Engineering Scalable Distributed Environments and Organisations for MAS

Alessandro Ricci
DISI, University of Bologna
Cesena, Italy
a.ricci@unibo.it

Andrei Ciortea
University of St. Gallen, Switzerland
Univ. Côte d'Azur Inria CNRS, France
andrei.ciortea@unisg.ch

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

Olivier Boissier
Univ. Lyon, MINES Saint-Etienne
CNRS Lab. Hubert Curien, France
olivier.boissier@emse.fr

Rafael H. Bordini*
School of Technology, PUCRS
Porto Alegre, RS, Brazil
rafael.bordini@pucrs.br

Jomi F. Hübner
Federal University of Santa Catarina
Florianopólis, SC, Brazil
jomi.hubner@ufsc.br

ABSTRACT

In MAS programming and engineering, the environment and the organisation can be exploited as first-class design and programming abstractions besides the agent one. A main example of a platform implementing this view is JaCaMo, which allows the programming of a MAS in terms of an organisation of cognitive agents sharing a common artifact-based environment. However, MAS models and platforms in general do not provide a satisfactory approach for MAS developers to uniformly deal with distribution at multiple dimensions — agent, environment, and organisation. Typically, environments are either centralised in a single node, or composed by parts that run on different nodes but with a poor support at the programming and execution levels to deal with that. In this paper, we tackle this problem by proposing a model for engineering world-wide distributed environments and organisations for MAS. The approach integrates the A&A (Agents and Artifacts) conceptual model with a web/resource-oriented view of distributed systems as proposed by the REST architectural style. To evaluate the approach, an extension of the JaCaMo open-source platform has been developed implementing the proposed model.

KEYWORDS

Multi-Agent Systems Engineering; Artifact-based Environments; Distributed Systems; JaCaMo

ACM Reference Format:
Alessandro Ricci, Andrei Ciortea, Simon Mayer, Olivier Boissier, Rafael H. Bordini, and Jomi F. Hübner. 2019. Engineering Scalable Distributed Environments and Organisations for MAS. In *Proc. of the 18th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2019), Montreal, Canada, May 13–17, 2019,* IFAAMAS, 9 pages.

1 INTRODUCTION

The concept of environment can be used as first-class abstraction in AOSE and MAS programming to model and design a software layer providing agents different kinds of functionalities [38]: to mediate the access to the external environment; to create an abstraction layer to model/design/represent (non autonomous) resources & services; to enable forms of mediated interaction, communication and coordination among agents, enriching those approaches based on direct communication (e.g., agent communication laguages based on speech acts).

This viewpoint can be brought from design down to programming level, by means of conceptual models that define environmental abstractions besides to agent abstraction to design and program a MAS. An example of conceptual model supporting this view is A&A (Agents and Artifacts), where the environment is modelled in terms of workspaces and artifacts [30]. Then, in the context of Multi-Agent Oriented Programming (MAOP) [4], the environment is one of the multiple dimensions including also agents and organisation used to design and develop a MAS.

A main example in literature is JaCaMo [4], a platform that emerged from the combination of Jason [7] for programming individual agents, CArtAgO [33] for the environment, and MOISE [22] for organisation, all providing first class abstractions at each of the three dimensions. Experience with the platform shows that the synergies between the three dimensions provide a powerful approach for the development of complex multi-agent systems. All the 3 dimensions in JaCaMo were conceived to be fully distributed, thus allowing the development of massive multi-agent systems. However, in practice, the implementation does not provide all the required support for distribution, and to the best of our knowledge, no agent platform that is similarly expressive in regards to all those levels of abstractions and with support for transparent distribution at all the levels is available.

The focus of this paper is on distribution and openness of agent environments, which are essential properties of MAS at scale. MAS environments need to be distributed for enabling physically and logically use cases, for instance regarding world-wide autonomous manufacturing systems [10]. At the same time, adopting central properties of the Web architecture, they should be *open*, providing interfaces to the outside world that make them usable by any type of (distributed/remote/external) agent, so that shared environments can function as an interoperability mechanism for heterogeneous MAS. When dealing with distributed MAS, the model adopted for the environment level should properly support distribution as well. In the MAOP perspective, this holds for every dimension, including agents and organisations, and the distribution

*Currently on a sabbatical leave funded by CAPES at the Universities of Genoa and Oxford.

Proc. of the 18th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2019), N. Agmon, M. E. Taylor, E. Elkind, M. Veloso (eds.), May 13–17, 2019, Montreal, Canada. © 2019 International Foundation for Autonomous Agents and Multiagent Systems (www.ifaamas.org). All rights reserved.

model adopted for one dimension may affect the one of another dimension.

The problem tackled in this paper is that current models and technology supporting environment as first-class abstraction do not provide a satisfying approach for engineering distributed environments for MAS. Typically, application environments are either centralised in a single node, or composed by parts that run on different nodes but with a poor support at the programming and execution levels to deal with that. In A&A model [30] and in CArtAgO/JaCaMo implementation, an environment can be distributed in multiple workspaces on different nodes, however: *(i)* there is no structure, so that a MAS is a flat sea of (unrelated) workspaces; *(ii)* low-level details about the network address of the workspaces is exposed in agent API and programs. The same applies for the organisation dimension. In MOISE, the organisational basis of JaCaMo, the concept of organisation is implicitly distributed, given that MOISE organisation can be managed by artifacts controlling each group and scheme individually. Unfortunately, no efficient implementation of transparent distribution for MOISE organisations is currently available.

In this paper we tackle this problem by proposing a distribution model/approach for environments and organisations in MAS, taking core principles from the web and resource-oriented environments as both inspiration and underlying fabric technologies to satisfy the following requirements/features: *(i) scalability* — our distribution mechanisms for MAS environments and organisations should in principle be able to scale to the size of the Web. This implies that we avoid tight coupling of individual components and that we make use of paradigms that support scalability, such as stateless interactions; *(ii) interoperability* — The platform can be opened to agents, artifacts, and organisations written in different languages and technologies (other than agreeing to a common meta-model); *(iii) flexibility & evolvability* — our distribution model needs to be as flexible as possible. The main reason for this requirement is that components of large-scale systems need to be enabled to evolve independently to foster evolvability of the system as a whole. Consequently, agents that make use of distributed environments should not depend on out-of-band information to understand and navigate them; rather, this information should be provided by the environment itself, just like in HATEOAS [13].

The outcome of this work is an effective model and tool to design and build distributed large-scale MAS organisations living in distributed MAS environments. We believe that this could be an important contribution to advance the state-of-the-art in agent programming platforms in the direction of increasing the uptake of the paradigm. The remainder of the paper is organised as follows: Sect. 2 provides a brief background about the state-of-the-art, focusing in particular on the A&A model and JaCaMo, which are extended in this paper; Sect. 3 describes the proposed model about distributed artifact-based environments and in 4 its mapping onto the Web; Sect. 5 describes current implementation and first evaluation of the model using a real-world manufacturing case study; Sect. 6 discusses the benefits of the approach and current limitations as well, envisioning the next steps that will be taken about; Finally Sect. 7 and Sect. 8 complete the paper discussing related work and providing concluding remarks.

2 BACKGROUND

The reference model suggested in [38] for approaches implementing application environments as first-class abstraction abstracts from distribution of the multiagent system. In the case of distributed application (distributed MAS), the application environment typically has to be distributed over the processors of the application nodes. For some applications, the same functionalities of the application environment are deployed on each node. For other applications, specific functionalities are deployed on different nodes (e.g., when different types of agents are deployed on different nodes). Some functionalities provided by the application environment may be limited to the local context (e.g., observation of the deployment context may be limited to resources of the local deployment context); other functionalities may be integrated (e.g., neighbouring nodes may share state). Integration of functionalities among nodes typically requires additional support. Such support may be provided by appropriate middleware. Examples are support for message transfer in a distributed setting (e.g., [3]) and support for a distributed pheromone infrastructure (e.g., [31]). Actually, the complexity of such as support – which is based on some distribution model – depends on the abstraction layer that the environment is meant to provide.

In the A&A model [30] and concrete platforms such as CArtAgO [33] and JaCaMo [4], the application environment of a MAS can be designed and programmed in terms of non-autonomous computational entities called *artifacts* representing resources and tools shared by agents to do their job. From the agent point of view, an artifact represents a piece of environment encapsulating some functionality and providing an interface to work with it. The interface consists in a set of operations – i.e., actions from the agent point of view – and a set of *observable properties* – i.e., information items that an agent can perceive and reason about, in terms of beliefs. Artifacts are created (by agents) and stored in *workspaces*, functioning as "containers" each one running on a specific host (network node) of the MAS. From an agent point of view, a workspace represents a local context of work, where to create and share resources related to some possibly cooperative activity. Supporting distribution in this case implies allowing agents to work concurrently in multiple contexts (workspaces), possibly running on different nodes. Differently from the case of environments for situated MAS [36] – where actions and perceptions are typically related to some kind of context which is local to the agent – in this case agents may need to act upon and perceive artifacts that are logically and physically on a different nodes.

Fig. 1 shows a simple example using JaCaMo, which fully implement A&A. In JaCaMo, a Jason agent can be spawn on some node and the *join* one or multiple workspaces, either belonging to its node or on a remote node. In the example, an agent running on host Node A works with artifacts of workspaces `main`, `w0` and `w1`, the latter hosted by a different node, Node B. In Fig. 2 shows the Jason agent code. In the plan for achieving the goal `!test_workspaces` the agent first creates a local `w0` workspace, creating inside a Counter artifact and using it (goal `!do_stuff`); then, the agent joins the remote `w1` workspace, then creating and using a Counter artifact – named `c1` – also there. Counter artifacts are simple counters, providing an inc action to increment the

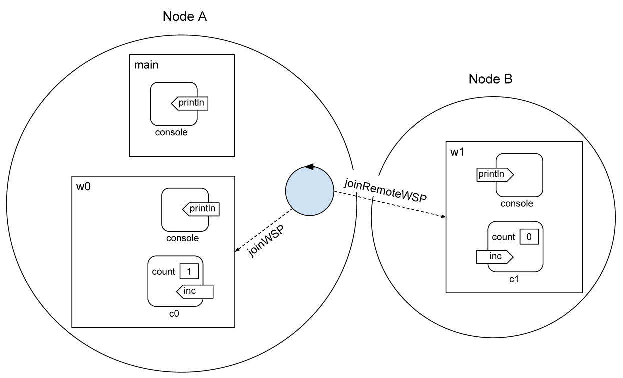

Figure 1: Agents and Workspaces in A&A

```
+!test_workspaces
  <- createWorkspace("w0");
     joinWorkspace("w0",WspID0);
     println("hello in ",WspID0);
     !do_stuff("c0");
     joinRemoteWorkspace("w1","192.168.1.100",WspID1);
     println("hello there ",WspID1);
     !do_stuff("c1").

+!do_stuff_on(Name)
  <- makeArtifact(Name,"Counter",[],Id);
     focus(Id);
     inc [artifact_id(Id)].

+count(V)
  <- println("count changed: ",V).
```

Figure 2: JaCaMo example.

count and a count observable property, keeping track of the current count.

This simple distribution model has two main weaknesses: *(i)* no support for *location transparency* — in the agent code, a different API is provided to work with local and remote workspaces; *(ii)* no support for *complex large-scale MASs* — large-scale MASs can be implemented in terms of large flat set of unrelated workspaces spread over different nodes.

We look forward a model and a corresponding infrastructure that would allow to raise the level of abstraction used to define and manage distributed, possibly large-scale and scalable, artifact-based environments. This includes the definition of a topology that: *(i)* would make it possible for agents to refer and work with artifacts and workspaces in a distributed environment abstracting from they physical network address; *(ii)* would support the design of complex environments, eventually composed by a dynamic and large set of workspaces running on different Internet nodes, and *(iii)* would enable a level of interoperability and openness by exploiting as much as possible the *web* as the underlying architecture and infrastructure on which the environment is running.

3 A MODEL FOR SCALABLE DISTRIBUTED ARTIFACT-BASED ENVIRONMENTS

We adopt a hierarchical model for structuring a distributed environment, using workspaces as basic unit/module of this structuring. In A&A, the concept of *workspace* has been originally introduced as a kind of container for artifacts [30, 33], a *context* of work for agent activities, meaning that agents can be in execution somewhere and join a workspace in order to share that context of work. However, instead of seeing a MAS as a flat set of workspaces, distributed in multiple nodes, each one running an independent piece of the MAS, in the approach proposed here a distributed MAS is modelled as a dynamic *hierarchy* of workspaces. See Figure 3.

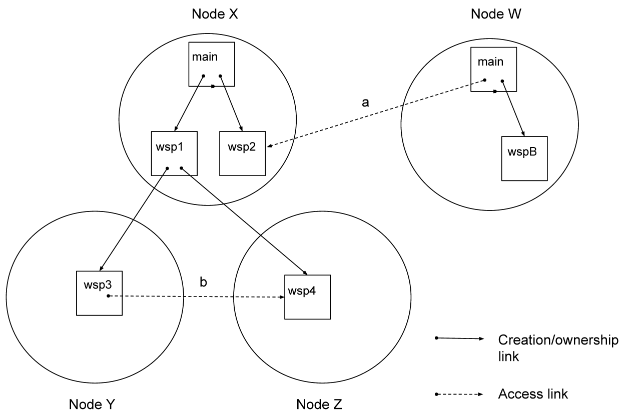

Figure 3: Distributed environment overview.

The hierarchy, however, is logical rather than physical: the workspaces of a MAS are (possibly) distributed into different nodes, and linked together so that a workspace has always a parent workspace (except the main/root workspace). In this view, workspaces can be used as a recursive concept to explicitly structure a complex environment into nested contexts, possibly at a different level of abstraction, from a whole coarse grained view to fine-grained contexts. Many real-world applications can be used to illustrate clearly why this approach is useful; consider for example a smart building, a factory floor, a smart city, and so forth, all of which will naturally have a hierarchical structure for the relevant workspaces.

Each workspace has a unique identifier wsp_{id} at the MAS level and a local logical name wsp_n (e.g., wsp0) which is unique with respect to the parent workspace. Analogously to workspaces, an artifact created within a workspace has an unique global identifier art_{id} at the MAS level and a local logical name art_n (e.g., myBlackboard) which is unique at the workspace level.

In fact, the model allows for two types of links among workspaces. The first type is given by *creation* links, relating a parent workspace to child workspaces. This link is created when an agent instantiates a new workspace (see 3.1), which happens always with the specification of a parent workspace. The second type is given by *access* links, relating two workspaces so that a workspace can be referenced by multiple paths, possibly belonging to different multi-agent systems and hierarchies. For instance, an access link could be created between workspace main.wsp1.wsp4 and main.wsp1.wsp3.b. This link can be created by an agent to associate two existing workspaces, which could even belong to different multi-agent systems. This is useful in particular when an agent is running in a MAS but needs to access resources of another MAS. For instance, consider an agent running on a mobile entering into a smart building.

3.1 From the Agent Perspective

Agents are spawned (or enter a MAS) in a specific workspace (their home), which typically does not change during their lifetime within that MAS, except in the case of mobile agents. Once a MAS is entered, an agent can work concurrently in multiple workspaces of the MAS by simply joining them. A joinWSP action is uniformly used to join any workspace of the MAS — either local or remote — by simply giving a reference to it as argument. The same workspace can be identified using different paths, depending on the links. Once a workspace is joined, the agent can interact with all the artifacts of that workspace. The example shown in Sect. 2 becomes:

```
+!test_workspaces
  <- createWSP(main.w0);
     joinWSP(main.w0,WspID0);
     println("hello in ",WspID0);
     !do_stuff("c0");
     joinWSP(main.w1,WspID1);  // this is remote
     println("hello there ",WspID1);
     !do_stuff("c1").
```

Besides joining, an agent can create a new workspace by means of a createWSP action, specifying the identifier of the parent workspace, the logical name to be used for the new workspace, and the Internet node that should host it (using uniform identifiers / logical names also for nodes i.e., no low-level networking info such as IP addresses)[1]. If no node is specified, the node would be the same as that of the parent workspace. Workspaces can be removed by agents by means of a removeWSP action, specifying the full name/path of the workspace to be removed. An agent can link two different workspaces by means of a linkWSP action, specifying the identifier of the two workspaces (linking and linked) and the logic name used in the linking workspace to refer to the linked one.

Finally, in the A&A model, any external action available to an agent must be provided by some artifact. Accordingly, the actions available to agents to work with workspaces are provided by a predefined artifact called wsp, which is available in every workspace by default (so there are multiple instances). The observable properties of wsp are used to store and make it observable to agents the dynamic information about the workspace, in particular the set of available artifacts, the set of agents that have their home there, the set of agents that joined the workspace, the set of children and linked workspaces.

[1]This node must have the infrastructure installed and running.

3.2 Formalisation

We conclude the section with a formal presentation of the proposed model, aimed at presenting the ideas unambiguously. A MAS can be represented as a tuple:

$$mas = \langle mas_{id}, Ag, Art, Wsp \rangle$$

where Ag is a set of agents, Art a set of artifacts, and Wsp the set of workspaces. Each agent $ag \in Ag$ can be defined by a tuple:

$$ag = \langle ag_{id}, wsph_{id}, \{wspj_{id} : ag_n, \ldots\} \rangle$$

where ag_{id} is the agent unique identifier at the MAS level, $wsph_{id}$ is the identifier of the agent home workspace, and $\{wspj_{id} : ag_n, \ldots\}$ is the dynamic set of workspaces that the agent joined., each element including the workspace identifier and the logical name used by that agent within that workspace. Each artifact $art \in Art$ can be defined by a tuple:

$$art = \langle art_{id}, wsp_{id} : art_n \rangle$$

where art_{id} is the artifact unique identifier at the MAS level and $wsp_{id} : art_n$ is the identifier of the workspace hosting the artifact, including its local name. Each workspace $w \in Wsp$ can be defined by a tuple:

$$w = \langle wsp_{id}, parent_{id} : wsp_n, \{ch_{id} : ch_n, \ldots\}, \{mas_{id} : lnk_{id} : lnk_n, \ldots\} \rangle$$

where wsp_{id} is the workspace unique identifier at the MAS level, $parent_{id} : wsp_n$ is the identifier of the parent workspace along with the local name as a child workspace, $\{ch_{id} : ch_n, \ldots\}$ is the set of children workspaces, and finally $\{mas_{id} : lnk_{id} : lnk_n, \ldots\}$ is the set of linked workspaces. The set of artifacts of a workspace always include the wsp artifact. Artifacts in general can have a dynamic set of observable properties, which can be perceived by agents focusing on them. The observable state of the wsp artifact includes the following properties::

$$wsp_{state} = \langle Art, HAg, JAg, CWsp, LWsp \rangle$$

where:

$$Arg = \{\text{artifact}(art_n, art_{id}), \ldots\}$$

is the set of observable properties related to hosted artifacts.

$$HAg = \{\text{hosted\_agent}(ag_n, ag_{id}), \ldots\}$$

is the set of observable properties related to hosted agents.

$$JAg = \{\text{joined\_agent}(ag_n, ag_{id}), \ldots\}$$

is the set of observable properties related to agents that joined the workspace.

$$CWsp = \{\text{child\_wsp}(wsp_n, wsp_{id}), \ldots\}$$

is the set of observable properties related to children workspaces.

$$LWsp = \{\text{linked\_wsp}(wsp_n, wsp_{id}, mas_{id}), \ldots\}$$

is the set of observable properties related to linked workspaces.

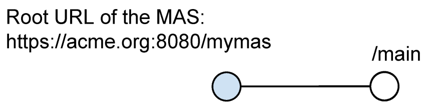

Figure 4: MAS web resource.

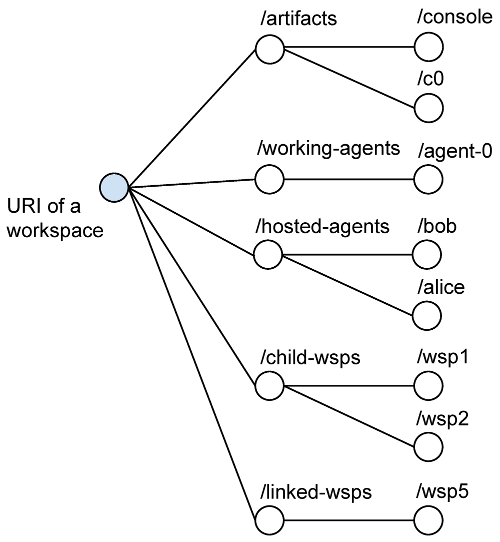

Figure 5: A workspace Web resource.

4 MAPPING ONTO THE WEB

The distribution model described in Section 3 is mapped onto the Web such that every element of a MAS — workspaces, agents, artifacts — is represented (and accessible) as a Web resource, as proposed by the REST architectural style [13].

The MAS itself is represented as a Web resource represented by the logical path http://<node>/{mas-name}, and is the root of the logical resource hierarchy shown in Fig. 4. This logical path – which is meant to be used in programs – is mapped by the platform itself to an IRI, as an immutable identifier generated and managed by the JaCaMo platform. Every entity of the MAS – agents, artifacts, workspaces – has both an IRI generated by the platform and one or multiple logical paths to access the entity as a web resource.

The MAS web resource includes the sub-resource /main pointing to the root (main) workspace of the MAS. A GET request on this sub-resource gets a representation of the main workspace.

Each workspace is represented as a Web resource with the hierarchical structure shown in Fig. 5. This hierarchy includes:

- /artifacts, container for artifacts located in this workspace;
- /working-agents, listing the sessions related to agents who joined this workspace;
- /hosted-agents, listing the agents that entered the MAS using this workspace as host/home workspace;

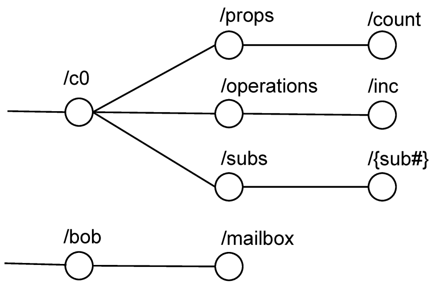

Figure 6: Web resources for artifacts and agents

- /child-wsps, listing the workspaces that are children of this workspace;
- /linked-wsps, listing the names of the workspaces that are linked by this workspace.

An agent can enter the MAS by issuing a POST request on the hosted-agents sub-resource, including in the payload the information possibly needed for its authentication. If the request is accepted, the workspace becomes the *home* workspace for the agent and a new working session is created, listing this session in /working-agents. Once entered, an agent can crawl the web of workspaces accessing to child-wsps and linked-wsps, to eventually join and work with other workspaces.

The Web API of a workspace web resource includes the interface to work with artifacts and workspaces, wrapping the functionalities provided by the wsp artifact. In particular an agent can:

- join a workspace by doing a POST on /working-agents;
- create a new artifact in the workspace by doing a POST on /artifacts, including the logical name and the parameters in the payload. The artifact can be disposed by issuing a DELETE on the corresponding web resource;
- create a new workspace by doing a POST on /child-wsps, specifying in the payload the logical name of the workspace. By default the workspace is created locally, on the same node of the parent workspace; instead, if the workspace is meant to be created on a remote node, the node IRI must be specified in the request[2];
- link an existing workspace to this workspace by issuing a POST on /linked-wsps, specifying in the payload the IRI of the workspace to be linked and the logical name to be used to refer to it from this workspace.

Fig. 6 (top) shows the resource hierarchy related to the artifact usage interface, in terms of properties /props and operations /operations. In the example, c0 is a counter, featuring a count observable property and an inc operation. The sub-resource /subs keeps track of the subscriptions related to agents who are focusing (observing) the artifact. The Web API for an agent to work with an artifact includes:

- triggering the execution of an operation (i.e., executing an action) by performing a POST on the specified operation,

[2]In the case of remote workspace creation, a JaCaMo infrastructure runtime must be installed and running on the node where the workspace is meant to be created.

e.g. /operations/inc, specifying in the payload the parameters;
- focusing the artifact by issuing a POST on /subs, that is, creating a new subscription to receive updates about observable properties and observable events.

Finally, the Web API include the possibility to send a message to an agent by doing a POST on the /mailbox sub-resource which is included in the agent resource representation (Fig. 6, bottom), specifying the details of the speech act in the payload.

5 IMPLEMENTATION AND FIRST EVALUATION

In this section, we first report on our implementation of the proposed distribution model, and then present concrete scenarios for applying this model in the context of industrial automation domain — for programming world-wide agent-based manufacturing systems.

5.1 Implementation

To validate the proposed distribution model, we implemented an extension of the JaCaMo platform [4]. In particular, we extended the CArtAgO framework to allow agents to join and work in remote workspaces in a seamless manner through the Web.

Each JaCaMo node[3] in our extension now exposes an HTTP API that allows remote clients to interact with local CArtAgO workspaces. The HTTP API follows the mapping presented in Section 4. To this end, we extended CArtAgO elements — workspaces, artifacts — to use IRIs as unique identifiers. The IRIs are generated automatically and are *minted* to elements: if an IRI is assigned to an element that is later destroyed, the IRI cannot be reassigned to a new element.

When an agent joins a workspace using the joinWorkspace operation, the logical path of the workspace is passed as a parameter and the platform translates it to an IRI. If the IRI identifies a remote workspace, the join operation is translated to an HTTP request (see Section 4) that is sent to the remote JaCaMo node in order to create a remote CArtAgO session in that workspace for the requesting agent. If the requesting agent is the first agent on the local node to join that remote workspace, a local workspace is created that acts as a facade for the remote workspace. Future agents joining the remote workspace from the same node will automatically join the local facade workspace as well (the operation is handled automatically by the platform). The facade workspace is used as a proxy: all operations performed by agents within the facade workspace are translated to HTTP requests (see Section 4) and are forwarded to the remote workspace; likewise, all notifications from the remote workspace are routed back to agents through the facade workspace.

To route remote notifications to agents, the payload of the HTTP request issued to join a remote workspace includes a callback IRI that identifies the requesting agent's CArtAgO session in the local facade workspace. This callback IRI is then used to route notifications from the remote workspace to the agent through the facade workspace. For instance, if an agent focuses on an artifact in the remote workspace, it will receive notifications whenever the artifact's observable properties change or an observable event is generated. These notifications are routed through the local facade workspace and handled in the same way as local CArtAgO notifications. From an agent's viewpoint, interaction with remote CArtAgO workspaces is thus seamless.

[3]A JaCaMo node is a JaCaMo runtime in execution, that typically coincides with a host/machine - but in principle you can run multiple nodes on the same host/machine using different Internet ports. A JaCaMo node can host multiple workspaces. Differently from the previous version of JaCaMo, the node concept is not meant to appear e.g. in the agent code: the idea is that at the modelling/programming level we want to have only workspaces, even in the case of distributed environments.

5.2 Applications in Industrial Automation

We present several application scenarios that demonstrate the usefulness of the proposed distribution model in the context of industrial automation. The scenarios build upon our previous experience with the deployment of a JaCaMo-based MAS in a prototypical production cell [10]. In our previous deployment, the concrete objective was to create highly adaptive production cells that can be reconfigured on-the-fly in order to enable mass customization, i.e. the manufacturing of customized products at mass-production cost. In that deployment, we encapsulated manufacturing devices (e.g., pick-and-place robots) as CArtAgO artifacts, and manufacturing agents programmed in Jason were able to synthesize production plans based on semantic descriptions of artifacts located in their workspaces. We used semantic MOISE [39] organizations to coordinate manufacturing agents.

The application scenarios presented in this section expand this deployment from a single production cell to a world-wide manufacturing system composed of multiple production cells that are distributed geographically and can contain a variety of production equipment. Multiple such geographically distributed manufacturing cells could compete for orders [28], or they could cooperate to produce more complex products collaboratively, forming a "global collective intelligence for manufacturing" [10].

Autonomous Marketplaces for Manufacturing — The core idea behind this application scenario is that – by virtue of our extension to the JaCaMo platform – collaborate in the manufacturing of a product in light of an order workflow that includes non-functional constraints. Concretely, in this scenario, a customer orders a product to be manufactured by our world-wide manufacturing system, but requires that *at least 60%* of the production process value-add is generated in a specific country (this is required to allow labeling a product as *Swiss-made*, for instance) – our marketplace is free to decide where and how the other half of the process value-add is generated. To enable this scenario in our implementation, we would assign individual production cells to a Web of Workspaces that is structured as a geographic hierarchy. Agents can then navigate this hierarchy and ask for price estimates on different levels of the tree (e.g., in Europe; in Switzerland; in St. Gallen), where requests are propagated toward the tree's leaf nodes (i.e. the workspaces and production cells). In our implementation, this would take the form of logical paths that induce this geographic hierarchy, e.g. `europe.ch.st-gallen.cell1`. Note that this scenario is not specific to the concrete non-functional constraint – geography, in this case. Rather, our implementation supports any constraint that can be mapped to an assignment of production cells to (hierarchical) workspaces.

Distributed Manufacturing — After the ordering process has been completed, our autonomous production cells need to produce the ordered product, in a distributed way. To efficiently steer this process, a production engineer who is using our system can make use of domain-specific taxonomies that are superimposed on the Web of Workspaces and structure them – this structure can, for instance, be based on the ANSI/ISA-95 standard, to structure workspaces according to the type of manufacturing process (*batch/continuous/repetitive*), or on the DIN 8580 norm to structure workspaces according to the type of industrial action they perform (DIN 8580 defines six main types of industrial actions and a hierarchy of actions underneath each of these types). In our implementation, these taxonomies can be expressed as logical paths between a hierarchy of workspaces where, again, the leaf workspaces contain the concrete production cells. This would allow to organize a world-wide manufacturing system in a way that enables production engineers to use their own domain-specific vocabulary and taxonomies when programming the system, and specifically to directly access production cells with the desired capabilities.

Distributed Failure Management — During execution of the production process, failures of individual production devices or entire production cells might occur. In these cases, the production plan might need to be adapted, subject to the non-functional constraints that are specified in the order. Ideally, local supervising agents of a deficient workspace would be notified of these problems and could decide whether they can cope with the problem locally or whether it needs to be escalated to a higher level. Our implementation supports this scenario since notifications can not only be received by local agents but travel across workspaces, which constitutes a natural escalation mechanism: if the local supervisor cannot handle a problem, agents in hierarchically superior workspaces can take care of it, deciding whether a solution can be found at their level, and whether this solution is consistent with the non-functional requirements that are part of the order in question.

6 DISCUSSION

The distributed manufacturing use case and the three discussed scenarios have been useful to evaluate the effectiveness of the extension described in this paper to A&A and to the JaCaMo platform with realistic challenges. In the following we summarise the key benefits and limitations of the approach, in particular related to MAS programming and MAS execution.

6.1 Impact on MAS Design and Programming

Distribution at the design level — The model makes it possible to explicitly design at a proper level of abstraction how a MAS environment (and a MAS) should structured as a distributed system, abstracting from a low-level network view. Workspaces are no more just containers but an abstraction to describe (and deploy) the environment (MAS) structure at a logical (vs. physical) level.

Location transparency and level of abstraction in agent programs — Distributed resources (artifacts, workspaces) can be referenced in agent programs abstracting from their physical location. An agent can refer to a remote workspace (e.g., to join) without specifying its network address.

Agent heterogeneity — The mapping on the Web fabric makes it possible for agents written in different languages and technologies to enter and work in a MAS based on artifact-based environments, without the need of developing any further specific integration technology.

Distributed organisations — In JaCaMo the part of the platform implementing the organisation based on MOISE – the organisation infrastructure – is implemented as a set of artifacts [21]. Then, the new model allows for deploying organisations that are distributed not only in terms of the involved agents (running on different nodes), but also in terms of the infrastructure implementing the management of the organisation. Besides, it is an enabling conceptual layer that could trigger an extension of the organisation meta-model to explicitly consider the distribution model when defining groups, social schemes, etc.

6.2 Impact on MAS Execution

Scalability — As discussed in literature [37], the engineering of agent environments for large-scale MAS accounts for several challenges, including the development of *scalable* structure and access to resources, scalable communication and interaction models. In this regards, the new model makes it possible to describe large-scale environments and MAS as a dynamic and structured web of workspaces, instead of a large flat set of unrelated workspaces. This structure can be dynamically expanded (or contracted) by means of agents creating (or deleting) workspaces, including linking existing workspaces possibly belonging to different MASs. There are no centralisations that would undermine scalability. A workspace still represents for agents a context of work—running on some specific node; however, a complex workspace can be structured e.g. as a hierarchy where some levels are on the same node and children workspaces are distributed on different nodes.

Principled failure handling & Resiliency — The hierarchical structure makes it possible to implement (at the infrastructure level) a supervision strategy based on the *ownership means commitment* principle [25], so that a parent workspace is responsible for managing failures of children workspaces. In a MAS setting, this means that if a workspace (and related hosted resources) becomes unavailable – because of e.g. a crash of the node or network problems – this event should be made observable to agents running in the parent workspace, so as to properly react. In the extended model, a failure of a child workspace is made observable as an event generated by the wsp artifact of the parent workspace. Agents can enact some resilience policy by focusing this artifact and then promptly react to the failure event.

6.3 Limitations and Next Steps

Current model does not fully exploit *hypermedia* as implied by REST-full architectures [13] and this negatively impact of the level of *openness* of the approach, which can be largely improved. A next step in this direction is to support semantic hypermedia for dynamic discovery of artifact interfaces, decoupling agents from artifacts through hypermedia controls. In literature, a first proposal in this direction can be found in [8]. This implies defining a more meaningful model and support for a *semantic* layer, which is crucial to achieve a full-fledge support for openness. For instance, when programming agents using logical paths for environments, we currently blend identification (the path itself) and semantics (the environment semantic model / topology is embedded in the path). This is good because it simplifies agent programming, but it also has limitations in terms of representing / semantic modeling of / reasoning about the environment. A next step could be to separate the semantic modeling of environments (for which we could use ontologies) from the uniform identification of environmental abstractions (workspaces, artifacts).

7 RELATED WORK

A first approach for designing distributed artifact-based environments in JaCaMo is proposed in [27]. It is based on hierarchical structuring of workspaces, similarly to the model proposed in this paper. The key different with respect to that work – besides details related to model per se – is about the mapping onto the web, building a distributed resource-oriented architecture for MAS environments, and its evaluation with real-world application scenario (i.e., industrial automation).

Distribution is a main feature of MASs and typically MAS platforms support it in terms of services for agent discovery, communication, and mobility. A main example is JADE [3], implementing the FIPA model and architecture [14]. A FIPA agent platform can be split onto several hosts, each one acting as a *container* of agents, providing a complete runtime environment for agents execution (lifecycle management, message passing facilities, etc.). At least one of these containers acting as the main container, responsible to maintain a registry of all other containers in the same Jade platform through which agents can discover each other. FIPA model/architecture and related MAS platforms typically do not support the environment as a first-class design or programming concept. Besides FIPA, distribution and fault-tolerant techniques are a main point tackled in models, architectures and platforms for massive multi-agent systems [20] and for MAS architectures in general [26]—typically focusing on agents, agent execution, communication, mobility and not considering the issue of distributed application environments.

In cognitive agent programming platforms (see [5, 6] for surveys), the environment is either centralised or, typical case, distributed among agents so that each agent has a local view and sphere of influence on it. That is: there is no middleware/infrastructure support to represent and manage the environment as a single distributed entity.

The engineering of agent environments for large-scale Multi-Agent Systems is a main challenge discussed in literature [37, 38]. In particular four requirements have been identified [37]: *(i) scalable structure* – distributing the computation and state of the agent environment multi-level or hierarchical, multi-stage or dynamic structures; *(ii) access to resources* – designing monitoring, trust, and security aspects so that the cost induced by managing access to resources does not become a bottleneck; *(iii) scalable communication* – providing means for communication between agents that do not involve any central point of access or control; *(iv) interaction model* – providing agents with efficient means for perceptions, actions, and interactions. Examples of approaches in literature of distributed agent environments for large-scale MAS include [18, 34]. In the former, the agent environment is decomposed into independent interaction spaces, each of which defines explicitly local environmental rules. In the latter, holonic modeling is applied in the domain of large-scale traffic simulation, so that the environmental processes apply only locally.

In agent environment literature, a main related work is the EIS (Environment Interface Standard) initiative and technology [2]. EIS objective is to define a generic environment interface standard, providing a support for connecting agents written in different agent programming languages and platforms to same shared environments. EIS does not constrain the type of environments that can be considered—this could range from being an existing game environment like Unreal Tournament to a simulation environment like the ones used in Multi-Agent Programming Contest [1]. As far authors' knowledge, it does not provide any specific support for distributed environments.

Finally, related work includes approaches using the Web as an infrastructure for distributed MASs. Early work was influenced by service-oriented architectures (SOA) based on the WS-* standards (SOAP, WSDL, UDDI etc.) [16, 23, 24, 35]. However, Web service design has evolved drastically over the past decade. It is now well recognized that WS-* services use the Web merely as a transport layer [32]. Based on similar ideas, FIPA proposed a specification for using HTTP as a transport protocol for messages exchanged among agents [15], which was implemented by several FIPA-compliant platforms (e.g., [11, 12, 19]). The problem with systems that use the Web merely for transport is that they are misaligned with the Web architecture (see Section 6.5.3 in [13] for a detailed discussion). Consequently, such systems make limited use of the existing Web infrastructure and its future extensions. More recent approaches for engineering Web-based MASs have turned to resource-oriented architectures (ROA) based on REST (e.g., [17, 29]), which use the Web as an application layer and are better aligned with the Web architecture. These approaches do not consider the environment as a first-class abstraction in MAS.

8 CONCLUSIONS

In this paper we considered the problem of designing and programming distributed and scalable environments for MAS where the environment concept is exploited as first-class design and programming abstraction. In particular, we focused on artifact-based environments as defined by the A&A model and implemented by platforms like CArtAgO and JaCaMo, and we proposed an extension for engineering complex distributed environments as hierarchies of workspaces distributed over the web, making MAS resources (artifacts, agents) accessible as web resources. In JaCaMo, the organisation management infrastructure implementing the MOISE model is based on artifacts; therefore the proposed extension impacts also on the organisation dimension, allowing for deploying distributed MAS organisations based on a distributed organisation management infrastructure.

We consider the contribution of this paper as the first enabling step for moving towards a full-fledge REST-ful architecture for artifact-based environments, exploiting the hypermedia and semantic web layer to fully achieve openness at the MAS environment level, toward a *hypermedia MAS* perspective [9].

REFERENCES

[1] Tobias Ahlbrecht, Jürgen Dix, and Niklas Fiekas. 2018. Multi-agent programming contest 2017 - The twelfth edition of the MAPC. *Ann. Math. Artif. Intell.* 84, 1-2 (2018), 1–16.

[2] Tristan M. Behrens, Koen V. Hindriks, Rafael H. Bordini, Lars Braubach, Mehdi Dastani, Jürgen Dix, Jomi Fred Hübner, and Alexander Pokahr. 2012. An Interface for Agent-Environment Interaction. In *Programming Multi-Agent Systems - 8th International Workshop, ProMAS 2010, Toronto, ON, Canada, May 11, 2010. Revised Selected Papers (Lecture Notes in Computer Science)*, Rem W. Collier, Jürgen Dix, and Peter Novák (Eds.), Vol. 6599. Springer, 139–158.

[3] Fabio Bellifemine, Giovanni Caire, Agostino Poggi, and Giovanni Rimassa. 2008. JADE: A software framework for developing multi-agent applications. Lessons learned. *Information & Software Technology* 50, 1-2 (2008), 10–21.

[4] Olivier Boissier, Rafael H. Bordini, Jomi Fred Hübner, Alessandro Ricci, and Andrea Santi. 2013. Multi-agent oriented programming with JaCaMo. *Sci. Comput. Program.* 78, 6 (2013), 747–761.

[5] Rafael H. Bordini, Mehdi Dastani, Jürgen Dix, and Amal El Fallah-Seghrouchni (Eds.). 2005. *Multi-Agent Programming: Languages, Platforms and Applications.* Multiagent Systems, Artificial Societies, and Simulated Organizations, Vol. 15. Springer.

[6] Rafael H. Bordini, Mehdi Dastani, Jürgen Dix, and Amal El Fallah-Seghrouchni (Eds.). 2009. *Multi-Agent Programming, Languages, Tools and Applications.* Springer.

[7] Rafael H. Bordini, Jomi Fred Hübner, and Michael Wooldrige. 2007. *Programming Multi-Agent Systems in AgentSpeak using Jason.* John Wiley & Sons. https://doi.org/10.1002/9780470061848

[8] Andrei Ciortea, Olivier Boissier Boissier, and Alessandro Ricci. 2018. Engineering World-Wide Multi-Agent Systems with Hypermedia. In *6th International Workshop on Engineering Multi-Agent Systems (EMAS 2018).*

[9] Andrei Ciortea, Simon Mayer, Fabien Gandon, Olivier Boissier, Alessandro Ricci, and Antoine Zimmermann. 2019. A Decade in Hindsight: The Missing Bridge BetweenMulti-Agent Systems and the World Wide Web. In *Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems, AAMAS 2019, Montreal, Canada, May 13-17, 2019.* International Foundation for Autonomous Agents and Multiagent Systems.

[10] Andrei Ciortea, Simon Mayer, and Florian Michaelles. 2018. Repurposing Manufacturing Lines on the Fly with Multi-agent Systems for the Web of Things. In *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems (AAMAS '18).* International Foundation for Autonomous Agents and Multiagent Systems, Richland, SC, 813–822.

[11] Oguz Dikenelli. 2008. Seagent MAS platform development environment. In *Proceedings of the 7th International Joint Conference on Autonomous Agents and Multiagent Systems: demo papers.* International Foundation for Autonomous Agents and Multiagent Systems, 1671–1672.

[12] Jose Exposito, Joan Ametller, and Sergi Robles. 2010. Configuring the JADE HTTP MTP. http://jade.tilab.com/documentation/tutorials-guides/configuring-the-jade-http-mtp/. (2010). Accessed: 15.11.2016.

[13] Roy Thomas Fielding. 2000. *Architectural styles and the design of network-based software architectures.* Ph.D. Dissertation. University of California, Irvine.

[14] Foundation for Intelligent Physical Agents. 2002. FIPA Abstract Architecture Specification. http://www.fipa.org/specs/fipa00001. (2002). Document number: SC00001L.

[15] Foundation for Intelligent Physical Agents. 2002. FIPA Agent Message Transport Protocol for HTTP Specification. http://www.fipa.org/specs/fipa00084/SC00084F.html. (2002). Document number: SC00084F.

[16] Nicholas Gibbins, Stephen Harris, and Nigel Shadbolt. 2004. Agent-based semantic web services. *Web Semantics: Science, Services and Agents on the World Wide Web* 1, 2 (2004), 141–154.

[17] Abdelkader Gouaïch and Michael Bergeret. 2010. REST-A: An agent virtual machine based on REST framework. In *Advances in Practical Applications of Agents and Multiagent Systems.* Springer, 103–112.

[18] Abdelkader Gouaïch, Fabien Michel, and Yves Guiraud. 2005. MIC*: A Deployment Environment for Autonomous Agents. In *Environments for Multi-Agent Systems*, Danny Weyns, H. Van Dyke Parunak, and Fabien Michel (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 109–126.

[19] Miguel Escrivá Gregori, Javier Palanca Cámara, and Gustavo Aranda Bada. 2006. A jabber-based multi-agent system platform. In *Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems.* ACM, 1282–1284.

[20] Zahia Guessoum, Jean-Pierre Briot, and Nora Faci. 2005. Towards Fault-Tolerant Massively Multiagent Systems. In *Massively Multi-Agent Systems I*, Toru Ishida, Les Gasser, and Hideyuki Nakashima (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 55–69.

[21] Jomi F. Hübner, Olivier Boissier, Rosine Kitio, and Alessandro Ricci. 2010. Instrumenting multi-agent organisations with organisational artifacts and agents. *Autonomous Agents and Multi-Agent Systems* 20, 3 (01 May 2010), 369–400.

[22] Jomi Fred Hübner, Jaime Simão Sichman, and Olivier Boissier. 2007. Developing organised multiagent systems using the MOISE. *IJAOSE* 1, 3/4 (2007), 370–395.

[23] Michael N Huhns. 2002. Agents as Web services. *IEEE Internet computing* 6, 4 (2002), 93.

[24] Michael N Huhns and Munindar P Singh. 2005. Service-oriented computing: Key concepts and principles. *IEEE Internet computing* 9, 1 (2005), 75–81.

[25] Roland Kuhn, Brian Hanafee, and Jamie Allen. 2017. *Reactive Design Patterns* (1st ed.). Manning Publications Co., Greenwich, CT, USA.

[26] Sanjeev Kumar and Philip R. Cohen. 2000. Towards a Fault-tolerant Multi-agent System Architecture. In *Proceedings of the Fourth International Conference on Autonomous Agents (AGENTS '00).* ACM, New York, NY, USA, 459–466.

[27] Xavier Limón, Alejandro Guerra-Hernández, and Alessandro Ricci. 2018. Distributed Transparency in Endogenous Environments: The JaCaMo Case. In *Engineering Multi-Agent Systems*, Amal El Fallah-Seghrouchni, Alessandro Ricci, and Tran Cao Son (Eds.). Springer International Publishing, Cham, 109–124.

[28] Simon Mayer, Dominic Plangger, Florian Michahelles, and Simon Rothfuss. 2016. UberManufacturing: A Goal-Driven Collaborative Industrial Manufacturing Marketplace. In *Proceedings of the 6th International Conference on the Internet of Things (IoT).* 111–119.

[29] Dejan Mitrović, Mirjana Ivanović, Zoran Budimac, and Milan Vidaković. 2014. Radigost: Interoperable web-based multi-agent platform. *Journal of Systems and Software* 90 (2014), 167–178.

[30] Andrea Omicini, Alessandro Ricci, and Mirko Viroli. 2008. Artifacts in the A&A meta-model for multi-agent systems. *Autonomous Agents and Multi-Agent Systems* 17, 3 (Dec. 2008), 432–456.

[31] H. Van Dyke Parunak, Sven Brueckner, and John A. Sauter. 2005. Digital Pheromones for Coordination of Unmanned Vehicles. In *Environments for Multi-Agent Systems, First International Workshop, E4MAS 2004, New York, NY, USA, July 19, 2004, Revised Selected Papers (Lecture Notes in Computer Science)*, Danny Weyns, H. Van Dyke Parunak, and Fabien Michel (Eds.), Vol. 3374. Springer, 246–263.

[32] Cesare Pautasso, Olaf Zimmermann, and Frank Leymann. 2008. 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).* ACM, New York, NY, USA, 805–814.

[33] Alessandro Ricci, Michele Piunti, and Mirko Viroli. 2011. Environment programming in multi-agent systems: an artifact-based perspective. *Autonomous Agents and Multi-Agent Systems* 23, 2 (Sept. 2011), 158–192.

[34] Sebastian Rodriguez, Vincent Hilaire, and Abder Koukam. 2006. Holonic Modeling of Environments for Situated Multi-agent Systems. In *Proceedings of the 2nd International Conference on Environments for Multi-Agent Systems (E4MAS'05).* Springer-Verlag, Berlin, Heidelberg, 18–31.

[35] Munindar P Singh and Michael N Huhns. 2006. *Service-oriented computing: semantics, processes, agents.* John Wiley & Sons.

[36] Danny Weyns and Tom Holvoet. 2007. A Reference Architecture for Situated Multiagent Systems. In *Environments for Multi-Agent Systems III, Third International Workshop, E4MAS 2006, Hakodate, Japan, May 8, 2006, Selected Revised and Invited Papers (Lecture Notes in Computer Science)*, Danny Weyns, H. Van Dyke Parunak, and Fabien Michel (Eds.), Vol. 4389. Springer, 1–40.

[37] Danny Weyns and Fabien Michel. 2015. Agent Environments for Multi-agent Systems — A Research Roadmap. In *Revised Selected and Invited Papers of the 4th International Workshop on Agent Environments for Multi-Agent Systems IV - Volume 9068.* Springer-Verlag New York, Inc., New York, NY, USA, 3–21.

[38] Danny Weyns, Andrea Omicini, and James J. Odell. 2007. Environment as a First-class Abstraction in Multi-Agent Systems. *Autonomous Agents and Multi-Agent Systems* 14, 1 (Feb. 2007), 5–30.

[39] Alexandra-Madalina Zarafin, Antoine Zimmermann, and Olivier Boissier. 2012. Integrating Semantic Web Technologies and Multi-Agent Systems: A Semantic Description of Multi-Agent Organizations. In *Proceedings of the First International Conference on Agreement Technologies, AT 2012, Dubrovnik, Croatia, October 15-16, 2012 (CEUR Workshop Proceedings)*, Sascha Ossowski, Francesca Toni, and George A. Vouros (Eds.), Vol. 918. CEUR-WS.org, 296–297.