] ;
30                                     js:propertyName "arguments" ;
31                                     js:required "achieved",
32                                         "feedback" ],
33                                 [ a js:StringSchema ;
34                                     js:const "provide_feedback" ;
35                                     js:propertyName "name" ] ;
36                             js:propertyName "params" ;
37                             js:required "arguments",
38                                 "name" ] ;
39                     js:required "id",
40                         "jsonrpc",
41                         "method",
42                         "params" ] ] .
```

Listing 2: Signifier for MCP tool

Signifiers for A2A A2A agents indicate their information through a JSON agent card, provided at, where {BASE_URL} is the base URL of the server: `{BASE_URL}/.well-known/agent-card.json`. 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 is used by external agents (referred to as users) to determine 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 JSON-RPC 2.0. gRPC, or HTTP+JSON/REST protocol. We consider the JSON-RPC 2.0 implementation, a common implementation of the protocol. The A2A protocol offers different potential affordances to users. We design signifiers for the sending a message to an A2A agent.

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 the name of the agent from the Agent Card, 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 *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 method being "message/send". The "parts" parameter indicates the different elements of the message. This parameter is a JSON array, with a corresponding JSON schema and a description that includes the input modes of the skill that determine the potential content of the message. See Listing 9 for a signifier corresponding to the A2A agent described in Listing 5.

Signifiers for UTCP The Universal Tool Calling Protocol (UTCP) enables language agents to directly interact with different types of APIs, including HTTP APIs, without relying on an intermediate MCP server [4]. To do so, the agent relies on a UTCP manual that 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 perform interactions with the tool. In Listing 6, we present a UTCP tool as described by a UTCP manual.

For each tool with HTTP transport and providing parameters in the request body, we create a signifier, whose *label* is {manual_name}_{tool_name}, where {manual_name} is a name that the designer of the SEM registers for 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 signified action possibility is created from both the *tool_call_template* and the *inputs* fields of the tool description. See Listing 10 for a signifier corresponding to the tool described in Listing 6.

Signifiers for WoT Things WoT Things expose Thing Descriptions (TDs) that indicate the interaction affordances (property, action, or event affordances)

that the WoT Thing. We focus on creating signifiers for affordances described by the TD. See Listing 7 for a partial example of an action affordance in a TD. We rely on the Thing Description Ontology [13] (prefix: *td*) to create these signifiers. The *label* of the signifier is {thing_name}_{affordance_name}, where {thing_name} is the name of 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 not present, no signifier for the affordance is created in the resource profile of the Thing. This is because our model of signifiers requires each signifier to have a natural language description. The signified affordance of the signifier 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. If many forms are available, only one is selected to match the proposed ontological model. 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 agents. See Listing 11 for a signifier corresponding to the TD interaction affordance described in Listing 7.

3.2 Selection of Signifiers Using Natural Language Context

The Signifier Exposure Mechanism (SEM), defined in [28], provides a hypermedia interface that enables agents to perceive contextual information about affordances. We enhance the SEM selection mechanism using an LLM-based assessment of contextual relevance. To do so, we create a Boolean-valued function that indicates whether the signifier is relevant given the agent profile. This function relies on an LLM call, whose output is parsed as a Boolean. We construct a parametric prompt in Python, presented in Listing 3, 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 proposed prompt is designed under the assumption that false negatives are worse than false positives because the agent needs to perceive at least some useful signifiers to be able to achieve its goals, but other implementations of the prompt remain possible. If the LLM returns either "True" or "False", the result of the function is the corresponding boolean value; otherwise the result is the boolean "True". The LLM temperature can be set to 0 to reduce the variability of the results and thus increase their reproducibility. This evaluation of the contextual relevance can be combined with other selection mechanisms used for selection (e.g., ability matching) to determine whether to expose a given signifier to an agent.

3.3 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 exploit the action possibilities described by the signifiers it perceives. To do so, we derive tools from the information provided by signifiers to enable agents to interact with the services or tools described by the signifiers. While these tools could be instantiated using

```
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 is True if the information provided by the signifier, according to
        ↪  its context, is directly relevant to the agent, according to its profile
        ↪  context. If the signifier context is not relevant to the agent, the answer
        ↪  is False.\n"
        "Avoid both false negatives and false positives, but false negatives are
        ↪  worse."
        "Respond with exactly 'True' or 'False'.\n"
        f"Profile context: {profile_context}\n"
        f"Signifier context: {signifier_context}"
    )
```

Listing 3: Prompt used by the SEM

many different frameworks (e.g., Langchain [3]), we rely on MCP to make the created tools interoperable across agent frameworks and we present a way to create MCP tools from signifiers. 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, provided 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 tools.

This mapping enables language agents with native support for MCP to exploit 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. This transformation introduces overhead and loss of expressiveness for signifiers corresponding to MCP tools. While we mapped signifiers to MCP tools, similar mappings can be constructed for other standards, such as UTCP, thus enabling an agent to resolve any tool into its available interface. A language agent could use a special tool to query signifiers, which are themselves created from the profiles provided by the different types of services considered (MCP, A2A, UTCP, WoT Thing). The retrieved signifiers are then converted as tools, using MCP (or potentially some other interface). Finally, the agent is able to interact with the original services using these tools. 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.

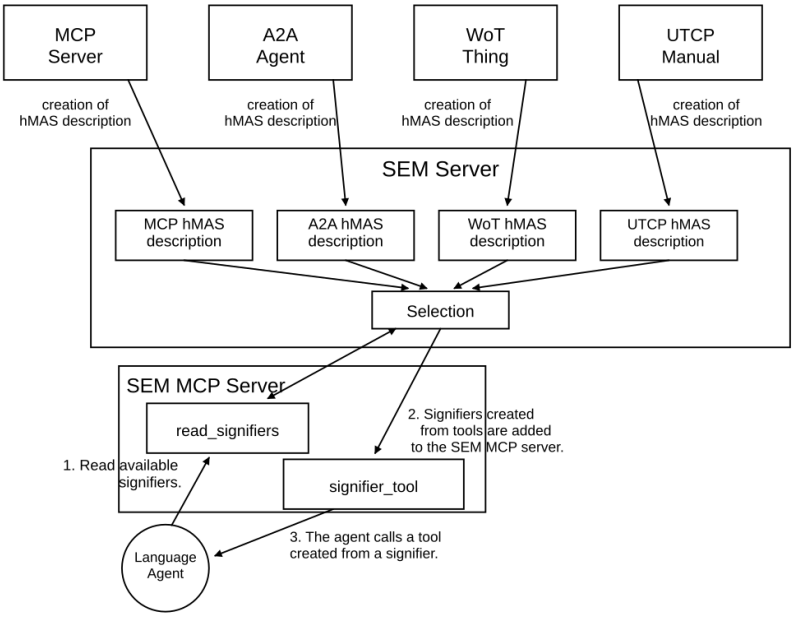

Fig. 1. Workflow of a language agent querying and using signifiers

4.1 Implementation of the Signifier Exposure Mechanism for Language Agents

We implement the Signifier Exposure Mechanism (SEM) platform as a Python Flask application [2][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, from 2 parameters: an identifier for the artifact (called `{artifact_id}`) and an entrypoint URL. These functions retrieve information about the instance (e.g., an MCP list of tools) and construct the resource profile for the instance (See Section 3.1). The endpoint `/artifacts/{artifact_id}` exposes the resource profile an artifact registered with the given `{artifact_id}`. The SEM relies on an LLM backend, supporting both models from the OpenAI API and local models using Ollama [5], to implement the selection mechanism for signifiers (see Section 3.2).

We implement an MCP server to allow language agents to interact with the SEM platform. This server provides the following tools: read_signifiers, to query signifiers for a given profile URL and make them available as new MCP tools using the mechanism described in Section 3.3, provided as input to the tool; register_profile to create an initial profile; and update_profile, to update the natural language context of the profile.

We use the Langchain framework [3] to implement a cognitive architecture for language agents, following the model for cognitive language agent presented in [26]. We use this cognitive architecture to instantiate the agents from our scenario.

4.2 Scenario

We define a scenario to evaluate our proposed system to enable a language agent to interact with Web services relying on different standards and protocols with which the agent is not natively able to interact.

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 its own acquired credentials. The proxy is able to execute 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 indicates an informal description in natural language of the goal that the agent needs to achieve, and a tool for the agent to provide feedback.

The Formalizer Agent, is an A2A language agent that can return a formal description of the goal from an informal description. This agent is able to use

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

internal tools in order to determine the value of the operation with the correct units.

The Executor Agent is an agent dedicated to achieving the user goal. It is not able to interact directly with tools and agents using the A2A, UTCP, or TD standards; and it can only interact directly with one MCP server: the one provided by the SEM as presented in Section 4.1.

Scenario Workflow At the start of the scenario, the Executor Agent first registers a profile with name "executor". The agent does not know the user goal it has to achieve and 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, retrieves the goal (e.g., "Move the robot by 10 centimeters") and stores it in its working memory. The agent updates its profile to indicate that it wants to provide feedback to the user and reads the new signifiers available. Using the appropriate created tool, the agent provides feedback to the user that it started to work on the goal.

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 change the position of the robot using a formal goal. The agent reads signifiers and controls the robot using the formal goal with the created tool. The Executor Agent updates its profile to indicate its desire to provide feedback to the user. The agent selects the corresponding created tool, after reading the signifiers, and provides feedback to the user.

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 the underlying LLM being used. To perform this evaluation, we use local open source models: *gemma3:270m* (1), *gemma3:1b* (2), *ministral-3:8b* (3), *ministral:14b* (4) through Ollama. We also use *gpt-4.1-mini* (5) and *gpt-5-mini* (6) 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. They are described for each task in the Appendix C. Although task "Goal/Feedback" 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 measure the precision (P) and recall (R) of each task for each LLM, with a temperature of 0 to have close to deterministic results for

each model/task. We present the results in Table 1, where each model number corresponds to the ones introduced for each LLM.

Table 1. Precision and recall of different LLMs for different agent context

Task/Model	1	2	3	4	5	6
Goal	P:0, R:0	P:0.20, R:1	P:1, R:1	P:1, R:1	P:0.33, R:1	P:0.33, R:1
Feedback	P:0, R:0	P:0.20, R:1	P:1, R:1	P:1, R:1	P:0.33, R:1	P:0.50, R:1
Formalizer	P:0, R:0	P:0.20, R:1	P:0.50, R:1	P:0.50, R:1	P:0.50, R:1	P:0.25, R:1
Goal/Feedback	P:0, R:0	P:0.40, R:1	P:1, R:1	P:1, R:1	P:0.67, R:1	P:0.67, R:1
Robot Control	P:0.33, R:1	P:0.20, R:1	P:0, R:0	P:0.25, R:1	P:0.20, R:1	P:0.20, R:1

We evaluate whether the Executor Agent, relying on the internal *gpt-4.1-mini* LLM, can perform the task successfully if the interface to the robotic arm proxy is registered from either a Thing Description or an UTCP manual, with the respective goals "Move the robot by 10 centimeters" and "Rotate the robot by 12 degrees". In both cases, using *gpt-4.1-mini* for the SEM, the agent can successfully complete the scenario. Using the LLM *gpt-4.1-mini* for the SEM, and a simulation of the robotic arm for repeated tests, we launch agents to achieve 15 times the goal "Move the robot by 10 centimeters" for both an UTCP interface and WoT TD interface to the robot simulation proxy. The agent achieves the goal successfully 14 out of 15 iterations for the UTCP interface and every iterations for the WoT interface.

5 Discussion

We use signifiers to indicate to language agents how to interact with a service implementing an Agentic Web standard, which may not be directly usable by the agents. To provide agents with contextually relevant signifiers and thus reduce their cognitive load, we extend the SEM with LLMs to be able to perform an adequate selection using the natural language information they provide. Our evaluation shows that different LLMs evaluate differently whether a given signifier is relevant to an agent with a given profile give different results. Some models, including *gemma:270m*, and *ministral-3:8b*, have a recall rate of 0 on some tasks, thus not displaying any relevant tool for the agent for these tasks. All other models have a recall rate of 1 on all tasks and can therefore be used to show all the signifiers required to achieve the goal. The precision of *gemma3:1b* is as low as possible for a recall of 1, which means it indicates all available signifiers as relevant and thus does not perform an effective selection mechanism, which could result in language agents in environments with many signifiers to be highly inefficient. The models *gpt-4.1-mini*, *gpt-5-mini* and *ministral-3:14b* have higher levels of precision while consistently having a recall of 1. These perform an effective selection of signifiers while enabling the agent to perform its task successfully. The results of *ministral-3:14b* show that even local open source

models can perform better than cloud models (*gpt-4.1-mini* and *gpt-5-mini*) for the task. As future work, we propose to update the selection mechanism to mitigate the risk of potentially hiding useful signifiers to agents by giving more options to agents to select the signifiers they can perceive.

While our model for signifiers enable agents to interact with services implementing many different standards, some features from the standards that we integrate cannot be integrated due to the limitation of the proposed model for only single HTTP requests as signified behavior. This is why our proposed model for MCP signifiers needs to hardcode the MCP-Session-Id token. Although this is not an issue if the SEM is only usable by trusted agents, it would create significant security issues in open environments. Our support for UTCP is limited to manuals for HTTP APIs and JSON bodies for tool parameters. Our support for A2A communication does not include long-running tasks. To address these limitations, our future work includes support for multi-step processes, more complex HTTP requests, and support for non-HTTP requests.

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 [28] 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 natural language information and we showed how agents can use tools created from signifiers. We implemented and evaluated these ideas in the lab environment to show how they could be used in practice.

As future work, we plan to address the limitations mentioned in Section 5; provide support for other Agentic Web protocols (e.g., Agent Network Protocol [11]); and integration of other agents (e.g., PRS agents [18]) as part of the same hMAS.

7 Acknowledgements

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

8 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 Figure 1.

References

1. Eclipse Language Model Operating System (LMOS) (2025), https://eclipse.dev/lmos/
2. Flask (2025), https://flask.palletsprojects.com/en/stable/
3. Langchain (2025), https://www.langchain.com/
4. Universal tool calling protocol (utcp) (2025), https://www.utcp.io/
5. Ollama (2026), https://ollama.com/
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. 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)
12. 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)
13. 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)
14. 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)
15. Chowa, S.S., Alvi, R., Rahman, S.S., Rahman, M.A., Raiaan, M.A.K., Islam, M.R., Hussain, M., Azam, S.: From language to action: a review of large language models as autonomous agents and tool users. Artificial Intelligence Review (2026)
16. Ciortea, A., Boissier, O., Ricci, A.: Engineering world-wide multi-agent systems with hypermedia. In: Engineering Multi-Agent Systems: 6th International Workshop, EMAS 2018, Stockholm, Sweden, July 14-15, 2018, Revised Selected Papers 6. pp. 285–301. Springer (2019)
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. Georgeff, M.P., Lansky, A.L.: Reactive reasoning and planning. In: AAAI. vol. 87, pp. 677–682 (1987)
19. 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
20. Ilin, M.: Converging hypermedia, protocols, and knowledge architectures: A new paradigm for grounded and interoperable llm-agent systems (2025)
21. 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/

22. 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 **92**(4), 815–835 (2024)
23. 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)
24. 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)
25. Norman, D.: The design of everyday things: Revised and expanded edition. Basic books (2013)
26. Sumers, T., Yao, S., Narasimhan, K., Griffiths, T.: Cognitive architectures for language agents. Transactions on Machine Learning Research (2023)
27. 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)
28. 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. pp. 1200–1208 (2023)
29. 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)
30. 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)
31. Weyns, D., Holvoet, T.: A formal model for situated multi-agent systems. Fundamenta Informaticae **63**(2-3), 125–158 (2004)
32. Weyns, D., Omicini, A., Odell, J.: Environment as a first class abstraction in multiagent systems. Autonomous agents and multi-agent systems **14**(1), 5–30 (2007)
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)
34. Yao, S., Zhao, J., Yu, D., Du, N., Shafran, I., Narasimhan, K.R., Cao, Y.: React: Synergizing reasoning and acting in language models. In: The eleventh international conference on learning representations (2022)

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.

```
{
  "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

```json
{
  "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

```json
{
      "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

```
1
2
3   <#sig> a hmas:Signifier ;
4       rdfs:label "formalizer_Formalize goal" ;
5       hmas:recommendsContext [ rdfs:comment "Converts a natural language goal into
        ↪  move(d) or rotate(a)." ] ;
6       hmas:signifies [ td:hasForm [ http:methodName "POST" ;
7                       hctl:forContentType "application/json" ;
8                       hctl:hasTarget <http://localhost:9997> ] ;
9               td:hasInputSchema [ a js:ObjectSchema ;
10                      js:additionalProperties false ;
11                      js:properties [ a js:NullSchema,
12                                  js:NumberSchema,
13                                  js:StringSchema ;
14                              js:propertyName "id" ],
15                          [ a js:ObjectSchema ;
16                              js:additionalProperties false ;
17                              js:properties [ a js:ObjectSchema ;
18                                      js:additionalProperties false ;
19                                      js:properties [ a js:StringSchema ;
20                                              rdfs:comment "A string that identifies the
                                                ↪  message" ;
21                                              js:propertyName "messageId" ],
22                                          [ a js:ArraySchema ;
23                                              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." ;
24                                              js:maxItems 1 ;
25                                              js:minItems 1 ;
26                                              js:propertyName "parts" ],
27                                          [ a js:StringSchema ;
28                                              js:const "user" ;
29                                              js:propertyName "role" ] ;
30                                      js:propertyName "message" ;
31                                      js:required "parts",
32                                          "role" ] ;
33                              js:propertyName "params" ;
34                              js:required "message" ],
35                          [ a js:StringSchema ;
36                              js:const "2.0" ;
37                              js:propertyName "jsonrpc" ],
38                          [ a js:StringSchema ;
39                              js:const "message/send" ;
40                              js:propertyName "method" ] ;
41                      js:required "id",
42                          "jsonrpc",
43                          "method",
44                          "params" ] ] .
45
```

Listing 9: Signifier for A2A agent

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.

```
1
2
3  <#sig> a hmas:Signifier ;
4      rdfs:label "cherrybot_utcp_operation" ;
5      hmas:recommendsContext [ rdfs:comment "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)" ] ;
6      hmas:signifies [ td:hasForm [ http:methodName "POST" ;
7                      hctl:forContentType "text/plain" ;
8                      hctl:hasTarget <http://localhost:8086/operation> ] ;
9              td:hasInputSchema [ a js:StringSchema ;
10                     js:description "Operation command as plain text, e.g. move(10)." ]
                       ↪  ] .
```

Listing 10: Signifier for UTCP tool

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

```
1
2
3  <#sig> a hmas:Signifier ;
4      rdfs:label "cherrybot_proxy_operation" ;
5      hmas:recommendsContext [ rdfs:comment "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)" ] ;
6      hmas:signifies [ td:hasForm [ http:methodName "POST" ;
7                      hctl:forContentType "text/plain" ;
8                      hctl:hasOperationType "invokeaction" ;
9                      hctl:hasTarget <http://localhost:8086/operation> ] ;
10             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”.

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

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

Robot Control: The context is: "The agent wants to control a robot from a formal goal." 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.