Enabling BDI Agents to Reason on a Dynamic Action Repertoire in Hypermedia Environments

Danai Vachtsevanou
University of St.Gallen
St. Gallen, Switzerland
danai.vachtsevanou@unisg.ch

Bruno de Lima
Federal University of Santa Catarina
Florianópolis, Brazil
bruno.szdl@gmail.com

Andrei Ciortea
University of St.Gallen
St. Gallen, Switzerland
andrei.ciortea@unisg.ch

Jomi Fred Hübner
Federal University of Santa Catarina
Florianópolis, Brazil
jomi.hubner@ufsc.br

Simon Mayer
University of St.Gallen
St. Gallen, Switzerland
simon.mayer@unisg.ch

Jérémy Lemée
University of St.Gallen
St. Gallen, Switzerland
jeremy.lemee@unisg.ch

ABSTRACT

Autonomy requires adaptability and persistence in pursuing long-term objectives within evolving contexts. While BDI agents can cope with dynamic and uncertain environments, their adaptability is typically constrained by static action repertoires, known *a priori* by designers. Through Semantic Web and Web of Things technologies, machine-readable action descriptions can be discovered in hypermedia environments, and updated dynamically to mirror the evolving landscape of actions offered by real-world environments. This paper proposes the integration of action-oriented BDI reasoning with signifiers that reveal information about action possibilities that may appear, disappear, or be modified in a hypermedia environment at any time. We extend the means-end reasoning of BDI agents with a mechanism for resolving signifiers discovered at run time into actions, which enables agents to adjust their decision-making based on action possibilities advertised in the environment. We evaluate our approach through experiments, where Jason agents discover signifiers expressed with available Web ontologies. The results demonstrate that our signifier resolution mechanism enhances action reasoning at run time towards effective goal achievement in dynamic and unknown environments.

KEYWORDS

Semantic Web; Belief-Desire-Intention agents; Practical reasoning

ACM Reference Format:
Danai Vachtsevanou, Bruno de Lima, Andrei Ciortea, Jomi Fred Hübner, Simon Mayer, and Jérémy Lemée. 2024. Enabling BDI Agents to Reason on a Dynamic Action Repertoire in Hypermedia Environments. In *Proc. of the 23rd International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2024), Auckland, New Zealand, May 6 – 10, 2024*, IFAAMAS, 9 pages.

1 INTRODUCTION

Autonomous agents have the ability to deliberate about goals and to make decisions towards achieving their design objectives without human intervention. However, the complexity of these tasks increases when agents have to deal with dynamic, shared, and uncertain environments. To address this challenge, BDI agents employ a *sense-reason-act* cycle, which enables balancing between proactive and reactive behavior. For example, an autonomous vacuum cleaner can continuously strive to maintain a room clean through plans that enable it to move around and clean dirt. Yet, this proactive behavior is adapted in reaction to changes in the environment, such as the accumulation of dirt in specific areas.

In open environments, and especially in Internet of Things or Web of Things (WoT) [18] environments, agents may encounter a plethora of sensors and actuators. These are often part of the environment rather than an agent's body (e.g., a robot's body), and not necessarily known at design time; hence, agents are required to discover such devices at run time, and may then use them as tools to achieve their goals. When an agent joins such an unknown and possibly dynamic environment, challenges arise: Consider, for example, a personal assistant agent deployed in a smart home environment, where the agent is tasked to assist the user in maintaining a comfortable environment. While the agent might have the procedural knowledge to increase room illuminance by turning on a lamp, it might lack information regarding available lamps in the environment, the type of inputs accepted by these lamps (numeric, RGB, etc.), or whether lamp control conflicts with current user preferences. Here, it would be beneficial if the agent could dynamically acquire such interaction-related information—and if that information then could inform the agent's reasoning upon selecting or resolving plans based on the current agent-environment situation.

To bridge this gap between agents' reasoning and the intricacies of modern environments, hypermedia formats can be used to describe the action possibilities offered by the agents' environment [10]. Such descriptions can be discovered by agents at run time in a hypermedia environment, and can signify to the agents how to interact with previously unknown environmental entities through hypermedia controls [34]. Although this approach of "advertising" actions in hypermedia environments is increasingly adopted for action discovery and exploitation by agents (cf. [1, 9, 10, 22, 25, 30, 34, 35]), it has primarily been integrated to facilitate classical planning (e.g., in [9, 26, 35]), or to fill in interaction details *after* reasoning (i.e., during action execution; e.g., in [3, 10]). A challenge that remains is to integrate information about available means discovered at run time with the flexible

Proc. of the 23rd International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2024), N. Alechina, V. Dignum, M. Dastani, J.S. Sichman (eds.), May 6 – 10, 2024, Auckland, New Zealand. © 2024 International Foundation for Autonomous Agents and Multiagent Systems (www.ifaamas.org).

means-end reasoning methods of BDI agents. This entails incorporating interaction-related information into the decision-making process *before* selecting an *existing* plan, for example, to ensure that a plan for increasing room illuminance is used with available and contextually-appropriate actions based on the run-time situation.

In this paper, we introduce a method for resolving abstract actions specified in pre-designed plans into concrete actions available within the agent's run-time environment and advertised through interaction cues (i.e., signifiers [25]). Further, we integrate this resolution mechanism in the means-end reasoning process of BDI agents, so that agents can determine whether their pre-defined plans are applicable within the context of the dynamic environment they encounter; this allows agents to execute plans based on action possibilities they discover in the environment at run time. As BDI reasoning does not inherently involve reasoning directly on actions, we explain how our resolution mechanism relates to reasoning about intentions as well as to action reasoning processes typically employed in a framework-specific scope (e.g., in 3APL, Jason, etc.). We evaluate our approach through the deployment of a system in which a Jason agent can discover and reason about signifiers that reveal different information about possible actions across a series of experiments. Our evaluation shows how incorporating interaction cues discovered at run time into the agents' reasoning processes enables more informed decisions in dynamic and unknown environments.

2 RELATED WORK

In this section, we first examine related work on action reasoning and execution in BDI agents to extract requirements for action representation (Sect. 2.1). Next, we examine approaches to action representation in hypermedia environments, which may provide insight into how to design interaction cues that enable BDI reasoning in dynamic environments (Sect. 2.2).

2.1 Reasoning Towards Action with Beliefs, Desires, and Intentions

In Agent-Oriented Software Engineering, *actions* are defined as the essential means by which agents can affect their environment [40]. In this context, agent programs use actions that are typically implemented by invoking functions, embedded within more complex constructs like plans. Consequently, within BDI frameworks, action invocation can be minimally specified using a function identifier representing the action type (e.g., `move`, `toggle`) and optionally arguments, typically variables instantiated at run time (e.g., `100` and `Y` in `move(100, Y)`). To permit the effective invocation of an action, an agent needs to know at least the action identifier together with the data types of the action's arguments; these aspects hence constitute the minimal requirements when designing discoverable interaction cues for agents. Ideally, the semantics of action and argument identifiers should also reflect the action and argument types enabling intuitive programming of action invocation.

Besides revealing information necessary for *action invocation*, interaction cues should also enable BDI agents to *reason about action* to ensure that acting is contextually appropriate based on *run-time conditions*. In the BDI architecture, an agent's actions stem from its current *intentions*. Thus, the conditions governing action invocation are subsumed by the conditions governing the agent's decisions to adopt or retain intentions, i.e., to start or continue working towards specific goals through a specific course of action [42]. For example, in Jason agents, intentions are represented as *plans* that the agent has committed to perform [4, 5]. The selection of a plan hinges on its applicability in the current system state, with applicability conditions encapsulated in the plan's *application context*. Therefore, action invocation occurs when the application context aligns with the agent's beliefs. BDI frameworks facilitate reasoning about action invocation based on the contextual relevance of composite activities, e.g., through *application contexts* of Jason plans, *conditional operators* of 3APL plans [15], *guards* of GWENDOLEN plans [16], or *contexts* of GOAL modules [19][1].

Some BDI frameworks go beyond the basic requirements of a BDI architecture [42], introducing additional conditions for advanced reasoning about action invocation, action success, and belief revision. These conditions, typically defined within action descriptions in terms of *preconditions* and *postconditions* (e.g., in [15, 19, 31]), exhibit variations in semantics and utility across frameworks. For example, both 3APL and GOAL formalize action preconditions to assess action availability in the run-time environment, contingent on alignment with agent beliefs. However, 3APL considers preconditions for controlling the invocation of only a specific action type (namely *capabilities*), and GOAL combines preconditions with other conditions in action rules for action control. In GWENDOLEN, action preconditions are not used for action control at all; instead, they are used along with action postconditions for plan reconfiguration via automated planning [33]. GWENDOLEN postconditions also contribute to evaluating action success based on whether they align with agent beliefs post-action. Action postconditions in 3APL and GOAL are used to update agent beliefs for accurate reflection of the expected system state after action execution.

Finally, action reasoning and invocation lead to the execution of code that implements the action based on details of the deployment context [38] (e.g., based the implementation of `move(X,Y)`). These details typically exceed the scope of agent programming languages, forming an integral part of the environment's implementation. Additionally, they are commonly considered *to be known* during design time, and *stable* during run time. This design approach is reasonable for many BDI applications that assume a static environment, one that is expected to remain unaltered except for the actions performed by agents [42]. The approach is further reinforced by the guidance provided to agent designers, who "should know precisely what the environment is like, and what are the things agents need to perceive from it so as to act appropriately, as well as what are the actions that agents will be able to perform so as to change the environment" [5]. This is facilitated by the fact that agent designers often take on the responsibility of designing and implementing the environment themselves: "[O]nce [designers] have a stable implementation of the environment, [they] could then start implementing the agents" [5]. However, this reasoning does not apply if artifacts are dynamically deployed in open environments, especially when artifact designers are independent from agent designers, as seen in highly distributed setups like the Web. In such cases, which are

[1]This is also the case in GOAL action rules; however, an action rule is typically used only for single actions, which is why we do not mention this in our list.

gaining importance in the MAS literature [2, 6, 12, 29, 34, 36], it becomes imperative for BDI agent design to be agnostic to specific implementation details of artifacts in the environment.

In summary, acting in the environment requires access to information that allows for action invocation, reasoning, and lower-level execution. In dynamic environments, interactions cues should convey all such necessary information at run time. Cues should be generic to accommodate various BDI agents while also being adaptable to evolving framework-specific requirements.

2.2 Representing Actions in Hypermedia Environments

Representing actions for BDI agents in open and dynamic environments can be informed by other research areas. An exemplary case is the Web, where actions are provided to users by a *hypermedia environment* [37]. Hypermedia environments inherently support the dynamic discovery and exploitation of actions, where actions are represented via hypermedia formats to support invocation, reasoning, and execution. For example, actions represented in HTML pages through hyperlinks and forms allow human users to invoke actions (and provide input data) using dynamically rendered controls. The controls typically convey semantic information (e.g., through labels, instructions, or the context in which they are embedded), which allows humans to reason about actions while navigating the Web. Such action representations also contain metadata that allows Web browsers to take care of lower-level implementation details, such as executing HTTP requests based on the hypermedia controls.

To enable software agents to discover and exploit actions in hypermedia environments, various interaction models and machine-readable hypermedia formats have been proposed [1, 13, 21, 24, 35]. For instance, *Hydra* [24] defines a vocabulary for representing hypermedia actions exposed by Web APIs, enabling the discovery of implementation details required for interacting with Web resources through hypermedia controls. Taking a step further, the *Web of Things* (WoT) initiative [39] introduces action descriptions that use a hypermedia control vocabulary, but also encompass concepts for conveying action-related information at a higher level of abstraction. Specifically, the W3C WoT Thing Description (TD) Recommendation [21] employs the HCTL vocabulary [7] for describing hypermedia controls, and provides a minimal vocabulary for *higher-level action types* that carry semantics for interactions with *WoT Things*, namely actions, properties, and events. For example, a *TD Action Affordance* defines metadata about how to affect the state of a WoT Thing, including references to input schemas. The Recommendation also gives examples of how such metadata can be enriched with semantics stemming from third-party vocabularies to allow for informed interactions within specific domains (e.g., in manufacturing, smart home applications etc.). As a result, Action Affordances become valuable interaction cues that can be dynamically discovered to inform agents about how to act on physical or virtual entities that are described through the W3C WoT TD.

In this work, we aim to leverage the insights derived from representing actions in hypermedia environments. Our goal is to transfer such insights to the design of interaction cues for hypermedia actions, while ensuring that these cues remain aligned to the requirements of BDI reasoning. This necessitates the development of a generalized approach to the representation of hypermedia actions that subsumes the scope of the WoT. Additionally, the approach should allow for *extensibility* when required to *specialize* action metadata to heterogeneous abilities of agents, such as variations in the reasoning skills of BDI agents. We argue that achieving a *general approach to this extensibility and specialization* requires formally representing the tailoring of interaction cues to autonomous agents with (similar yet) different abilities, so as to support the effective discovery and interpretation of action possibilities even in larger-scale, open, and dynamic multi-agent systems.

3 RESOLVING SIGNIFIERS IN BDI REASONING

In this section, we introduce a method for resolving interaction cues (i.e., signifiers), to possible actions during means-end reasoning. Our aim is to support BDI agents to effectively reason about and take action in unknown, dynamic environments. In Sect. 3.1, we first examine how signifiers allow agents to *perceive action possibilities* in hypermedia environments. In Sect. 3.2, we then propose a method that uses signifiers to *resolve abstract actions* programmed at design time to concrete actions available in the environment at run time.

3.1 Action Perception in Dynamic Hypermedia Environments

Action perception is the ability of autonomous agents to sense and interpret percepts that relate to action possibilities offered by environmental artifacts. Such percepts serve as interaction cues that can provide agents with information about the available actions—for example, their parameters, contextual relevance, and implementation specifics (see Sect. 2.1). To facilitate action perception of BDI agents in dynamic hypermedia environments, we use *signifiers* to model and represent interaction cues, as introduced in [34]. The term emphasizes the fulfillment of a set of requirements, which align with the primary properties and role of our interaction cues: being easily discoverable and meaningful to agents, and revealing information about behaviors that can be performed in the environment. Specifically, signifiers should reveal all necessary information to meet the requirements of BDI-based action invocation, reasoning, and execution (see Sect. 2.1). Furthermore, they should conform to the modeling and formatting standards governing the representation of hypermedia actions (see Sect. 2.2) to enable their dynamic publication, discovery, and usage. In the following, we progressively examine the design of signifiers for BDI agents.

3.1.1 Describing Actions and Action Parameters. Our objective is to facilitate the provisioning of actions by both physical and virtual artifacts (such as devices, digital services, knowledge repositories, Web pages, etc.), where artifacts and their offered actions may become available or unavailable at run time. To permit this, we design signifiers as machine-readable information resources (here, in RDF [41]). This approach enables two key functionalities: Firstly, the independent publication and modification of signifiers by different designers, and potentially, software agents; secondly, the discovery and exploitation of signifiers by autonomous agents at run time, e.g., by crawling the hypermedia environment, accessing relevant repositories, sharing signifiers, or using hypermedia search engines (cf. [3]). To support this approach, we employ the signifier model presented in [34] and represent them using an ontology

Listing 1: An interaction cue in a resource profile, signifying a specification of toggling a lamp as a WoT TD Affordance.

```
 1 @base <http://ex.org/wksp/1/arts/1> .
 2 ...
 7 <> a hmas:ResourceProfile ;
 8   hmas:isProfileOf <#artifact> ;
 9   hmas:exposesSignifier <#togglable-signifier>.
10
11 <#artifact> a td:Thing, saref:Lamp ;
12   td:title "lamp" ;
13   td:hasSecurityConfiguration [
14     a wotsec:NoSecurityScheme ] ;
15   td:hasActionAffordance <#toggle-spec> .
16
17 <#togglable-signifier> a hmas:Signifier ;
18   hmas:signifies <#toggle-spec> .
19
20 <#toggle-spec> a td:ActionAffordance,
21   saref:ToggleCommand ;
22   td:name "toggle";
23   td:hasInputSchema <#input-constraints> .
24
25 <#input-constraints> a js:ObjectSchema,
26       saref:OnOffState ;
27   js:required "lampState" ;
28   js:properties [ a js:IntegerSchema ;
29     js:enum 0 ;
30     js:propertyName "lampState" ] .
31
32 <#toggle-spec> td:hasForm <#form> .
33
34 <#form> a hctl:Form ;
35   hctl:hasTarget <https://lamp.example.org/state> ;
36   htv:methodName "PUT" .
```

for Hypermedia Multi-Agent Systems (hMAS) [10, 11]. Our goal is to reuse the concepts introduced with that model as a means to *generalize* descriptions of action possibilities in hypermedia environments, and specifically consider such descriptions for enabling the representation, discovery, and use of signifiers for BDI agents.

First, we consider that signifiers can be published and discovered within a *resource profile* [34] associated with the artifact offering the represented actions. For example, we conceive of the W3C WoT Thing Description (TD) as a type of resource profile. Then, if an artifact can be modeled as a WoT Thing, its related signifiers can be exposed in a resource profile modeled as a WoT TD document (see Sect. 2.2). There, TD Action Affordances describe how agents can change the artifact's state. This metadata serves as an action specification that conveys all essential information for defining the invocation of an action when programming an agent's plan. As an example, Lst. 1 represents the profile of a lamp artifact modeled as a WoT Thing. This profile provides an action specification of toggling the lamp as a TD Action Affordance (Lines 20-30). Lines 25-30 define constraints imposed on the input required for invoking this action. These constraints are represented in JSON Schema [8], which can be used to automatically validate action invocation parameters. The schema in Lst. 1 specifies that the input should correspond to a JSON object with a field "lampState" of value 0.

The generalization of WoT TDs to hMAS resource profiles allows for the alternative representation of input constraints through SHACL shapes [23]. SHACL shapes permit representing and validating constraints applied to RDF graphs, while remaining *agnostic to specific serialization formats*. As a result, SHACL emerges as a natural choice for expressing constraints in RDF—including constraints that are related to required input parameters within signified action specifications. Lst. 2 represents the profile of the same lamp artifact as Lst. 1, but constraints on action invocation are now captured in an action specification as a SHACL shape [23] (Lines 16-29).

Listing 2: An interaction cue in a resource profile, signifying a specification of toggling a lamp as a SHACL shape.

```
 1 @base <http://ex.org/wksp/1/arts/1> .
 2 ...
 7 <> a hmas:ResourceProfile ;
 8    hmas:isProfileOf <#artifact> ;
 9    hmas:exposesSignifier <#togglable-signifier> .
10
11 <#artifact> a hmas:Artifact, saref:Lamp .
12
13 <#togglable-signifier> a hmas:Signifier ;
14   hmas:signifies <#toggle-spec> .
15
16 <#toggle-spec> a sh:NodeShape ;
17   sh:class hmas:ActionExecution, saref:ToggleCommand ;
18   sh:property [ sh:path hmas:hasInput ;
19     sh:qualifiedValueShape <#input-constraints> ;
20     sh:qualifiedMinCount 1 ;
21     sh:qualifiedMaxCount 1 ] .
22
23 <#input-constraints> a sh:NodeShape ;
24   sh:class saref:OnOffState ;
25   sh:property [ sh:path saref:hasValue ;
26     sh:minCount 1 ;
27     sh:maxCount 1 ;
28     sh:hasValue 0 ;
29     sh:name "lampState" ] .
30
31 <#toggle-spec> sh:property [ sh:path prov:used ;
32     sh:minCount 1 ;
33     sh:maxCount 1 ;
34     sh:hasValue <#form> ] .
35
36 <#form> a hctl:Form ;
37   hctl:hasTarget <https://lamp.example.org/state> ;
38   htv:methodName "PUT" .
```

Finally, in both Listings 1 and 2, action specifications are annotated with domain-specific semantics that convey the semantic type of the action and the semantic types of the action's expected parameters. Specifically, the two listings reveal information about the *ToggleCommand* action that affects the *OnOffState* of a *Lamp*, using the Smart Applications Reference (SAREF) vocabulary [14].

3.1.2 Describing the Contextual Relevance of Actions. Some BDI frameworks use more descriptive representations that go beyond enabling action invocation, and rather enable performing reasoning about action *in context*. Action representations with contextual information aim to explicitly establish the relationship between actions and specific system states, including both the state of the acting agent and of its environment. To accommodate BDI approaches that explicitly consider action descriptions (see Sect. 2.1), we can follow the WoT TD approach and make use of framework-specific vocabularies to represent contextual information along with their framework-specific semantics. As an illustration, we can extend both Listings 1 and 2 with the metadata in Lst. 3 to employ a vocabulary specific to a BDI framework such as 3APL. The extended signifier reveals information about a 3APL agent's capability to toggle (Lines 47-49): toggling necessitates that the room is empty of other agents (indicated by the 3APL precondition `room(empty)`), and results in decreasing the illuminance in the room (indicated by the 3APL postcondition `room(decreased_illuminance)`).

Listing 3: Interaction cue extension that reveals the contextual relevance of toggling a lamp with a 3APL vocabulary.

```
 1 ...
44 <#togglable-signifier> hmas:recommendsAbility [
45   a apl3:ThreeAPLReasoner ] .
46
47 <#toggle-spec> sh:class apl3:Capability ;
48   apl3:hasPrecondition "room(empty)" ;
49   apl3:hasPostcondition "room(decreased_illuminance)" .
```

Although signifiers that use third-party vocabularies (e.g., vocabularies for 3APL, Jason, GOAL, and GWENDOLEN agents) can be effectively interpreted by their intended agents, they do not formally establish their relevance to a specific agent type. Consequently, such signifiers do not facilitate their *intuitive discovery* by the intended agents, or their exclusion by agents for whom the signifiers are inappropriate and should remain out of scope. We hence consider that signifiers may explicitly refer to *abilities* [34] that agents are recommended to have for effectively using the signified interaction metadata. An example of such recommended abilities is illustrated in Lst. 3, where it is explicitly stated that the represented signifier is recommended for use by 3APL agents (Lines 44-45).

The explicit recommendation of abilities does not impose restrictions on other types of BDI agents, since other agents may use action descriptions while disregarding those semantic annotations that they cannot interpret. However, such recommendation can assist agents in *directing their action perception* towards signifiers that might be more meaningful to them. This is particularly beneficial in large-scale environments that feature a large number of available signifiers and heterogeneous agents. Furthermore, recommended abilities induce opportunities for driving agent evolution as well as agent-to-agent collaboration: When agents encounter a signifier that does not align with their abilities, they may seek for a means to exhibit such abilities using external services and tooling (e.g., for vocabulary alignment [17]), or they may interact with other agents that possess the recommended abilities to request the performance of the signified action or to delegate relevant goals to these agents.

Lastly, we introduce an additional option that allows for the representation of actions' contextual relevance that is *more general* than the one supported by WoT TDs. To achieve this, we employ the concept of *recommended context* [34] which offers a uniform means for conveying contextual metadata. In the case of BDI agents, such action metadata can facilitate the reasoning about action based on common abstractions such as beliefs, desires, and intentions. As an example, Lst. 4 represents metadata for the toggling action and specifies the constraints that must be satisfied in the run-time situation for the reasoned invocation of toggling. The metadata can be used to extend Listings 1 and 2; however, contrary to Lst. 3, it avoids using any framework-specific concepts. Instead, the extended signifier recommends that toggling should be performed by an agent if the agent *believes* that the room is empty of other agents and *desires* to decrease the illuminance in the room. In this case, agents require no additional abilities for meaningfully interpreting the signifier other than being capable of logic-based BDI reasoning[2], as indicated by the recommended ability in Lines 44-45 of Lst. 4.

[2]The representation for BDI concepts in Lst. 4 is relevant to logic-based BDI agents. SHACL and additional ontologies could be used to support other types of agents, such as Jadex agents [27] whose beliefs are represented using common Java data types.

Listing 4: Interaction cue extension that reveals the contextual relevance of toggling a lamp with a generic BDI vocabulary.

```
 1 ...
44 <#togglable-signifier> hmas:recommendsAbility [
45   a bdi:LogicBDIReasoner ] ;
46   hmas:recommendsContext <#empty-room-constraints> .
47
48 <#empty-room-constraints> a sh:NodeShape ;
49   sh:class hmas:ResourceProfile ;
50   sh:property [ sh:path hmas:isProfileOf ;
51     sh:qualifiedValueShape <#agent-constraints> ;
52     sh:qualifiedMinCount "1"^^xs:int ] .
53
54 <#agent-constraints> a sh:NodeShape ;
55   sh:class hmas:Agent ;
56   sh:property [ sh:path bdi:hasBelief ;
57     sh:minCount "1"^^xs:int;
58     sh:hasValue "room(empty)" ] ;
59   sh:property [ sh:path  bdi:hasDesire ;
60     sh:minCount "1"^^xs:int;
61     sh:hasValue "room(decreased_illuminance)" ] .
```

Adopting a uniform way for capturing the contextual relevance of actions promotes the interoperability of agents over action possibilities, and the reusability of interaction cues. Even if agents are designed based on different methodologies, they can effectively interact in an open environment, and exchange information about action in a shared vocabulary. Importantly, they still maintain the freedom to exploit the advertised information according to their individual approaches during action reasoning. For instance, upon encountering the signifier extended by Lst. 4, a Jason agent could use the recommended belief (Lines 56-58) to resolve the application context of a plan that involves toggling, a 3APL or GOAL agent could employ it to control the invocation of toggling, and a GWENDOLEN agent might consider it during plan patching (cf. [32]). All agents could also choose to ignore the contextual metadata, since *recommendation* is used here to provide interaction guidance but without regimenting behavior or limiting autonomy.

In summary, our approach allows designers of BDI agents to uniformly capture contextual metadata at their preferred abstraction level. This level is realized through the distribution of used metadata across shared or more specialized vocabularies, which in turn affects the balance between interoperability and performance in action reasoning. In all cases, the relevance of signifiers to their target agents is explicitly captured through recommended abilities to permit the intuitive discovery and interpretation of signifiers.

3.1.3 Describing Action Implementation Details. For enabling the execution of signified actions, that follows action reasoning and invocation (see Sect. 2.1), signifiers should convey information about implementation details. In our approach, we make use of the HCTL vocabulary [7] for representing hypermedia controls, which is used by the WoT TD and is aligned with the Hydra model[3]. All discussed listings feature the same hypermedia controls as an HCTL Form (see Lines 32-36 and 31-38 in Listings 1 and 2, respectively). This form can be used to construct an HTTP request to execute the toggling of the lamp. Hypermedia controls serve as low-level interaction instructions, which can be left outside the scope of agent programs and agents' reasoning cycles. Instead, these controls

[3]Details about the alignment of HCTL, WoT TD, and Hydra are available in [7].

Listing 5: A plan that represents the knowledge about an abstract action provided at design time.

```
1   +!room(decreased_illuminance) : true
2           <- invokeAction("saref:ToggleCommand", ["saref:OnOffState"], [0]) ;
3              ?room(decreased_illuminance) .
```

can be handled by external entities responsible for implementing the actions invoked by agents. For instance, in MAS that follow the Agents & Artifacts meta-model [4], implementation details can be managed by a local artifact tasked with executing action implementations on behalf of agents (e.g., as demonstrated in [10]), similar to how a Web browser functions for human agents.

3.2 Action Resolution Through Signifiers

We next examine how agents that dynamically perceive signifiers, such as the ones in Sect. 3.1, can exploit signified information for action. To this end, we introduce a *Signifier Resolution Mechanism (SRM)*: An SRM is the mechanism of an agent that resolves the agent's knowledge about abstract actions to possibilities of concrete actions at run time. For BDI agents, the SRM is part of the agents' means-end reasoning process, that resolves abstract actions in plans (or action rules and other similar structures) to concrete actions available for invocation, reasoning, and execution at run time.

Given the diversity of frameworks for BDI agents, SRMs may vary: The variation may stem from framework-specific sub-processes for means-end reasoning, their formalization using specialized concepts or other factors. In what follows, we introduce SRM functionality based on concepts inspired by Jason and AgentSpeak [5], but this does not restrict the generality of our approach—and, where applicable, we draw parallels to concepts with similar semantics and utility in other BDI models and frameworks.

We consider that abstract actions are defined within *plans*, which represent means that enable agents to execute a simple or complex series of actions. In Lst. 5, we use a simple plan structure to show how signifiers can be used for resolution, considering that this structure is expressive enough to capture the details of action metadata discussed in Sect. 3.1. Plans, alongside analogous concepts like action rules or activities, play a fundamental role in reasoning about the best approach to pursue an intention. The intention relevant to the plan presented in Lst. 5 is to decrease room illuminance (see Line 1, where `+!` denotes the addition of this achievement goal event in AgentSpeak [28]). This is a single-action plan whose *application context* subsumes any conditions that must hold at run time to invoke the action (see `true` in Line 1). These conditions may include guards, preconditions, conventional programming conditionals, etc. Finally, to address any conditions that are expected to hold post-action, the plan is declaratively programmed by adding a test goal after action invocation (see Line 3, denoted by the AgentSpeak test operator ?). This caters to approaches where action postconditions should be verified against the agent's beliefs to evaluate action success (via the test goal addition) as well as approaches where postconditions need to be added in the belief base by the agent itself (via plan execution for addressing the test goal addition such as `+?room(decreased_illuminance) <- +room(decreased_illuminance)`). Finally, the plan involves the invocation of a command, that is compatible to `ToggleCommand` with an `OnOffState` in the SAREF ontology, on a source of illumination (see Line 2).

At run time, the agent's SRM is responsible for the resolution of such knowledge about abstract actions captured in plans to possibilities of concrete actions as revealed by discovered signifiers. For example, the abstract action presented in Lst. 5 can be resolved to a concrete action based on the signifier in Lst. 1 or Lst. 2. Resolution by an SRM is achieved based on the following reasoning tasks:

- If a signifier signifies a specification of the action type that appears in an action invocation of a plan, then the action is considered available for execution through the hypermedia controls defined in the specification. For example, upon perception of the signifier in Lst. 1 or Lst. 2, the SAREF `ToggleCommand` action in the plan of Lst. 5 is considered available for execution based on the form in the signifier.
- If a signifier recommends an ability, then it is considered relevant for perception by an agent that exhibits this ability. For example, the signifier extension in Lst. 3 can be considered relevant to 3APL agents, while the signifier extension in Lst. 4 can be considered relevant to all BDI agents.
- If a signifier recommends a context that applies constraints on the agent's beliefs, then the application context of the plan is resolved to capture this recommended context, which should additionally hold for considering that the action is available. For example, upon perception of the signifier extended with Lst. 4, the application context of the plan in Lst. 5 (`true`) is resolved to include the literal `room(empty)`.
- If a signifier recommends a context that applies constraints on the agent's desires, then the evaluation conditions of the plan are extended to capture this recommended context. For example, upon perception of the signifier extended with Lst. 4, the test goal `?room(decreased_illuminance)` in Lst. 5 could be achieved through the plan `+?room(decreased_illuminance) <- +room(decreased_illuminance)`.

These reasoning tasks can be implemented using an SRM that integrates with the BDI reasoning cycle. Here, we use signifier resolution to enhance agents' means selection function [42], that is the function that an agent uses to decide *how* to achieve its intentions, such as by selecting a plan. Algorithm 1 presents the functioning of an SRM, which serves as an extension of the means selection function for BDI agents.[4] It assumes access to the agent's native means selection function's output, represented as a set of plans P, as well as access to the agent's beliefs stored in the set B and the agent's abilities in the set Ab. Additionally, we assume that the perceived signifiers are stored in the set Sig, from which we can extract the signified actions of the set A_{Sig} with their related set of recommended abilities Ab_{Sig}, and recommended context C_{Sig}.[5]

The algorithm evaluates the run-time availability of means in set P by verifying if their abstract actions can be resolved using signifiers in Sig. It iterates through each plan $\pi \in P$ and, and checks if, for all the abstract actions A_π of π, there exist signifiers with matching signified actions ($A_\pi \subset A_{Sig}$), while excluding any signifier whose recommended context does not align with the agent's beliefs ($B \not\models C_{Sig}$), or whose recommended abilities are not included

[4]Therefore, Algorithm 1 is not concerned with action evaluation conditions as it is only concerned with means-end reasoning, which precedes action invocation.
[5]The notation is slightly refined from the signifier model in [34] to emphasize the utility of the elements in the context of an SRM.

Algorithm 1 Algorithm for the Signifier Resolution Mechanism that extends the BDI reasoning cycle's means selection function.

```
$Sig \leftarrow$ perceived signifiers, $P \leftarrow$ relevant plans, $B \leftarrow$ beliefs, $Ab \leftarrow$ set
of agent's abilities, $A_{Sig} \leftarrow$ set of actions signified by $Sig$
for each $\pi \in P$ do
    $A_\pi \leftarrow$ set of actions of $\pi$
    if $A_\pi \subset A_{Sig}$ then
        $MeansAvailable \leftarrow$ true
        for each $\alpha \in A_\pi$ do
            $C_{Sig} \leftarrow$ recommended context of signifier signifying $\alpha$
            if $B \not\models C_{Sig}$ then
                $MeansAvailable \leftarrow$ false
                break
            end if
            $Ab_{Sig} \leftarrow$ recommended abilities set of signifier signifying $\alpha$
            if $Ab_{Sig} \not\subseteq Ab$ then
                $MeansAvailable \leftarrow$ false
                break
            end if
        end for
        if $MeansAvailable$ then
            return $\pi$
        end if
    end if
end for
return null
```

in the agent's abilities ($Ab_{Sig} \not\subseteq Ab$). If there are relevant signifiers for all abstract actions in a set A_π, then the plan π is returned as the output of the means selection function. If no such set is found, the agent infers that it currently lacks the means to achieve its goals.

4 IMPLEMENTATION AND EVALUATION

To ground our approach, we implement and evaluate the use of signifiers and an SRM as part of the Jason reasoning cycle [4, 5].[6] Specifically, we implemented Algorithm 1 as an extension of the means selection function of Jason agents. For creating and processing signifiers, we developed a Java library[7] that manages signifiers based on the hMAS ontology. All agents used in our evaluation are deployed within a JaCaMo application.[8]

In our evaluation, we test four BDI agents (A, B, C, and D) in four different scenarios to assess signifier resolution effectiveness in a variety of run-time situations. While Agent A relies solely on the regular Jason reasoning cycle, Agents B, C, and D use different variants of an SRM within their Jason reasoning cycle. The SRM enables agents to consider different interaction metadata when determining plan applicability and actions to invoke:

- Agent A: Lacking an SRM, it executes the first relevant plan, potentially resulting in errors if actions are unavailable.
- Agent B: With a basic SRM, it prioritizes action availability, opting for alternative plans when actions are unavailable.
- Agent C: Building on Agent B, it also considers context alignment, invoking actions that match the recommended context.
- Agent D: This encompasses all prior SRM features and invokes an action only if its abilities align with the abilities recommended by the corresponding signifier.

Table 1: Features of the different SRM versions employed by BDI agents in our evaluation.

SRM Features	*A*	*B*	*C*	*D*
Signified Actions	✗	✓	✓	✓
Recommended Context	✗	✗	✓	✓
Recommended Abilities	✗	✗	✗	✓

Table 1 summarizes the evaluated SRM features and their usage by agents in our scenarios. In all scenarios, the performance of agents is assessed based on two distinct types of objectives: *Agent Objectives* and *Design Objectives*. Agent Objectives are used to evaluate an agent's ability to effectively behave rationally with respect to its goals. Design Objectives are used to evaluate an agent's ability to behave based on the designer's intention for the agent role within an environment. As an illustration, consider a laboratory assistant agent entering an unfamiliar lab environment. The agent is equipped with plans to perform tasks for achieving its (Agent) Objectives, like adjusting the lighting and operating lab equipment. However, the lab environment it enters may have been designed with specific Design Objectives in mind, which could either coincide with, overlap, or conflict with the agent's objectives. For instance, the agent's primary goal may be to create a comfortable working condition and it might hence increase room illuminance. However, the lab's designer may have intended that the lab conserves energy, hence striving to keep the lights off in unoccupied areas. If the agent increases the lighting while no people are present, it fulfills its Agent Objective but fails to meet the Design Objective.

In our evaluation, the four agents are subjected to four scenarios that incrementally increase in complexity. Each scenario takes place in a room that contains environmental artifacts, including a lamp, window blinds, and robotic arms, which deployed agents use to achieve their overarching objective of conserving energy.

Scenario 1: Basic Plan Applicability. In this baseline scenario, the environment is static, providing agents with all the necessary artifacts for their predefined plans. The performance of agents is evaluated on the basis of achieving both an Agent Objective and a Design Objective, both aimed at reducing room illuminance. Each agent has two plans for achieving this: toggling the lamp or lowering the window blinds. All agents achieve the objectives using either plan because the required artifacts are always available.

Scenario 2: Plan Applicability based on Action Availability. In this scenario, all agents continue to have the (Agent and Design) objective to decrease room illuminance, and the same plan library as in Scenario 1. However, now, the environment is dynamic: Artifacts may unexpectedly become unavailable, e.g. because they are blocked by other agents, are being moved to other locations, experience network issues, and more. Here, we implement a situation where the lamp becomes unavailable for toggling—a change that is reflected by the absence of any signifiers exposed for the lamp's actions. Agents B, C, and D—which are all equipped with

[6]Our implementation is available in https://github.com/danaivach/jacamo-hypermedia-srm
[7]Our implementation uses the signifier library available in https://github.com/danaivach/hmas-java
[8]The JaCaMO project is available in https://github.com/jacamo-lang/jacamo

an SRM—achieve their objectives by reasoning that the only usable plan is the one for lowering the blinds. On the other hand, agent A, lacking an SRM, fails to achieve its objectives because it attempts to toggle the lamp. This happens because, for all agent programs, the definition of the plan to toggle the lamp precedes the definition of the plan to lower the blinds.

Scenario 3: Plan Applicability based on Context Recommendation. This scenario replicates Scenario 2, but introduces a Design Objective that deviates from the Agent objective of reducing illuminance. Here, the environment's designer suggests that reducing illuminance should only occur when a room is unoccupied. This guideline is conveyed as a recommended context in any signifiers related to the desire of reducing illuminance, including those for toggling the lamp and lowering the blinds. This illustrates the use of signifiers commonly employed by environment designers to customize an environment, similar to adding a sign that says "Please close the door when leaving!" or "Don't touch if in use!" in the physical world. Human agents are able to reason about these signs while retaining the autonomy to disregard them. In our scenario, agents C and D are enabled by their SRMs to reason about recommended contexts, and thereby decide to not decrease the illuminance if they do not believe that the room is empty[9]—adhering to the Design Objective. We assume that agents could redirect their efforts for conserving energy by pursuing other Agent Objectives, such as putting unused lab equipment into idle mode. Agent B lowers the blinds, achieving its Agent Objective but not the Design Objective. Agent A remains error-prone due to its inability to reason about available signifiers, as demonstrated in Scenario 2.

Scenario 4: Plan Applicability based on Ability Recommendation. In the final scenario, we examine signifiers for controlling robotic arms within the room. These signifiers reveal actions for controlling the position of these robotic arms, which are more complex than toggling lamps or lowering blinds. All signifiers refer to the same type of action to move the robot, and recommend moving only when the robot isn't in use by another agent to conform to the Designer's Objective. However, some arms are controlled through an interface that accepts parameters in the Cartesian space, and others through an interface defined in terms of the robot's joints. These input parameter specifics are represented in the signifiers to support acting. Additionally, an agent's ability to work within a specific operational space (here, the Cartesian or joint space) is denoted as a recommended ability to facilitate signifier discovery and advanced action reasoning.

All deployed agents share the Agent Objective of moving the robotic arms to their idle position, and each has a plan to achieve this. Agent A remains error-prone when controlling unavailable robots and fails to consider whether they are in use, missing all objectives. Agent B also overlooks robot usage by others, missing all objectives. Agent C has the potential to align with both Objectives by reasoning about recommended context but fails because it overlooks the specification of the parameter space, and enters the parameters in Cartesian space while the robot expects joint space parameters. Agent D can reason based on recommended abilities, proceeding to control the robot only if it can handle the required operational space. In cases where it lacks this ability, we assume that it can request another capable agent to perform the desired action. Agent D is the only agent whose SRM allows for effective reasoning to achieve all objectives.

[9]For our deployment, we assume that agents infer whether the room is empty by monitoring the activity status of the room's robotic arms.

Table 2: Scenario results. The first mark represents the Agent Objective, and the second mark the Design Objective.

Scenario	*A*	*B*	*C*	*D*
1	✓✓	✓✓	✓✓	✓✓
2	✗✗	✓✓	✓✓	✓✓
3	✗✗	✓✗	✓✓	✓✓
4	✗✗	✗✗	✗✗	✓✓

The performance of the agents across all four scenarios is shown in Table 2, where check marks indicate success in achieving the Agent Objective (first mark), and Design Objective (second mark). As the scenarios introduced added layers of complexity—from basic plan applicability to contextual awareness and ability-based relevance—it is evident that agents with a more complex SRM are better equipped to both fulfill their goals and adhere to external design objectives.

5 CONCLUSION

In this paper, we employ the signifier abstraction [34] to advertise discoverable hypermedia actions for BDI agents. A signifier resolution mechanism (SRM) is introduced to enhance BDI agents that perform means-end reasoning with the ability to resolve predefined abstract actions in plans to available concrete actions at run time. Our implementation shows how various SRM configurations provide differing levels of support during reasoning in dynamic and unfamiliar environments.

Further research on SRMs could further improve the support they offer to agents in such environments: Integration with a trust and reputation model (e.g., [20]) could enable resolution based on signifier provenance and trustworthiness to enhance interaction transparency and accountability in unknown environments. Additionally, the trade-offs of SRMs should be examined, especially in comparison to mechanisms where environmental entities pre-filter signifiers advertised to agents [25, 34]. These mechanisms may relieve agents from reasoning about signifiers but potentially affect privacy (e.g., by requiring details about agents' intentions) or autonomy (e.g., by hiding useful signifiers). Further investigation could provide insights into effective strategies for flexibly combining and alternating between SRMs and complementary mechanisms, aiming to balance agent autonomy and performance when reasoning about evolving action repertoires.

ACKNOWLEDGMENTS

This research has received funding from the Swiss National Science Foundation under grant No. 189474 (*HyperAgents*) and from the European Union's Horizon 2020 research and innovation program under grant No. 957218 (*IntellIoT*).

REFERENCES

[1] Mike Amundsen. 2017. *RESTful Web Clients: Enabling Reuse Through Hypermedia*. O'Reilly Media, Inc.

[2] André Antakli, Akbar Kazimov, Daniel Spieldenner, Gloria Elena Jaramillo Rojas, Ingo Zinnikus, and Matthias Klusch. 2023. AJAN: An Engineering Framework for Semantic Web-Enabled Agents and Multi-Agent Systems. In *International Conference on Practical Applications of Agents and Multi-Agent Systems*. Springer.

[3] Simon Bienz, Andrei Ciortea, Simon Mayer, Fabien Gandon, and Olivier Corby. 2019. Escaping the Streetlight Effect: Semantic Hypermedia Search Enhances Autonomous Behavior in the Web of Things. In *Proceedings of the 9th International Conference on the Internet of Things*. Association for Computing Machinery.

[4] Olivier Boissier, Rafael H Bordini, Jomi Hubner, and Alessandro Ricci. 2020. *Multi-Agent Oriented Programming: Programming Multi-Agent Systems Using JaCaMo*. Mit Press.

[5] Rafael H Bordini, Jomi Fred Hübner, and Michael Wooldridge. 2007. *Programming Multi-Agent Aystems in AgentSpeak Using Jason*. John Wiley & Sons.

[6] Victor Charpenay, Tobias Käfer, and Andreas Harth. 2021. A Unifying Framework for Agency in Hypermedia Environments. In *International Workshop on Engineering Multi-Agent Systems*. Springer.

[7] Victor Charpenay and Matthias Kovatsch. 2023. *Hypermedia Controls Ontology*. W3C Internal Document. W3C. https://www.w3.org/2019/wot/hypermedia.

[8] Victor Charpenay, Maxime Lefrançois, and María Poveda Villalón. 2023. *JSON Schema in RDF*. W3C Internal Document. W3C. https://www.w3.org/2019/wot/json-schema.

[9] Victor Charpenay, Antoine Zimmermann, Maxime Lefrançois, and Olivier Boissier. 2022. Hypermedea: A Framework for Web (of Things) Agents. In *Companion Proceedings of the Web Conference 2022*. Association for Computing Machinery.

[10] Andrei Ciortea, Olivier Boissier, and Alessandro Ricci. 2019. Engineering World-Wide Multi-Agent Systems with Hypermedia. In *Engineering Multi-Agent Systems*, Danny Weyns, Viviana Mascardi, and Alessandro Ricci (Eds.). Springer.

[11] Andrei Ciortea, Fabien Gandon, Maxime Lefrançois, Danai Vachtsevanou, Christopher Leturc, Olivier Boissier, Luis Gustavo Nardin, Simon Mayer, Victor Charpenay, and Antoine Zimmermann. 2021. *Hypermedia MAS Core Ontology*. Technical Report. HyperAgents. https://purl.org/hmas/core.

[12] Andrei Ciortea, Simon Mayer, Fabien Gandon, Olivier Boissier, Alessandro Ricci, and Antoine Zimmermann. 2019. A Decade in Hindsight: The Missing Bridge Between Multi-Agent Systems and the World Wide Web. In *Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems*. International Foundation for Autonomous Agents and Multiagent Systems.

[13] Andrei Ciortea, Antoine Zimmermann, Olivier Boissier, and Adina Magda Florea. 2015. Towards a Social and Ubiquitous Web: A Model for Socio-Technical Networks. In *2015 IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology (WI-IAT)*, Vol. 1. IEEE.

[14] Laura Daniele, Raúl Garcia-Castro, Maxime Lefrançois, and María Poveda Villalón. 2020. *SAREF: the Smart Applications REFerence ontology*. ETSI Technical Specification. ETSI. https://www.w3.org/TR/2017/REC-shacl-20170720/.

[15] Mehdi Dastani, M. van Birna Riemsdijk, and John-Jules Ch Meyer. 2005. Programming Multi-Agent Systems in 3APL. *Multi-agent programming: Languages, platforms and applications*. Springer.

[16] Louise A. Dennis and Berndt Farwer. 2008. GWENDOLEN: A BDI Language for Verifiable Agents. In *Proceedings of the AISB 2008 Symposium on Logic and the Simulation of Interaction and Reasoning, Society for the Study of Artificial Intelligence and Simulation of Behaviour*.

[17] Jérôme Euzenat, Pavel Shvaiko, et al. 2007. *Ontology Matching*. Vol. 18. Springer.

[18] Dominique Guinard, Vlad Trifa, Friedemann Mattern, and Erik Wilde. 2011. From the Internet of Things to the Web of Things: Resource-Oriented Architecture and Best Practices. *Architecting the Internet of things*. Springer.

[19] Koen V Hindriks. 2009. Programming Rational Agents in GOAL. In *Multi-agent programming: Languages, tools and applications*. Springer.

[20] Trung Dong Huynh, Nicholas R Jennings, and Nigel Shadbolt. 2004. FIRE: An Integrated Trust and Reputation Model for Open Multi-Agent Systems. Springer.

[21] Sebastian Käbisch, Ege Korkan, and Michael McCool. 2023. *Web of Things (WoT) Thing Description 1.1*. W3C Proposed Reccommendation. W3C. https://www.w3.org/TR/2023/PR-wot-thing-description11-20230711/.

[22] Tobias Käfer and Andreas Harth. 2018. Rule-based Programming of User Agents for Linked Data.. In *Proceedings of the 11th International Workshop on Linked Data on the Web at the Web Conference*. CEUR-WS.

[23] Holger Knublauch and Dimitris Kontokostas. 2017. *Shapes Constraint Language (SHACL)*. W3C Recommendation. W3C. https://www.w3.org/TR/2017/REC-shacl-20170720/.

[24] Markus Lanthaler. 2021. *Hydra Core Vocabulary: A Vocabulary for Hypermedia-Driven Web APIs*. Unofficial Draft. Hydra W3C Community Group. http://www.hydra-cg.com/spec/latest/core/.

[25] Jérémy Lemée, Danai Vachtsevanou, Simon Mayer, and Andrei Ciortea. 2022. Signifiers for Affordance-driven Multi-Agent Systems. In *International Workshop on Engineering Multi-Agent Systems (EMAS) at the 21st International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*.

[26] Simon Mayer, Ruben Verborgh, Matthias Kovatsch, and Friedemann Mattern. 2016. Smart Configuration of Smart Environments. *IEEE Transactions on Automation Science and Engineering* 13, 3 (2016), 1247–1255. https://doi.org/10.1109/TASE.2016.2533321

[27] Alexander Pokahr, Lars Braubach, and Winfried Lamersdorf. 2005. Jadex: A BDI Reasoning Engine. *Multi-Agent Programming: Languages, Platforms and Applications*. Springer.

[28] Anand S Rao. 1996. AgentSpeak (L): BDI Agents Speak Out in a Logical Computable Language. In *European Workshop on Modelling Autonomous Agents in a Multi-Agent World*. Springer.

[29] Alexandru Sorici and Adina Magda Florea. 2023. Towards Context-based Authorizations for Interactions in Hypermedia-Driven Agent Environments-The CASHMERE framework. In *International Workshop on Engineering Multi-Agent Systems (EMAS) at the 22nd International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*.

[30] Steffen Stadtmüller, Sebastian Speiser, Andreas Harth, and Rudi Studer. 2013. Data-fu: A Language and an Interpreter for Interaction with Read/Write Linked Data. In *Proceedings of the 22nd International Conference on World Wide Web*. Association for Computing Machinery.

[31] Peter Stringer, Rafael C. Cardoso, Clare Dixon, and Louise A. Dennis. 2021. Implementing Durative Actions with Failure Detection in GWENDOLEN. In *International Workshop on Engineering Multi-Agent Systems*. Springer.

[32] Peter Stringer, Rafael C. Cardoso, Clare Dixon, Michael Fisher, and Louise A. Dennis. 2023. Updating Action Descriptions and Plans for Cognitive Agents. In *Proceedings of the 2023 International Conference on Autonomous Agents and Multi-agent Systems*. International Foundation for Autonomous Agents and Multiagent Systems.

[33] Peter Stringer, Rafael C. Cardoso, Xiaowei Huang, and Louise A. Dennis. 2020. Adaptable and Verifiable BDI Reasoning. *arXiv preprint arXiv:2007.11743*.

[34] Danai Vachtsevanou, Andrei Ciortea, Simon Mayer, and Jérémy Lemée. 2023. Signifiers as a First-Class Abstraction in Hypermedia Multi-Agent Systems. In *Proceedings of the 2023 International Conference on Autonomous Agents and Multi-agent Systems*. International Foundation for Autonomous Agents and Multiagent Systems.

[35] Ruben Verborgh, Vincent Haerinck, Thomas Steiner, Davy Van Deursen, Sofie Van Hoecke, Jos De Roo, Rik Van de Walle, and Joaquim Gabarro. 2012. Functional Composition of Sensor Web APIs. In *Proceedings of the 5th International Conference on Semantic Sensor Networks (SSN)*. Association for Computing Machinery.

[36] Rem W. Collier, Eoin O'Neill, David Lillis, and Gregory O'Hare. 2019. MAMS: Multi-Agent MicroServices. In *Companion Proceedings of the 2019 World Wide Web Conference*. Association for Computing Machinery.

[37] Norman Walsh and Ian Jacobs. 2004. *Architecture of the World Wide Web, Volume One*. W3C Recommendation. W3C. https://www.w3.org/TR/2004/REC-webarch-20041215/.

[38] Danny Weyns, Andrea Omicini, and James Odell. 2007. Environment as a First Class Abstraction in Multiagent Systems. *Autonomous Agents and Multi-Agent Systems* 14, 1. Springer.

[39] Erik Wilde. 2007. *Putting Things to REST*. Technical Report UCB iSchool Report 2007-015. University of California at Berkeley.

[40] Michael Winikoff and Ling Padgham. 2013. Agent-Oriented Software Engineering. In *Multiagent Systems, Second Edition*, Gerhard Weiss (Ed.). MIT Press.

[41] David Wood, Richard Cyganiak, and Markus Lanthaler. 2014. *RDF 1.1 Concepts and Abstract Syntax*. W3C Recommendation. W3C. https://www.w3.org/TR/2014/REC-rdf11-concepts-20140225/.

[42] Michael Wooldridge. 2013. Agent Architectures and Organizations. In *Multiagent Systems, Second Edition*, Gerhard Weiss (Ed.). MIT Press.