EMAS @ AAMAS 2017

5th International Workshop on Engineering Multi-Agent Systems

Workshop held at AAMAS 2017
8-9 May 2017
Sao Paulo, Brazil

(Informal) Proceedings

Preface

This volume contains the papers to be presented at **EMAS 2017:** 5th International Workshop on **Engineering Multi-Agent Systems** held at AAMAS 2017, International Conference on Autonomous Agents and Multi-Agent Systems - May 8-9, 2017, Sao Paulo

EMAS aims to gather researchers and practitioners in the domains of Agent-Oriented Software Engineering (AOSE), PROgramming Multi-Agent Systems (ProMAS) and Declarative Agent Languages and Technologies (DALT) to present and discuss their research and outcomes in MAS engineering.

The purpose of this workshop is to facilitate the cross-fertilization of ideas and experiences in the various fields to: further our knowledge in MAS engineering and improve the state-or-the art; define research directions that are useful to practitioners; and encourage practitioners to use established methodologies for the development of large-scale MAS.

EMAS 2017 attracted 18 submissions. Each submission was reviewed by 3 members of the Program Committee and finally 14 papers have been accepted to be presented during the workshop. A subset of this paper will be invited to be part of the post-proceedings of the workshop.

Acknowlegments

EMAS Organisers would like to thank the members of the Programme Committee for their work. Thanks to EasyChair guys for their great tool.

May 2017,Sao Paulo

Amal El Fallah-Seghrouchi (LIP6 - University Pierre and Marie Curie, Paris France)
Tran Cao Son (New Mexico State University, Department of Computer Science, USA)
Alessandro Ricci (DISI, University of Bologna, Italy)

Table of Contents

Distributed Transparency in Endogenous Environments: the JaCaMo Case

Xavier Limón[1], Alejandro Guerra-Hernández[1], Alessandro Ricci[2]

[1] Universidad Veracruzana, Departamento de Inteligencia Artificial, Sebastián Camacho No 5, Xalapa, Ver., México 91000
xavier120@hotmail.com, aguerra@uv.mx
[2] DISI, Università di Bologna, Via Sacchi, 3, Cesena, Italia 46100
a.ricci@unibo.it

Abstract. This paper deals with distribution aspects of endogenous environments, in this case, distribution refers to the deployment in several machines across a network. A recognized challenge is the achievement of distributed transparency, a mechanism that allows the agent working in a distributed environment to maintain the same level of abstraction as in local contexts. In this way, agents do not have to deal with details about network connections, which hinders their abstraction level, and the way they work in comparison with locally focused environments, reducing flexibility. This work proposes a model that creates a distinctive layer for environment distribution, which the agents do not manage directly but can exploit as part of infrastructure services. The proposal is in the context of JaCaMo, the Multi-Agent Programming framework that combines the Jason, CArtAgO, and MOISE technologies, specially focusing on CArtAgO, which provides the means to program the environment. The proposal makes an extensive use of the concept of workspace to organize the environment and transparently manage different distributed sites.

Keywords: Distributed environments, Endogenous environments, Environment Programming, JaCaMo framework.

1 Introduction

Traditionally, agents are conceived as entities situated in an environment, which they can perceive and modify through actions, also reacting to changes in it accordingly [16]. Not only that, but the agents' goal is to achieve an environment desired state. This conception of environment, as the locus of agent perception, action, reaction, interaction, and goals, stays true in current MAS development.

Two general perspectives are adopted when defined the concept of environment in MAS: exogenous, and endogenous [14]. The exogenous perspective is rooted in Artificial Intelligence, conceiving the environment as the external world, separated to the actual MAS, which can be only perceived and acted upon by agents. An example of this conception can be found in EIS [1]. In contrast,

the endogenous perspective, grown in the context of Agent-Oriented Software Engineering (AOSE) [10], conceives the environment as a first class abstraction for MAS engineering [17], that not only can be perceived and acted upon, but it can also provide services and tools for the agents to aid them in their tasks, and as such, it is designed and implemented as part of the whole system. An example of this conception is found in CArtAgO [13].

From a Software Engineering point of view, environments can also be of two types: local, and distributed. In the local case, the entirety of the environment is centralized in a single process, being the easiest case for implementation. Distributed environments, on the other hand, entail multiple processes, possibly across a network, to contain the environment, and presents various challenges in the implementation and conceptualization side. In this work, we expose the idea that, from the agents point of view, there should not be any difference between local and distributed environments, the right level of abstraction should be encouraged instead, recognizing this by the term Distributed Transparency.

Distribution is not a new topic in MAS, multiple MAS technologies such as JADE [2], have a mature support for it, but it is mostly centered in agent communication and interaction, not on the endogenous conception of environment which this works entails. In this regard, JaCaMo [4] is a better example, as it supports endogenous distributed environments. This support is practical, but it lacks distributed transparency as there is a clear conceptual distinction between local and distributed environments, so agents, and agent plan programmers, need to handle the difference, reducing in this way the flexibility of the system, and forcing the programmer to change the level of abstraction in each case.

Scalability and fault tolerance are also issues when dealing with distribution, a flexible configuration is required in order to deploy the system in different settings, allowing it to grow or change as network problems arise. A good example of a distributed deployment system for JADE is [6]. Returning to the case of JaCaMo, there is no support for fault tolerance, and it lacks proper configuration facilities for distributed deployment.

This work proposes an extension to the Agents & Artifacts model of JaCaMo for modeling distributed transparent environments, while giving insights of how to address distributed deployment and fault tolerance. The outcome is an implemented JaCaMo-oriented infrastructure and Agent API that gives support to the mentioned requirements, while extending the dynamics and possibilities of MAS programming in general.

The paper is organized as follows. Section 2 briefly introduces the JaCaMo framework, addressing its distributed model. Section 3 introduces the problems present in the JaCaMo distributed model, presenting a proposal to solve them, first in an intuitive and informal manner, and then formally. Section 4 features different aspects of the implementation, such as the general architecture, and configuration and deployment. Section 5 discusses a case study that shows how the new JaCaMo-oriented implementation approach compares to current JaCaMo, giving code examples. Being a work in progress, section 6 discusses vari-

ous topics regarding future work, including proper evaluation and fault tolerance implementation. Finally, section 7 closes this paper with a conclusion.

2 Background

Although the discussion here is about endogenous environments in general, we adopt the JaCaMo [4] framework to implement our proposed model and guide our discussion, this is due the fact that, to the best of our knowledge, it has the most mature implementation of endogenous environments for MAS. As such, a brief introduction of JaCaMo is presented in this section. JaCaMo is the result of the orthogonal composition of three technologies for MAS: Jason [5] (taken as a proper name inspired by Greek mythology), CArtAgO [13] (Common ARTifact infrastructure for AGents Open environments), and MOISE [9] (Model of Organisation for multI-agent SystEms).

Jason provides the means for programming autonomous agents. It is an agent oriented programming language that entails the Belief-Desire-Intention (BDI) approach, it is based on the abstract language $AgentSpeak(L)$ [12]. Apart from its solid BDI theoretical foundations, the language offers several facilities for programming Java powered, communicative MAS. Communication in Jason is based on Speech Acts, as defined in KQML [7].

CArtAgO provides the means to program the environment, following an endogenous approach where the environment is part of the programmable system. In CArtAgO terms, the aspects that characterize a model for environment programming are the following [14]: 1) Action model: how to perform actions in the environment. 2) Perception model: how to retrieve information from the environment. 3) Environment computational model: how to represent the environment in computational terms. 4) Environment data model: how to share data between the agent and environment level to allow interoperability. 5) Environment distributed model: how to allow computational distributed environments. Aspects 1-3 are directly supported by artifacts [11], which are dynamical sets of computational entities that compose the environment and encapsulate services and tools for the agents. Artifacts are organized and situated in workspaces, which essentially are logical places (local or remote) where agents center their attention and work. Aspect 5 is supported by workspaces, but also partially by artifacts, as artifact actions can be executed remotely. Aspect 4, on the other hand, depends on the underlying agent programming language used and is not directly related to artifacts or workspaces.

MOISE provides the means to create agent organizations, which have the aim to control and direct agent autonomy in a general purpose system. To this end, it is possible to specify tree aspects: i) Structural, consisting on the different agent groups and roles that take part in the organization; ii) Functional, defined by social schemes, missions, and goals which direct the agent behaviour toward organization ends; and finally iii) Normative, defined though norms that bind roles to missions, constraining agent's behaviour when entering a group and playing a certain role.

2.1 JaCaMo and CArtAgO Distribution Model

As mentioned earlier, environment programming in JaCaMo is provided by means of CArtAgO, considering distribution in its model. At the higher level, distribution is achieved through workspaces, which serve as logical places where agents may center their attention, and where artifacts are situated. Agents can create, join, and quit workspaces. If an agent is in a workspace, it can use the artifacts situated there.

At the low level, nodes enable distribution. A node is a CArtAgO process that can be remote, where workspaces can be spawned. When a JaCaMo MAS is deployed, it is contained in a default node, that node is also the default for agents, which consider it as it's local context, so workspaces created in that node are also local workspaces, but workspaces created in different nodes are considered remote workspaces. The distinction between remote and local workspace is not only conceptual, but also syntactical, requiring IP and port information at the agent level to manipulate remote workspaces. Figure 1 depicts the current CArtAgO environment model from the workspaces and nodes perspective. From the figure, it is apparent the fact that there is no connection between nodes, and in consequence between workspaces in different nodes, needing to explicitly know the IP address and port of each node.

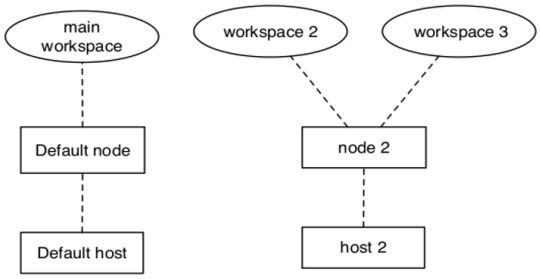

Fig. 1. Current CArtAgO environment model depicting multiple nodes and workspaces deployed.

More concretely, the following code snippet shows the difference in the JaCaMo API for the local and remote versions of join workspace, taking as a basis figure 1 where *default node* represents the local node, and *node*2 a remote one:

```
1    joinWorkspace("main", WspId1);
2    joinRemoteWorkspace("workspace2", "192.168.0.2:8091", WspId2);
```

3 Proposal

Environment programming in JaCaMo comes with various shortcomings regarding distributed settings, being the most important the fact that local and remote workspaces are defined and treated differently, which derives in the following problems: i) There is not distributed transparency for agents, being forced to directly manipulate network information, making network distinctions between workspaces. ii) The underlying environment topology is difficult to represent and exploit by the agents as it does not follow any structure or workspace relations beyond the sharing of the same node. All of these problems have the consequence of reducing the abstraction level in which agents work, impacting flexibility and environment exploitation as well.

Another problem is the lack of proper configuration facilities to allow the inclusion of remote workspaces information at deployment time, meaning that host information for remote workspace spawning need to be hard-coded on the agent programs or externally supported. To spawn a remote workspace, a CArtAgO node needs to be running on the destination host, and there is not any integrated facility to manage them automatically when needed. Furthermore, the current distributed implementation does not exhibit any degree of fault tolerance, this is specially important for possible network connection problems that may arise in a distributed system.

In this section, a proposal to solve the identified problems is presented. A separation between environment and infrastructure is suggested. The environment is represented as a hierarchical tree structure, which represents the topology. In this tree, each node is a workspace which actual physical placement on the distributed system is irrelevant. Workspaces may be deployed in different places, but for the agents point of view, it only matters their placement in the topology. A workspace may be the logical parent of another one, multiple workspaces can be in the same physical place, and there is no restriction about how the topology may be organized, e.g.; workspaces on the same physical place may be on different branches. This allows to organize environments as it is usually done in CArtAgO, but in a more structured way, also supporting remote workspaces transparently.

In a practical sense, each workspace in the tree is represented by a path starting at the root workspace, these paths brings the notion of logical placement that agents require to organize and exploit their environment. We adopt a Unix-like path format to represent this placement, but using a "." instead of a "/", following Jason syntax. These paths are used by the agents to execute environment related actions, such as creating new workspaces or joining one. From the proposed JaCaMo API, there is no difference between local and remote actions related to workspaces. For example, returning to the code snipped presented in section 2.1 for joining local and remote workspaces, which it is related to figure 1; with the proposal, a workspace topology would be created, a possibility is to have *workspace*2 and *workspace*3 as direct descendants of the root workspace *main*, with this setting the associated code snipped is as follows:

```
joinWorkspace("main", WspId1);
joinWorkspace("main.workspace2", WspId2);
```

As in current CArtAgO, agents may work in multiple workspaces at the same time, but the concept of current workspace is dropped since in actuality all the joined workspaces should be considered the current context of working. Nevertheless, agent may specify the target workspace for an action. A new introduced concept is the home workspace of an agent, which it is the workspace where the agent is initially deployed, serving as a relative reference to other places in the topology, providing a default place for the agent, and also serving as the default workspace to execute actions when a target workspace is not specified.

On regard of the infrastructure, a layer is added to manage distribution, this layer provides the required services for the agents to exploit their distributed environment. These services include: i) Workspace management, so agents can create, join, and quit workspaces no matter their physical placement; ii) Topology inspection, so agents can reason about the current topology organization and do searches concerning workspaces; iii) Workspace dynamics observation, so agents can know when other agents manage workspaces, or when workspaces disconnect and reconnect after a network problem; iv) Disconnection and fault tolerance to manage and recuperate from network problems, which it is currently left as future work, but initially designed as presented in section 6.2 . We believe that the set of mentioned services do not only bring distributed support, but also enhance the dynamics of MAS in general, extending its possibilities.

3.1 Formal description

JaCaMo assumes an endogeneous approach to MAS, i.e., the environment is an explicit part of the system:

Definition 1. *A MAS is composed by a set of agents (Ags), their environment (Env), and an infrastructure (Infr) running both of them:*

$$MAS = \{Ags, Infr, Env\}$$

The set of agents is composed by $n \geq 1$ agents:

$$Ags = \{a_1, \ldots, a_n\}$$

Each agent, as usual, is composed by beliefs, actions, and other elements equal to:

$$a_i = \{bels, acts, \ldots\}$$

By default, when created, an agent includes minimally:

$$a_i = \{joined(home)\}, \{join, quit, create\}, \ldots\}$$

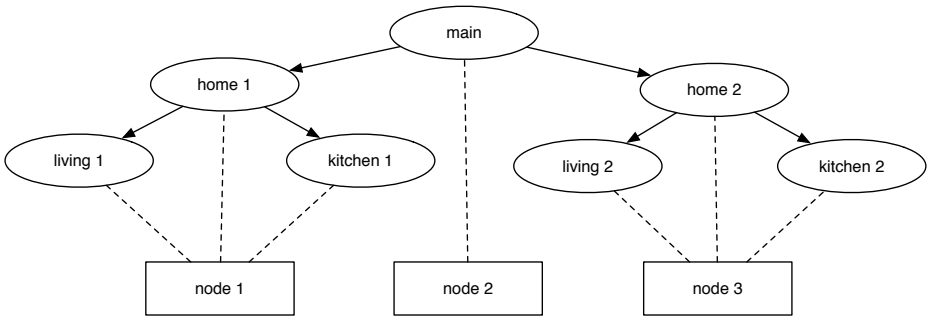

Fig. 2. The intented view of an endogeneous environment.

which means that every agent believes he has joined a home workspace, and has actions to join, quit, and create workspaces; and update the information about the environment.

Figure 2 illustrates the intended view of the environnment in this proposal. First, the environment, properly speaking, is a tree of workspaces, expressing a kind of spatial relation among workspaces, e.g., the kitchen 1 is at the home 1. Second, nodes and hosts are not part of the environment, but are defined as part of the infrastructure of the MAS, nevertheless, workspaces keep information about its corresponding physical node.

The infrastructure is a layer hidden to the agents, that gives the low level support to distribution, formally defined as:

$$Infr = \{Nodes, Hosts\}$$

where:

- $Nodes = \{node_1, \ldots, node_k\}$ is a set of CArtAgO nodes, i.e.; processes, possibly remote, where workspaces can be created. Each $node_i$ is a tuple $\langle ni, SWsps, hi, port \rangle$, where ni is an unique identifier for the node; $SWsps \subseteq W$ is the set of spawned workspaces in the node, containing at least a default workspace for the node; hi is an identifier of the host computer where the node exists; and $port$ is the host port used by the node process.
- $Hosts = \{host_1, \ldots, host_p\}$ is the set of available computer devices on the distributed system. Each $host_i$ is a tuple $\langle hi, HNodes, ip \rangle$, where hi is a host unique identifier, $HNodes \subseteq Nodes$ is a set of hosted nodes, and ip is the IP address of the computer.

Formally, the environment Env is defined as a graph:

$$Env = \{W, E\}$$

where:

- $W = \{w_1, \ldots, w_i\}$ is a finite, non-empty set of $i \geq 1$ workspaces that contain artifacts. Each $w_i = \langle idW, name, ni \rangle$, where idW is an unique identifier for the workspace, $name$ is a logical name, and ni is a reference to the CArtAgO node in $Infr$ that contains w_i. The *node* element establishes a connection between the environment and the infrastructure, in order to forward agent actions to the destined physical place.
- $E \subset W^2$ is a set of edges over the workspaces, such that Env is minimally connected, i.e., it is a rooted tree that represents the environment topology.

For instance, following Figure 2, $Env = \{W, E\}$, and considering for simplicity only the name of each w_i, such that:

- $W = \{main, home1, home2, living1, kitchen1, living2, kitchen2\}$
- $E = \{\{main, home1\}, \{main, home2\}, \{home1, living1\}, \ldots\}$

The expression $w_1.w_2 \ldots .w_n$ denotes a path on Env, if:

- $w_i \in W$ for $i = 1, \ldots, n$;
- $\{w_{i-1}, w_i\} \in E$ for $i = 2, \ldots, n$.

Abusing a little bit of the notation, we can write $w_1 \ldots . w_n \in Env$. For instance, $main.home1.living1 \in Env$. Some useful operations over paths, include:

- $last(w_1.w_2 \ldots . w_n) = w_n$
- $butlast(w_1.w_2 \ldots . w_{n-1}.w_n) = w_1.w_2 \ldots . w_{n-1}$
- $add(w, w_1.w_2 \ldots . w_n, Env) = w_1.w_2 \ldots . w_n.w$. This involves adding w to W, and $\{w_n, w\}$ to E in Env.
- $del(w, w_1.w_2 \ldots . w_n.w, Env) = w_1.w_1 \ldots . w_n$. This involves deleting w from W, and $\{w_n, w\}$ from E in Env.

In what follows, the transition rules related to environment agent actions are described, workspaces denote paths in the environment.

Joining a workspace An agent can ask himself about the workspaces he has currently joined: $ag_{bels} \models joined(w)$, if and only if, w is a workspace currently joined by the agent. Recall that by default $ag_{bels} \models joined(home)$. An agent can join different worspaces concurrently, so that $ag_{bels} \models joined(Ws)$ unifies Ws with a list of the workspaces joined by the agent. Two transtion rules define the behavior of the action *join*. First, an agent can join a worspace w, if and only if w is a path in the environment Env and it is not believed to be already joined:

$$(\mathbf{join_1}) \qquad \frac{join(w) \mid w \in Env \wedge ag_{bels} \not\models joined(w)}{\langle ag, Env \rangle \rightarrow \langle ag', Env \rangle}$$
$$s.t.\ ag'_{bels} = ag_{bels} \cup \{joined(w)\}$$

Second, nothing happens if an agent tries to join a previously joined worspace:

$$(\mathbf{join_2}) \quad \frac{join(w) \mid ag_{bels} \models joined(w)}{\langle ag, Env \rangle \rightarrow \langle ag, Env \rangle}$$

Any other use of *join* fails.

Quiting workspaces An agent can quit the workspace w if he believes he had joined w. The agent forgets such belief.

$$(\mathbf{quit_1}) \quad \frac{quit(w) \mid ag_{bels} \models joined(w)}{\langle ag, Env\rangle \rightarrow \langle ag', Env\rangle} \quad s.t.\ ag'_{bels} = ag_{bels} \setminus \{joined(w)\}$$

If the agent tries to quit a workspace he has not joined yet, nothing happens:

$$(\mathbf{quit_2}) \quad \frac{quit(w) \mid ag_{bels} \not\models joined(w)}{\langle ag, Env\rangle \rightarrow \langle ag, Env\rangle}$$

Creating workspaces An agent can create a workspace w, if it is not a path in the environment, but $butlast(w)$ is one:

$$(\mathbf{create_1}) \quad \frac{create(w) \mid w \notin Env \wedge butlast(w) \in Env}{\langle ag, Env\rangle \rightarrow \langle ag, Env'\rangle} \quad s.t.\ Env' = add(last(w), butlast(w), Env)$$

Observe that the result of creating a workspace must be propagated to the rest of the agents in the MAS. This could be done by the infrastructure, or broadcasting the *add* operation. The actual node where the workspace is going to be created is decided by the infrastructure following a policy, by default the infrastructure spawns the workspace on the same node where its parent workspace is running.

Trying to create an existing workspace does nothing:

$$(\mathbf{create_2}) \quad \frac{create(w) \mid w \in Env}{\langle ag, Env\rangle \rightarrow \langle ag, Env\rangle}$$

4 Implementation

The model introduced on section 3 is open enough to allow different implementations. This section presents a practical possibility, intended to be integrated with JaCaMo. The core implementation and main design choices are related to the general architecture, and configuration and deployment.

4.1 General architecture

The architecture to support agent services is based on the concept of Node, which refers to the $Nodes$ element in $Infr$. Nodes represent independent CArtAgO processes, possibly remote, running on a given host ($Hosts$ element in $Infr$),

and associated to a port. Nodes are the main abstraction to manage workspaces (W element in Env), and as such, they provide all the necessary tools to create, join, and quit workspaces, as well as the means to communicate with other nodes in order to maintain a consistent workspace topology, and properly dispatch topology related events. The workspace topology corresponds to the E element in Env. A NodeArtifact is the gateway used by an agent to interact with the node services and to observe the distributed environment. There is a NodeArtifact in each workspace, and every agent has access to one of them, which one depends on its *home* workspace, which in turn it is intended to be on the same node as the agent process.

Nodes communicate between each other following a centralized approach: one node is designated as the central node, this is usually the node deployed by default by JaCaMo, so every change on the topology is inspected and approved by a single node, and the associated actions and events are dispatched from there. This centralized approach makes possible to maintain a consistent topology, avoiding run conditions. To exemplify node communication, the workflow for creating a new workspace is the following:

- An agent that wants to create a workspace issues the action to its corresponding NodeArtifact, passing a tree path.
- The artifact checks if the tree path is consistent with the topology tree, if it is, it sends a request to the central node.
- The central node issues a request to the end node where the workspace is actually going to exist. By default, it chooses as end node the same as the parent workspace from the path given.
- The end node creates the workspace and returns control to the central node.
- The central node makes the corresponding changes to the workspace topology and communicates the success to the original requesting node. It also dispatches a create and tree change event to the other nodes.

As the node model is centralized, there exists the concern of a single point of failure, that is why all nodes maintain redundant information about the topology, so it is possible to recuperate from a central node dropping, as any node can take the role of central node. The topology structure is also lightweight, which speeds up the tree synchronization among nodes, this synchronization is only required when there is a change in the topology. This redundancy also allows to boost the efficiency of operations such as joining and quitting workspaces, since those operations only need to read from the topology, so the local copy is used in those cases. Communication with the central node is only required in cases where a change in the topology is required. We believe that in traditional environment management, it is more common for the agents to join and quit workspaces than to create new ones.

4.2 MAS configuration and deployment

To ease the deployment of the distributed infrastructure is a goal of our overall proposal, this means to be able to configure and launch the desired hosts, nodes,

and workspaces that will take part in the MAS from the start. It is also possible to manually add new nodes after deployment. The idea is to extend the deployment of JaCaMo, where only workspaces are considered. JaCaMo uses a text file known as the JCM file to configure the deployment of the MAS. The intention is to further include fields in this file to also configure host, and nodes for distributed systems; and add the facilities to automatically launch CArtAgO nodes in remote machines through a daemon service.

The changes to the JCM file include:

- Host configuration: assign a logical name and IP address to each host.
- Node configuration: assign a logical name for the node, i.e.; the name of the default workspace; the related host name; and optionally a port number.
- Workspaces configuration: relate each workspace to a specific node.
- Lib file: the path to a jar file containing all the necessary files to launch CArtAgO nodes. This includes custom artifacts binaries, third party libraries, custom classes binaries, and any configuration file. This file is intended to be shared among all nodes.

5 Case study

In order to show how to exploit the proposed distributed model and implementation, and how it compares to the current version of JaCaMo, a small case study is presented in this section. This case study pretends to show the new proposed agent API, and the flexibility of the proposed model, focusing mainly on workload distribution, which is one of the aspects that can be enhanced, but other aspects such as fault tolerance, reactiveness on environment changes, and more complex agent organizations and dynamics are also possible.

The case study consists on constantly fetching current weather information for every city in a country, and with that information, construct weather prediction models for each city, so the models could be consulted by end users through a user interface. The construction of each model is an online learning [3] task that can be heavy on the computational side, so work distribution is desirable. To simplify the distributed setting, assume that the cities of the country are divided in west-cities and east-cities, one computer is in charge of the models from the west-cities, and another one from the models of the east-cities; furthermore, information fetching need to be quick and constant, and also the end user searching service, so one computer takes care of both. The described setting yields a total of three different computers in the distributed system.

Workflow is modeled through 3 agent programs: fetcher, which fetches weather information and forwards it to a destination depending the type of city; learner, which task consist on creating online weather prediction models for each city with the data provided by the fetcher agent; and finally, a searcher agent, which attends end user requests to retrieve information from the learned models.

When implementing this case study in current JaCaMo some problems will arise: the IP addresses of the different computers should be hard-coded in the agent programs; every CArtAgO node representing a remote workspace should

be manually started in every computer on the distributed setting; startup plans should be implemented in order to focus the attention of each agent in its designated place; when the searcher agent has to collect information about learned models to attend a petition, it necessarily has to ask learner agents about its location, not only considering the workspace but also the ip address where the workspace is, making also necessary to distinguish between local and remote workspaces when the agent intend to return the result to the end user. A better solution using our new approach that solves all the mentioned problems, and better exploits the environment is presented next.

– A possible JACAMO configuration file for this example, following the idea from section 4.2, is the following. For the sake of clarity, artifact related configuration, and MOISE related organization is not shown.

```
mas weather {
   host c1 {
     ip: 192.168.0.2
   }
   host c2 {
     ip: 192.168.0.3
   }
   node west {
     host: c1
     port: 8080
   }
   node east {
     host: c2
     port: 8080
   }
   agent fetcher : fetcher.asl {
     home: main
   }
   agent west_learner : learner.asl {
     home: main.west
   }
   agent east_learner : learner.asl {
     home: main.east
   }
   agent searcher : searcher.asl {
     join: main.central
   }
   lib_file : /home/system/wheather.jar
}
```

Note that workspaces *main*, *main.west*, and *main.east* are implicitly created as they are the default workspaces for the nodes, e.g.; *main* is the node deployed by default by JaCaMo.

In our proposed model, agents refer to workspaces only through their path in the topology, giving in this way a more structured sense of placement. As in UNIX file system paths, tree paths can be considered absolute or relative. A path is relative from the *home* workspace of the agent, and should start with a single dot, any other path is considered absolute and must begin at the root workspace (by default *main*).

A simplified version of the agent programs is presented next.

– fetcher:

```
//creations and initializations omitted
+!fetch : true <-
  getData(Data);
  category(Data, Cat);
  if(Cat == "west") {
    .send(west_learner, tell, add(Data));
  }
  else {
    .send(east_learner, tell, add(Data));
  };
  !fetch.
```

– learner:

```
//initialization of agent omitted
+add(Data): true <-
   getCity(Data, City);
   .concat(".", City, Path); //from home
   createWorkspace(Path); //does nothing if already present
   joinWorkspace(Path);// does nothing if already joined
   //creations and initializations omitted
   addData(Data)[wsp(Path)]; //route action to wsp
   induceModel[wsp(Path)].
```

– searcher:

```
+search(Query) : true <-
   //Query is just the name of the city
   .concat(".*", Query, RegExp);
   searchPaths("main", RegExp, [H | T]);
   .length(List, Len);
   if(Len > 0) {
     joinWorkspace(H);
     getForecast(Forecast)[wsp(H)];
     quitWorkspace(H);
     sendReply(Forecast);
   }.
```

Some of the actions from the agent programs correspond to the API of the proposed model, such actions are described as follows.

– $joinWorkspace(+WspPath, -WspId)$: the agent adds a specified workspace to its joined workspaces list, obtaining a workspace ID.
– $quitWorkspace(+WspPath)$: removes a specified workspace from its joined workspaces.
– $createWorkspace(+WspPath)$: creates a new workspace on the specified path. By default, the workspace will actually be spawned on the same CArtAgO node as the parent workspace derived from $WspPath$, this allows workload management for workspaces.
– $searchPaths(+PointPath, +RegExp, -WspsPathList)$: returns a list with the workspaces that follow a certain regular expression, the search is restricted to the subtree given by $PointPath$. This action exemplifies how the topology organization may be exploited.

It is worth mentioning that the actual API is more extensive, including perception, events, and more actions related to the environment. For example, it is possible for the agents to react to the creation of a new workspace through the event $created(WspPath)$, or analyze and exploit the current topology organization through the perception $topology\_tree(List)$ where $List$ is of the form: $[main, [subNode_1, [subsubNode_1, [...], subsubNode_m]], ..., [subNode_n]]$.

As the example shows, agents do not concern about computer distribution, they simply work with workspaces: learner agents organize their work in workspaces that can be on different computers; and the searcher agent can reach any workspace directly, not relying on agent communication, but directly acting though the knowledge of the topology. Deployment is also greatly enhanced since the distributed setting is properly configured beforehand, and the launching of nodes is automatic.

6 Discussion and future work

As a work in progress, our proposal still lacks a proper treatment of different aspects such as evaluation, and fault tolerance. This sections briefly discusses and outlines these topics, which are considered as immediate future work.

6.1 Evaluation

Our proposed model, while introduced in some formal way, still needs a proper formal evaluation, for this end, the adoption of a more formal representation such as the ones proposed in [8], and [15] seem to be required.

Concerning performance evaluation, with the adopted centralized model, it is required to asses scalability issues that may arise as nodes are added to the MAS. In case of finding such problems, we can still improve some of the required subprocess, as the synchronization of the topology among all nodes, and event propagation, which could be distributed.

6.2 Fault tolerance

Given the proposed architecture, connection loss is the same as node dropping, and as such it directly impacts the topology tree structure used by agents as all the corresponding workspaces are also lost. An intuitive idea of how fault tolerance could be implemented following our design choices is described next.

Following the overall node organization introduced in section 4.1, all nodes maintain a keepalive connection with the central node, and a ordered list of connected nodes. If a node losses connection, then the central node issues the corresponding dropping event to the rest of the nodes, and modifies the topology tree structure accordingly. The disconnected node tries to establish a connection with the rest of the nodes, following the order of the connected nodes list, this being useful on the case that several nodes lost connection or the central node dropped. With the available nodes (it may be only one), the node in the upper

position on the connected nodes list is designated as the new central node, issuing the corresponding disconnection events and creating a new tree node structure where every default workspace from the nodes is on the same level. The new central node keeps trying to reconnect to the original central node for a period of time.

When successfully reconnecting, the original central node will try to return the topology to the way it was before the problem, but sometimes that would not be possible, e.g.; when one of the nodes keep missed. It is strongly recommended that every default workspace corresponding to a node is mounted on the same upper tree level of the tree, that way when reconnecting, the tree structure will keep consistent with the way it was before, otherwise the tree topology may vary in an unexpected manner, which can be problematic on certain applications. After the node tree structure is recreated, the reconnecting nodes return to work with the original central node, and the central node triggers the corresponding reconnection events.

7 Conclusion

The introduced model is a step forward to improve environment programming for MAS, it addresses issues related to distribution, which are important for a wide variety of applications. We see distributed transparency as the most important contribution of this work, as Multi-Agent Systems are intended to raise the level of abstraction in software development, as compared with other industry established programming paradigms such as POO. Proper abstraction levels for aspects such as concurrency management are already accomplished, but distributed computing is still an important topic to improve.

With a solid foundation for distributed environment programming, it is possible to address new challenges like MAS interoperability, which refers to the integration and collaboration of independent Multi-Agent Systems. An extension to the proposed model and implementation is envisaged to support MAS interoperability features, such as MAS composition where two different MAS can fuse together to extend the scope of their work; and also MAS attachment, where a mobile MAS can temporally join a MAS in order to exploit services. These features bring new possibilities to the dynamics of MAS in general, and are also interesting from the software engineering point of view as they allows an upper level of flexibility and scalability.

References

1. Tristan M Behrens, Koen V Hindriks, and Jürgen Dix. Towards an environment interface standard for agent platforms. *Annals of Mathematics and Artificial Intelligence*, 61(4):261–295, 2011.
2. Fabio Bellifemine, Agostino Poggi, and Giovanni Rimassa. Jade–a fipa-compliant agent framework. In *Proceedings of PAAM*, volume 99, page 33. London, 1999.

3. Albert Bifet, Geoff Holmes, Richard Kirkby, and Bernhard Pfahringer. Moa: Massive online analysis. *Journal of Machine Learning Research*, 11(May):1601–1604, 2010.
4. 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.
5. Rafael H. Bordini, Jomi F. Hübner, and Mike Wooldridge. *Programming Multi-Agent Systems in Agent-Speak using Jason*. John Wiley & Sons Ltd, 2007.
6. Lars Braubach, Alexander Pokahr, Dirk Bade, Karl-Heinz Krempels, and Winfried Lamersdorf. Deployment of distributed multi-agent systems. In *International Workshop on Engineering Societies in the Agents World*, pages 261–276. Springer, 2004.
7. T. Finin et al. An overview of KQML: A knowledge query and manipulation language. Technical report, University of Maryland, CS Department, 1992.
8. Matthew Hennessy. *A distributed Pi-calculus*. Cambridge University Press, 2007.
9. Jomi F Hübner, Olivier Boissier, Rosine Kitio, and Alessandro Ricci. Instrumenting multi-agent organisations with organisational artifacts and agents. *Autonomous Agents and Multi-Agent Systems*, 20(3):369–400, 2010.
10. James J Odell, H Van Dyke Parunak, Mitch Fleischer, and Sven Brueckner. Modeling agents and their environment. In *International Workshop on Agent-Oriented Software Engineering*, pages 16–31. Springer, 2002.
11. Andrea Omicini, Alessandro Ricci, and Mirko Viroli. Artifacts in the a&a meta-model for multi-agent systems. *Autonomous agents and multi-agent systems*, 17(3):432–456, 2008.
12. A.S. Rao. AgentSpeak(L): BDI agents speak out in a logical computable language. In Rudy van Hoe, editor, *Seventh European Workshop on Modelling Autonomous Agents in a Multi-Agent World*, Eindhoven, The Netherlands, 1996.
13. A. Ricci, M. Viroli, and A. Omicini. Construenda est cartago: Toward an infrastructure for artifacts in MAS. *Cybernetics and systems*, 2:569–574, 2006.
14. 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, 2011.
15. Alessandro Ricci, Mirko Viroli, and Maurizio Cimadamore. Prototyping concurrent systems with agents and artifacts: Framework and core calculus. *Electronic Notes in Theoretical Computer Science*, 194(4):111–132, 2008.
16. Stuart Jonathan Russell, Peter Norvig, John F Canny, Jitendra M Malik, and Douglas D Edwards. *Artificial intelligence: a modern approach*, volume 2. Prentice hall Upper Saddle River, 2003.
17. 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.