Engineering World-Wide Multi-Agent Systems with Hypermedia

Andrei Ciortea[1], Olivier Boissier[1], and Alessandro Ricci[2]

[1] Univ. Lyon, MINES Saint-Étienne, CNRS Lab Hubert Curien UMR 5516
Saint-Étienne, France F-42023
{andrei.ciortea,olivier.boissier}@emse.fr
[2] University of Bologna, Cesena, Italy
a.ricci@unibo.it

Abstract. A well studied problem in the engineering of open MASs is to enable uniform interaction among heterogeneous agents. However, AOSE as a field has grown to recognize that a MAS consists of more than only agents and thus should be designed on multiple dimensions (including the environment, organization etc.). The problem of enabling interaction among heterogeneous entities across dimensions is either not considered, or it is addressed in an *ad hoc* and non-uniform manner. In this paper, we introduce a novel approach to use hypermedia as a general mechanism to support uniform interaction in MASs. The core idea is that agents use hypermedia to discover (i) other entities in a MAS (e.g., other agents, tools, organizations) and (ii) the means to interact with those entities. This reduces coupling and enhances the scalability and evolvability of the MAS. We present a demonstrator that supports these claims. We believe that a hypermedia-based mechanism for uniform interaction in MASs could provide a foundation for engineering world-wide MASs.

Keywords: Multi-agent systems; hypermedia systems; interaction

1 Introduction

The vision of world-wide multi-agent systems (MASs) has been around for some time. In 2001, the Agentcities initiative was aiming to create a world-wide open network of heterogeneous agents to which any organization or individual researcher could connect their agents [32]. The same year, the seminal paper on the Semantic Web was published [2], which promoted the vision of a Web for both people and autonomous agents. Since then, we have witnessed significant progress both in MASs and Semantic Web research, but we have yet to witness the deployment of world-wide and long-lived MASs.

The Web, on the other hand, has had remarkable success as a world-wide and long-lived system *of people*. One way to look at the Web is that it provides people with a *distributed hypermedia environment* (composed of interrelated Web pages) that they can navigate and use in pursuit of their goals. This hypermedia environment was specifically designed to be open, Internet-scale, and to allow people to use it in new and unanticipated ways [3,12]. All these properties were designed into the Web architecture, and its central distinctive feature is the use of hypermedia as an engine for uniform interaction between components [12]. We believe that we can apply the same design rationale as an effective means to engineer open, scalable, and evolvable MASs – which implies significantly more than just implementing MASs using Web services.

Our hypothesis is that we can use hypermedia to create a general mechanism for uniform interaction in MASs. Given such a mechanism, heterogeneous agents would then be able to discover and interact in a uniform manner with other agents as well as other heterogeneous entities (tools, knowledge repositories, organizations, datasets etc.) that could help them achieve their goals. Engineers in different parts of the world could then develop and deploy agents and other entities independently from one another, and old and new implementations could co-exist in one system.

A mechanism for uniform interaction in MASs such as the one described above is currently lacking. Perhaps the closest solution can be found in the FIPA standards[3], but it only addresses interaction among agents. Interaction between agents and other entities in a MAS is not addressed.

In this paper, we introduce an approach to use hypermedia for uniform interaction in MASs. To support our approach, we implemented a demonstrator in which BDI agents are able to discover and interact with artifacts in a distributed hypermedia environment. The distribution of the environment is seamless from the agents' viewpoint, and agents are able to exploit new functionalities added to artifacts at runtime. This demonstrator confirms key elements of our hypothesis and suggests it could provide a foundation for engineering world-wide MASs.

We discuss background and related work in Section 2. In Section 3, we propose a set of design principles and a model for hypermedia MASs. We report on our implementation and experience in Section 4, and then conclude in Section 5.

2 Background and Related Work

In the following, we first discuss the role of hypermedia in the Web architecture, and then current approaches to engineer Web-based and world-wide MASs.

2.1 Hypermedia and HATEOAS

A central feature of REST, the architectural style of the Web, is that it uses hypermedia to drive interaction between components, a principle known as *hypermedia as the engine of application state (HATEOAS)* – see [12] for details. To illustrate this principle, an HTML page typically provides the user with a number of affordances, such as to navigate to a different page by clicking a hyperlink or to submit an order by filling out and submitting an HTML form. Performing any such action transitions the application to a new state, which provides the

[3] `http://www.fipa.org/repository/standardspecs.html`, accessed: 10.05.2018.

user with a new set of affordances. In each state, the user's browser retrieves an HTML representation of the current state from a server, but also a selection of next possible states and the information required to construct the HTTP requests to transition to those states. Retrieving all this information through hypermedia allows the application to evolve without impacting the browser, and allows the browser to transition seamlessly across servers.

In contrast, when using a non-hypermedia Web service (e.g., an implementation of CRUD operations over HTTP), developers have to hard-code into clients all the knowledge required to interact with the service. This approach is simple and intuitive for developers, but clients are tightly coupled to the services they use (hence the need for API versioning). In recent years, hypermedia has started to receive more attention in Web service design (e.g., [21]), in particular in the context of the Web of Things (WoT) [19] – where it is important for devices to be able to interact with one another in a loosely coupled manner rather than having developers constantly updating the integrations as devices evolve.

2.2 Web-based Multi-agent Systems

There has been extensive research on using the Web as an infrastructure for distributed MASs by integrating MASs with Web services. However, most of the existing approaches only use the Web as a transport layer and therefore are misaligned with the Web architecture (see Section 6.5.3 in [13]). This includes all MAS platforms that implement the FIPA specification for using HTTP as a message transport protocol [15], as well as those approaches that implement the WS-* standards (e.g., [1,29]) – which also use the Web only for transport (see [24] for more details). More recent approaches for Web-based MASs have turned to REST-like Web services (e.g., [23,17]), but they generally do not use hypermedia or HATEOAS.

2.3 World-wide Multi-agent Systems

There have been considerable efforts to support the engineering of open, world-wide, and long-lived MASs – among the most prominent, the FIPA standards, Agentcities [32], DARPA CoABS GRID [20], and the Semantic Web [2]. These efforts have generally focused on enabling uniform interaction among heterogeneous agents (e.g., see [14,16]), but a general mechanism for uniform interaction with *any* entity in an open MAS is lacking. From an architectural perspective, most previous efforts to engineer large-scale MASs generally relied on RPC-like architectures.[4] When it comes to engineering world-wide systems, RPC-like architectures have shortcomings as compared to REST-style, resource-oriented architectures. For instance, they cannot (or make it very difficult to) use intermediary layers that can process requests almost as well as their intended recipients (see Section 6.5.2 in [13]). Intermediaries have proven very useful in

[4] This includes approaches based on Web services that tunnel RPC-like method invocations through HTTP (e.g., using SOAP).

world-wide systems such as the Web. Furthermore, in a REST-style architecture, hypermedia enables the serendipitous use of resources, a property promoted in recent years by the *linked data* initiative [4]. Support for new and unanticipated applications is an important property for sustaining long-lived systems.

3 Hypermedia Multi-agent Systems

We consider that a sensible path towards the engineering of open, world-wide, and long-lived MASs is to use hypermedia as a uniform interaction engine in MASs. In a *hypermedia MAS*, agents are situated in a *distributed hypermedia environment* that they can navigate and use in pursuit of their goals. The environment is a first-class abstraction in the system and on the surface it provides agents with all typical functionalities of endogenous environments (see [31,26]), such as interaction with services, tools, and the external world, or mediating interaction and coordination with other agents. However, in contrast to typical endogenous environments, a *hypermedia environment* uses hypermedia to *drive interaction* in the MAS: agents navigate the hypermedia environment to discover other entities in the MAS, as well as the means to interact with those entities. This reduces coupling and enhances the scalability and evolvability of the systems. We introduce a set of design principles for hypermedia MASs in Section 3.1, and then present a concrete model in Section 3.2.

3.1 Design Principles for Hypermedia Multi-agent Systems

As discussed in Section 2.2, the mere use of Web services is not sufficient to create a hypermedia MAS. We introduce three key design principles meant to ensure the proper use of hypermedia as a general mechanism for uniform interaction in MASs. These principles are based on the design rationale behind the Web architecture [12]. As it is generally the case with software design principles, the proposed principles impose constraints on the design of MASs. Engineers can choose to ignore one or more of these principles, but then the MASs would most likely make limited use of hypermedia and would not achieve uniform interaction.

Principle 1 (Uniform resource space) *All entities in a hypermedia MAS and relations among them should be represented in the hypermedia environment in a uniform, resource-oriented manner.*

In a hypermedia system, and in particular in REST-style systems, a *resource* is the key abstraction of information [12]. The core idea behind this first principle is to project the entire observable state of the MAS into the distributed hypermedia environment in a uniform, resource-oriented manner (e.g., as an *RDF graph* [10]) such that agents can interpret, reason upon, and interact with the MAS by consuming and producing hypermedia. For instance, one agent could send a message to another by writing an RDF representation of the message in the hypermedia. To receive messages, an agent could observe a resource that represents its mailbox in the hypermedia. To turn on a light bulb, an agent could manipulate the state of a resource that represents the light bulb in the hypermedia. Interactions between agents and resources in their hypermedia environment should conform to the REST constraints. In what follows, we discuss only three key aspects of applying REST to hypermedia MASs (and refer readers to [12] for more details about REST): *uniform identification* of entities, *uniform representation* of entities, and the use of *relations* among entities to enable discoverability.

Uniform identification (e.g., via IRIs [11]) allows entities in a hypermedia MAS to be referenced globally. Agents can then use hypermedia to interact with the entities regardless of their location. For instance, if an agent or a light bulb in its environment are identified via IRIs, then they can be referenced without the need for additional contextual information – such as how to interpret platform-specific identifiers, or low-level network information (e.g., IP addresses of hosts). In non-hypermedia MASs, this is not generally the case. FIPA defined its own standard uniform identifier for agents (see Section 3 in [16]), but uniform identification of other entities in a MAS is not addressed. Platforms that support distributed endogenous environments, such as CArtAgO [26], either require agents to manage the low-level network information when joining remote nodes, or use additional infrastructure to manage this information (e.g., [22]).

Uniform representation of entities in hypermedia MASs allows to hide any implementation-specific details behind standardized knowledge models. For instance, the state of a light bulb could be represented in the hypermedia environment using RDF and some standard ontology. An agent could then interact with the light bulb by interpreting and manipulating its semantic representation either directly or via some intermediary tool (e.g., to translate it in a different knowledge representation language, or to use the light bulb via an artifact). A similar approach was taken by FIPA to define a standard format for ACL messages exchanged between agents [14], and to describe agents and the services they provide [16]. However, the uniform modeling of any other entities in a MAS is not addressed.

In a hypermedia MAS, *relations* among entities (e.g., agents, tools, documents, organizations, datasets) can be represented explicitly in the hypermedia. The relations can then be crawled to discover entities of interest in the MAS. Agents could do the crawling themselves, or they could use search engines that do the crawling for them. This approach to discoverability has proven very practical for open, large-scale and decentralized systems, such as the Web [6]. In non-hypermedia MASs, discoverability is typically based on the registration of agents (and their services) to centralized directories, such as the *Directory Facilitator (DF)* standardized by FIPA [16]. Multiple DFs can then be federated to support decentralized searches, where requests are propagated between DFs up to a maximum depth level. However, this approach to discoverability is biased towards the locality of agents, which may prove inefficient in a Web-scale MASs. In contrast, crawling-based DFs could avoid the locality bias by exploiting relations in the distributed hypermedia environment. Furthermore, having typed relations among federated DFs could help propagate search requests in an informed manner.

The uniform resource space principle provides the underpinning for scalable and evolvable hypermedia MASs: it enables the seamless distribution of the hypermedia environment, and it allows agents to interact with other entities in the MAS in a uniform manner through hypermedia. The implied trade-off is interoperability vs. innovation: translating an implementation-specific model to a uniform representation promotes interoperability, but may loose implementation-specific features.

Principle 2 (Single entry point) *Given a single entry point into the environment of a hypermedia MAS, an agent should be able to discover the knowledge required to participate in the system by navigating the hypermedia.*

The core idea behind this second principle is to maximize the usage of hypermedia in order to minimize coupling in the MAS. As mentioned for Principle 1, hypermedia can help reduce coupling by enabling system-wide discoverability – agents can crawl the hypermedia to discover what other agents, tools, or entities in the system can help them achieve their goals (hard-coding any such relations into the agents would increase coupling). Equally important, however, agents can also discover in the hypermedia how to interact with entities: the affordances of resources in their environment (e.g., operations exposed by a light bulb and how to perform them), specifications of agent interaction protocols in a given language (e.g., BSPL [28]), specifications of organizations in a given language (e.g., MOISE [18]), polices created by policy engineers (e.g., the terms of service of a hypermedia search engine), or norms created by other agents (e.g., norms that emerged in a given society) etc. The single entry point principle implies that any knowledge required to participate in the system that can be represented in the hypermedia should be represented in the hypermedia.

In other words, given an entry point into the environment of a hypermedia MAS, agents should require minimal a priori knowledge to interact with entities in the system besides the general knowledge required to consume and produce hypermedia. Any a priori knowledge and assumptions required to participate in the system should be standardized at the system-level (i.e., shared by everyone in the system). All other knowledge required to participate in the system should be discovered in the hypermedia. Furthermore, if an agent has to interact with one or more entities to achieve its goals and those entities are present in the system, the hypermedia environment should allow their eventual discovery via crawling. Violating any of these two constraints (i.e., hard-coding *ad hoc* knowledge into agents instead of placing it in the hypermedia, not enabling navigability) would violate the single entry point principle.

To illustrate the above point with an example, say an agent in a hypermedia MAS has to turn on a light bulb. A priori knowledge required by the agent to achieve its goal could include the HTTP protocol, RDF standards, a general model of its environment, and an ontology describing light bulbs and the operations they expose (e.g., turn on), which could all be standardized at the system-level. Knowledge that should be discovered in the hypermedia would include the uniform identifier of a light bulb and the specification of an HTTP request that turns on the light bulb. Hard-coding into the agent the light bulb's identifier or, for instance, the knowledge required to use the Philips Hue HTTP API[5] (e.g., the Philips Hue data model) would couple the agent to the light bulb, and similar knowledge would have to be hard-coded in order to use light bulbs from other manufacturers.

The single entry point principle is central to designing evolvable and long-lived hypermedia MASs. The main trade-off is that relying on knowledge discovered in the hypermedia can increase the complexity of programming the MAS, but this can be mitigated through the use of appropriate middleware (as we show in Section 4).

Principle 3 (Observability) *In a hypermedia MAS, any resource in the hypermedia environment that could be of interest to agents should be observable.*

The first two principles ensure the dynamic discovery of a hypermedia MAS via crawling. However, constantly crawling large hypermedia MASs to keep track of their evolution would be inefficient. Instead, this third principle promotes the use of mechanisms that allow agents to selectively observe entities of interest in the MAS via the entities' representation in the hypermedia environment, which could include the entities' states, affordances, relations to other entities etc. This principle improves the scalability of hypermedia MASs: agents can handle larger environments, and at the same time the load on the hypermedia infrastructure is decreased (and thus it can serve more agents). The trade-off is the extra complexity added by observability mechanisms, but this can be mitigated through the use of appropriate middleware and intermediary components (see Section 4).

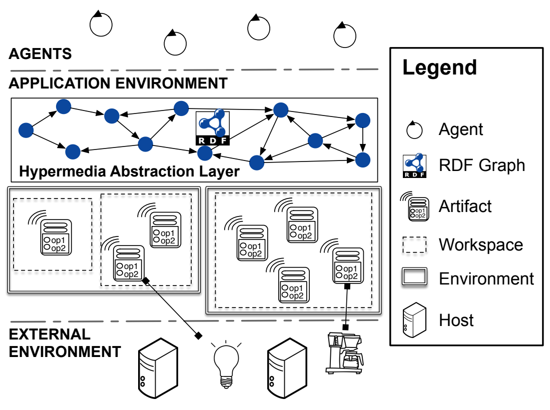
3.2 A Model for Hypermedia Multi-agent Systems

We applied the proposed design principles to define a model for hypermedia MASs. This model is intended to provide an extensible conceptual foundation and thus only defines the core abstractions required to design and program hypermedia MASs.[6] The model (see Figure 1a) is based on the *Agents and Artifacts (A&A)* meta-model [26] and our previous research on *socio-technical networks* (e.g., see [7,9]). We present the model in what follows, and then introduce a Web ontology that formalizes this model and discuss its usage.

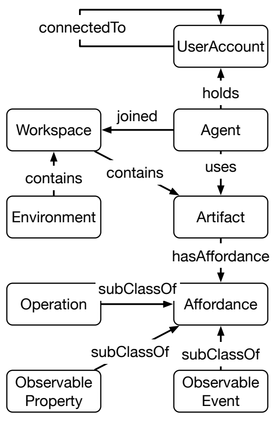
Cognitive agents in artifact-based hypermedia environments Following the A&A meta-model, we *design* and *program* hypermedia MASs in terms of *agents* and *artifacts*. In A&A, *artifacts* are first-class programming abstractions (and not just design abstractions): they are as much a programmable part of the

[5] `https://developers.meethue.com/documentation/getting-started`, accessed: 06.05.2018.

[6] We leave a more complete treatment of MASs as future work (see Section 5).



(a) Layers of abstraction in hypermedia MASs.



(b) EVE ontology.

Fig. 1: The core concepts used to model hypermedia MASs.

MAS as are agents. We use artifacts to program the *application environment* (cf. Figure 1a) such that it provides agents with a *uniform, general interface* defined in terms of *observable properties*, *observable events*, and *operations* (see [26] for more details on artifacts). Artifacts thus help reduce the coupling between agents and their environment.

As depicted in Figure 1a, all entities (e.g., agents, their services, artifacts, workspaces), the relations among them, and the affordances of artifacts are projected into a uniform RDF abstraction layer – cf. Principle 1 (uniform resource space). This layer effectively decouples agents from the application environment and enables system-wide discoverability via crawling. From an agent's viewpoint, the set of all affordances of its environment is determined by the artifacts discovered in the hypermedia, where the affordances of a given artifact are also discovered in the hypermedia – see Principle 2 (single entry point). To avoid dealing with low-level manipulation of hypermedia, agents could use hypermedia controllers (similar to Web browsers) in order to interact seamlessly with artifacts through the hypermedia layer (see implementation in Section 4.2).

The *environment* and *workspace* abstractions (cf. Figure 1a) are containers that allow agents to prune their crawling in the hypermedia environment, as well as focus their observations on those parts of the environment that are of interest – cf. Principle 3 (observability). When observing an artifact, the artifact's observable properties and events constitute percepts. In BDI architectures, such percepts can be modeled inside agents as beliefs about the state of their environment. Agents can use artifacts to observe other agents in the system regardless of their location.

Hypermedia networks of agents and artifacts We formalized the proposed model in an OWL 2 ontology, which we call the *Agent Environment (EVE)* ontology (see Figure 1b). Engineers can use the ontology to create uniform representations of hypermedia MASs based on the proposed core model. The ontology can then be extended with modules for other dimensions of MASs, or with domain- and application-specific modules. A benefit of choosing Web standards is that engineers of hypermedia MASs can then benefit from all the results and resources provided by the Semantic Web community (ontologies, tooling etc.).

To enforce Principle 2 (single entry point), it is important to consider the navigability of hypermedia MASs. Representing explicitly in the environment all `eve:joined` and `eve:contains` properties guarantees that all agents and artifacts in the system are discoverable. Agents are represented in the system via *user accounts*, which can be used (among others) to interact with other agents and observe their relations. Representing explicitly the agents' relations (e.g., via `eve:connectedTo`) can help enhance navigability. The affordances of artifacts should also be described explicitly. The EVE ontology defines concepts and properties for this purpose (cf. Figure 1b). However, it is outside the scope of this ontology to describe implementations of affordances. Other existing ontologies can be used in conjunction with the EVE ontology for this purpose (e.g., [19]).

4 Implementation and Experience

To demonstrate our approach, we developed a prototype platform for hypermedia MASs and used it to deploy a distributed hypermedia environment. We present the demonstrator scenario in Section 4.1, and an overview of the deployed system in Section 4.2. We discuss what has been achieved through this demonstrator and what are the limitations in Section 4.3.

4.1 Demonstrator Scenario

We implemented a demonstrator in which a BDI agent has to notify a human whenever an event occurs. The agent is situated in a hypermedia environment that contains one workspace with two artifacts: an artifact that generates two types of observable events (i.e., `positive` and `negative`), and a light bulb artifact that can be used to send visual notifications to humans. The hypermedia environment is distributed. The agent is given an entry IRI in the hypermedia environment and has to discover the rest of the system at runtime.

In the beginning of our demonstration, the environment contains only the event generator artifact (henceforth the `event-gen` artifact). The agent discovers the `event-gen` artifact and starts observing the generated events, but at this point the agent has no means to notify humans. The `light-bulb` artifact is added to the environment. The agent discovers the `light-bulb` artifact and its affordances, and can now send visual notifications to humans by flicking the light bulb. At this point, however, the visual notifications do not differentiate between `positive` or `negative` events – the light bulb is simply turned on and off.

While the system is running, a developer extends the `light-bulb` artifact with a new operation for setting the color of the light. The agent discovers the newly added operation and can now send visual notifications with different color codes (i.e., green light for `positive` events, blue light for `negative` events).

4.2 System Overview and Deployment

We deployed the hypermedia environment in our demonstrator scenario using a prototype platform for hypermedia MASs, named Yggdrasill[7]. The environment was distributed across two Yggdrasill nodes: one node hosted the `event-gen` artifact and was deployed on a virtual machine in the cloud, and the other node hosted the `light-bulb` artifact and was deployed on a Raspberry Pi in our local network. We used a Philips Hue light bulb that is accessed via the HTTP API exposed by a Philips Hue bridge in the local network. The BDI agent in our scenario was implemented in a separate JaCaMo [5] application that was deployed on a MacBook Air machine in the local network.

In what follows, we first present the deployed hypermedia environment, and then discuss our implementation considerations for Yggdrasill and the JaCaMo application.

Hypermedia environment We constructed the hypermedia environment in our scenario using the EVE ontology. Dereferencing the environment's IRI retrieves the RDF representation shown in Listing 1.1. This representation points to one contained workspace (line 4), which allows agents to continue their crawling. Dereferencing the workspace IRI returns a similar representation (presented later in Listing 1.3) that points to the artifacts available in that workspace.

Listing 1.1: A Turtle [25] representation of the deployed environment created with the EVE ontology.

```
1 @prefix eve: <http://w3id.org/eve#> .
2
3 <http://yggdrasill.andreiciortea.ro/environments/env1> a eve:Environment ;
4   eve:contains <http://yggdrasill.andreiciortea.ro/workspaces/wksp1> .
```

Dereferencing the IRI of the `light-bulb` artifact retrieves the representation shown in Listing 1.2. In addition to the EVE ontology, this representation uses the W3C Web of Things (WoT) Thing Description (TD) ontology (currently being standardized [19]) and is based on a WoT TD used at the latest plugfest of the W3C WoT Working Group. The WoT TD ontology was designed to describe interactions with *things* in the WoT. Even though the *thing* and *artifact* abstractions have been developed independently in two different communities (and for different purposes), the abstractions define a similar interface composed of *observable properties*, *observable events*, and *actions* (or *operations*, respectively). This makes the WoT TD ontology a good candidate standard for describing interfaces of artifacts in hypermedia environments. Here, we use the WoT TD to describe the HTTP API of the Philips Hue light bulb in our deployment.

[7] In Norse mythology, Yggdrasill is an immense tree that connects all existing worlds.

Listing 1.2 shows the description of an artifact operation for setting the color of a Philips Hue light bulb. In our demonstrator, this operation is added in the hypermedia by a developer while the system is running. We have created similar operation descriptions for turning the light bulb on and off.

Listing 1.2: This listing shows an excerpt from the RDF representation used for the `light-bulb` artifact in our deployment. The operation shown here allows an agent to set the color of a given Philips Hue light bulb. The description includes a full specification of the HTTP request that implements the operation.

```
@prefix td: <http://www.w3.org/ns/td#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
@prefix iot: <http://iotschema.org/> .
@prefix http: <http://iotschema.org/protocol/http> .

<http://85.204.10.233:8080/artifacts/hue1>
  a eve:Artifact, td:Thing, iot:Light, iot:BinarySwitch ;
  eve:hasName "Lamp"^^xsd:string ;
  td:base "http://192.168.0.101/"^^xsd:anyURI ;
  td:interaction [
    a eve:Operation, td:Action, iot:SetColor ;
    td:name "Set Color"^^xsd:string ;
    td:form [
      http:methodName "PUT"^^xsd:string ;
      td:href "/api/YqqaHVH8QF-...-UQc/lights/3/state"^^xsd:anyURI ;
      td:mediaType "application/json"^^xsd:string ;
      td:rel "invokeAction"^^xsd:string
    ] ;
    td:inputSchema [
      td:schemaType td:Object ;
      td:field [
        td:name "on"^^xsd:string ;
        td:schema [ td:schemaType td:Boolean; td:const true ]
      ] ;
      td:field [
        td:name "xy"^^xsd:string ;
        td:schema [
          td:schemaType td:Array ;
          td:items [ a iot:CIExData; td:schemaType td:Number ],
            [ a iot:CIEyData; td:schemaType td:Number ]
    ] ] ] ],
(...)
```

Yggdrasill The hypermedia environment deployed in our demonstrator was hosted on Yggdrasill. While still under early development, Yggdrasill provides two core functionalities required by our demonstrator: (i) it serves as a repository for hypermedia environments that conform to the model in Section 3.2, and (ii) it acts as a hub that (partially) implements the W3C WebSub recommendation[8]; agents (or any software clients) can use this functionality to observe resources in the environment.

Yggdrasill implements an event-driven non-blocking architecture using *Vert.x*[9], a framework that is both powerful enough to support high-throughput Web

[8] `https://www.w3.org/TR/2018/REC-websub-20180123/`, accessed: 11.05.2018.
[9] `http://www.vertx.io/`, accessed: 11.05.2018.

servers[10], and lightweight enough to perform well on small devices, such as the Raspberry Pi[11]. The platform exposes a REST HTTP API for creating, updating, and deleting RDF representations of *environment*, *workspace*, and *artifact* abstractions. For instance, Listing 1.3 shows an HTTP request that retrieves a Turtle [25] representation of the workspace used in our deployment. Lines 12-14 point to the artifacts contained in this workspace (and make them discoverable).

Listing 1.3: A sample HTTP request (and the corresponding response) for retrieving the representation of a workspace from Yggdrasill.

```
1  GET /workspaces/wksp1 HTTP/1.1
2  Host: yggdrasill.andreiciortea.ro
3
4  HTTP/1.1 200 OK
5  Content-Type: text/turtle
6  Link: <http://yggdrasill.andreiciortea.ro/hub>; rel="hub"
7  Link: <http://yggdrasill.andreiciortea.ro/workspaces/wksp1>; rel="self"
8
9  <http://yggdrasill.andreiciortea.ro/workspaces/wksp1>
10   a <http://w3id.org/eve#Workspace> ;
11   <http://w3id.org/eve#hasName> "wksp1" ;
12   <http://w3id.org/eve#contains>
13     <http://85.204.10.233:8080/artifacts/hue1> ,
14     <http://yggdrasill.andreiciortea.ro/artifacts/event-gen> .
```

The response shown in Listing 1.3 contains two `Link` headers that conform to the W3C WebSub recommendation. Agents can thus *discover and use* these headers to subscribe for notifications whenever the workspace evolves (e.g., an artifact is added or removed, the workspace is deleted). All resources hosted on Yggdrasill support WebSub by default.

JaCaMo application To facilitate access to the hypermedia environment, our JaCaMo application provides agents with "middleware" they can use. Given the IRI of an environment in the hypermedia (e.g., see Listing 1.1), the middleware automatically crawls the environment to reflect in the local CArtAgO environment all discovered workspaces and the artifacts they contain. Agents are notified whenever a workspace or an artifact has been reflected. The middleware also provides agents with a CArtAgO artifact that serves as a hypermedia controller (i.e., a facade) for hypermedia artifacts with WoT TDs (e.g., see Listing 1.2). A controller is instantiated for each such artifact discovered in the hypermedia. Using these controllers, agents can then interact with hypermedia artifacts as they would typically interact with local CArtAgO artifacts. The main difference is that each controller instance exposes metadata via observable properties, such as what are the operations supported by the hypermedia artifact. To perform an operation, such as the one of changing the light color in Listing 1.2, agents use a generic `act` operation provided by the controller, as shown in Listing 1.4. This generic operation takes as arguments the IRI of the actual intended operation

[10] According to independent benchmarks for Web frameworks: `https://www.techempower.com/benchmarks/`, accessed: 11.05.2018.

[11] `http://vertx.io/blog/vert-x3-web-easy-as-pi/`, accessed: 11.05.2018.

and its parameters, which can also carry semantics via IRIs. The IRIs could denote terms defined by an ontology.

Listing 1.4: The Jason plan used to send visual notifications via colored light.

```
+!thing_colored_light_notification(ArtifactName, CIEx, CIEy) : true <-
  act("http://iotschema.org/SetColor", [
           ["http://iotschema.org/CIExData", CIEx],
           ["http://iotschema.org/CIEyData", CIEy]
        ])[artifact_name(ArtifactName)];
  .wait(2000);
  act("http://iotschema.org/SwitchOff", [])[artifact_name(ArtifactName)].
```

In addition to hypermedia controllers, the middleware can also instantiate regular CArtAgO artifacts based on semantic descriptions discovered in the hypermedia, such as the one in Listing 1.5 for the `event-gen` artifact in our demonstrator. This allows offloading the execution of artifacts on the client-side – if the artifact code can be retrieved at runtime (e.g., via JavaScript, OSGi).

Listing 1.5: RDF description of the `event-gen` artifact, which includes the canonical name of the Java class of the corresponding CArtAgO artifact. Initialization parameters could also be specified in the description.

```
<http://yggdrasill.andreiciortea.ro/artifacts/event-gen> a eve:Artifact ;
  eve:hasName "event-gen" ;
  eve:hasCartagoArtifact "emas.EventGeneratorArtifact" .
```

4.3 Discussion

Our demonstrator proves key elements of our hypothesis. In what follows, we analyze what has been demonstrated for the scalability and evolvability of hypermedia MASs, and then discuss the limitations of our demonstrator.

Scalability Our demonstrator shows that agents can perceive and act upon a distributed hypermedia environment while being agnostic to the underlying infrastructure. The environment in our demonstrator is distributed across loosely coupled origin servers, and agents observe the environment using WebSub hubs discovered at runtime. All entities in our demo use the same WebSub hub, but each entity could use any number of hubs (e.g., to distribute the load). Other publish/subscribe mechanisms for Web resources can also be used, such as the one built into CoAP [27]. In principle, any mechanisms that have proved useful for managing the growth of the Web (e.g., load balancers, intermediaries for enforcing security or encapsulating legacy systems) could be applied to manage the growth of the hypermedia environment infrastructure.

Evolvability Our demonstrator shows that hypermedia-driven interaction allows agents and their environments to be deployed and to evolve independently from one another. Agents can discover the distributed hypermedia environment at runtime starting from a single entry point, and they can observe the environment as it evolves – for instance, as workspaces and artifacts are added to the environment. Artifacts themselves can also evolve at runtime without disrupting the behavior of agents. This allows engineers to enrich the MAS with features that were not anticipated when the system was initially deployed. Furthermore, both engineers and agents could further exploit the hypermedia to enrich the system over time, for instance by writing new resources for agents to discover and use (e.g., shared knowledge) or by rewiring relations among resources (e.g., to create mash-ups of artifacts). The Web has already shown that a resource-oriented, evolvable environment can support new and unanticipated applications, which is essential for sustaining long-lived MASs.

Limitations In our demonstrator, agents are able to use artifacts as they evolve at runtime, but the behaviors that use the artifacts are pre-programmed (e.g., see Listing 1.4). However, this is a limitation of our demonstrator and not an intrinsic limitation of our approach. One way to avoid this limitation would be to advertise in the hypermedia *artifact manuals* (e.g., [30,8]) that would allow agents to infer how to achieve their goals using the discovered artifacts.

As mentioned, Yggdrasill is in an early stage of development. As such, it does not yet implement an engine for actually running the artifacts, such as CArtAgO [26]. The logic of the `event-gen` artifact was simulated for the purpose of this demonstrator by sending the generated events via HTTP to Yggdrasill, which then dispatches the events to subscribers. The "middleware" developed in our JaCaMo application is an *ad hoc* solution meant to demonstrate that the additional programming complexity that comes with hypermedia environments can be mitigated through tooling. A proper extension would require a deeper integration into CArtAgO.

5 Conclusions

This paper presents an approach to enable uniform interaction among heterogeneous entities in an open MAS such that the entities can be developed, deployed and can evolve independently from one another. The core idea is to use hypermedia to drive the interaction between agents and their environment. Our demonstrator proves that this approach: (i) can effectively decouple agents from their environment, and (ii) allows a seamless distribution of the environment. Even though in this paper we focus on the agent $\leftrightarrow$ environment interaction, the environment can also mediate interaction with other dimensions of a MAS: it can be used to discover and interact with other agents (e.g., as we have shown in [9,7]), with organizations (e.g., by means of organizational artifacts [5]) etc. In principle, any abstract entity in a MAS that is relevant to agents can be reified in the hypermedia environment – either as a passive resource that agents can discover and consume (e.g., to learn a new interaction protocol), or as an active resource (e.g., a tool) that agents can interact with. The results presented in this paper suggest that using hypermedia as a general mechanism to support uniform interaction in MASs enhances the systems' scalability and evolvability – and could potentially enable the deployment of world-wide and long-lived MASs.

References

1. Estefania Argente, Vicente Botti, Carlos Carrascosa, Adriana Giret, Vicente Julian, and Miguel Rebollo. An abstract architecture for virtual organizations: The thomas approach. *Knowledge and Information Systems*, 29(2):379–403, 2011.
2. T. Berners-Lee, J. Hendler, and O. Lassila. The Semantic Web. *Scientific American*, 284(5):34–43, May 2001.
3. Tim Berners-Lee and Mark Fischetti. *Weaving the Web: The Original Design and Ultimate Destiny of the World Wide Web by Its Inventor*. DIANE Publishing Company, 2001.
4. Christian Bizer, Tom Heath, and Tim Berners-Lee. Linked data - the story so far. *International Journal on Semantic Web and Information Systems (IJSWIS)*, 5(3):1–22, MarMar 2009.
5. Olivier Boissier, Rafael H Bordini, Jomi F Hübner, Alessandro Ricci, and Andrea Santi. Multi-agent oriented programming with jacamo. *Science of Computer Programming*, 78(6):747–761, 2013.
6. Sergey Brin and Lawrence Page. The anatomy of a large-scale hypertextual web search engine. *Computer Networks and ISDN Systems*, 30(1):107 – 117, 1998. Proceedings of the Seventh International World Wide Web Conference.
7. Andrei Ciortea, Olivier Boissier, Antoine Zimmermann, and Adina Magda Florea. Give agents some rest: A resource-oriented abstraction layer for internet-scale agent environments. In *Proceedings of the 16th Conference on Autonomous Agents and MultiAgent Systems*, AAMAS '17, pages 1502–1504, Richland, SC, 2017. International Foundation for Autonomous Agents and Multiagent Systems.
8. Andrei Ciortea, Simon Mayer, and Florian Michahelles. Repurposing manufacturing lines on-the-fly with multi-agent systems for the web of things. In *Proceedings of the 17th Conference on Autonomous Agents and MultiAgent Systems*, page xx (in press), 2018.
9. Andrei Ciortea, Antoine Zimmermann, Olivier Boissier, and Adina Magda Florea. Hypermedia-driven Socio-technical Networks for Goal-driven Discovery in the Web of Things. In *Proceedings of the 7th International Workshop on the Web of Things (WoT)*. ACM, 2016.
10. Richard Cyganiak, David Wood, and Markus Lanthaler. RDF 1.1 Concepts and Abstract Syntax, W3C Recommendation 25 February 2014. W3C Recommendation, World Wide Web Consortium (W3C), February 25 2014.
11. M. Duerst and M. Suignard. Internationalized Resource Identifiers (IRIs). RFC 3987 (Proposed Standard), January 2005.
12. Roy T. Fielding and Richard N. Taylor. Principled design of the modern web architecture. *ACM Trans. Internet Technol.*, 2(2):115–150, May 2002.
13. Roy Thomas Fielding. *Architectural styles and the design of network-based software architectures*. PhD thesis, University of California, Irvine, 2000.
14. Foundation for Intelligent Physical Agents. FIPA ACL Message Structure Specification. http://www.fipa.org/specs/fipa00061/SC00061G.html, 2002. Document number: SC00061G.
15. Foundation for Intelligent Physical Agents. FIPA Agent Message Transport Protocol for HTTP Specification. http://www.fipa.org/specs/fipa00084/SC00084F.html, 2002. Document number: SC00084F.
16. Foundation for Intelligent Physical Agents. FIPA Agent Management Specification. http://www.fipa.org/specs/fipa00023/SC00023K.html, 2004. Document number: SC00023K.

17. Abdelkader Gouaïch and Michael Bergeret. Rest-a: An agent virtual machine based on rest framework. In *Advances in Practical Applications of Agents and Multiagent Systems*, pages 103–112. Springer, 2010.
18. Jomi F. Hübner, Jaime S. Sichman, and Olivier Boissier. Developing Organised Multiagent Systems Using the MOISE+ Model: Programming Issues at the System and Agent Levels. *Int. J. Agent-Oriented Softw. Eng.*, 1(3/4):370–395, December 2007.
19. Sebastian Kaebisch and Takuki Kamiya. Web of Things (WoT) Thing Description, W3C Working Draft 5 April 2018. W3C Working Draft, W3C, April 2018.
20. Martha L. Kahn and Cynthia Della Torre Cicalese. Coabs grid scalability experiments. *Autonomous Agents and Multi-Agent Systems*, 7(1):171–178, Jul 2003.
21. Markus Lanthaler and Christian Gütl. Hydra: A vocabulary for hypermedia-driven web apis. In *Proceedings of the WWW2013 Workshop on Linked Data on the Web*, volume 996 of *CEUR WS*, 2013.
22. Xavier Limon, Alejandro Guerra-Hernandez, and Alessandro Ricci. Distributed Transparency in Endogenous Environments: the JaCaMo Case. In *Proceedings of the 5th International Workshop on Engineering Multi-Agent Systems*. Springer, 2017.
23. Dejan Mitrović, Mirjana Ivanović, Zoran Budimac, and Milan Vidaković. Radigost: Interoperable web-based multi-agent platform. *Journal of Systems and Software*, 90:167–178, 2014.
24. Cesare Pautasso, Olaf Zimmermann, and Frank Leymann. Restful web services vs. "big"' web services: Making the right architectural decision. In *Proceedings of the 17th International Conference on World Wide Web*, WWW '08, pages 805–814, New York, NY, USA, 2008. ACM.
25. Eric Prud'hommeaux and Gavin Carothers. RDF 1.1 Turtle - Terse RDF Triple Language, W3C Recommendation 25 February 2014. W3C Recommendation, World Wide Web Consortium (W3C), February 25 2014.
26. Alessandro Ricci, Michele Piunti, and Mirko Viroli. Environment programming in multi-agent systems: an artifact-based perspective. *Autonomous Agents and Multi-Agent Systems*, 23(2):158–192, Sep 2011.
27. Z. Shelby, K. Hartke, and C. Bormann. The Constrained Application Protocol (CoAP). RFC 7252 (Proposed Standard), June 2014.
28. Munindar P. Singh. Information-driven interaction-oriented programming: Bspl, the blindingly simple protocol language. In *The 10th International Conference on Autonomous Agents and Multiagent Systems - Volume 2*, AAMAS '11, pages 491–498, Richland, SC, 2011. International Foundation for Autonomous Agents and Multiagent Systems.
29. Dante I Tapia, Sara Rodríguez, Javier Bajo, and Juan M Corchado. FUSION@, a SOA-based Multi-Agent Architecture. In *International Symposium on Distributed Computing and Artificial Intelligence 2008 (DCAI 2008)*, pages 99–107. Springer, 2009.
30. Mirko Viroli, Alessandro Ricci, and Andrea Omicini. Operating instructions for intelligent agent coordination. *The Knowledge Engineering Review*, 21(1):4969, 2006.
31. Danny Weyns, Andrea Omicini, and James Odell. Environment as a first class abstraction in multiagent systems. *Autonomous agents and multi-agent systems*, 14(1):5–30, 2007.
32. S. Willmott, J. Dale, B. Burg, P. Charlton, and P. O'Brien. Agentcities: A worldwide open agent network. *Agentlink News*, 8, 2001.