A Flexible and Lightweight Agent Deployment Architecture

Andrei Olaru
Department of Computers
University Politehnica of Bucharest
Bucharest, Romania
andrei.olaru@cs.pub.ro

Alexandru Sorici
Department of Computers
University Politehnica of Bucharest
Bucharest, Romania
alexandru.sorici@cs.pub.ro

Adina Magda Florea
Department of Computers
University Politehnica of Bucharest
Bucharest, Romania
adina.florea@cs.pub.ro

***Abstract*—While the multi-agent system (MAS) paradigm is popular for many application domains, different frameworks exist for different applications. These frameworks are not interoperable and many times force the developer into a specific model for agents and for the system as a whole.**

This paper presents a framework featuring a lightweight, open model for the entities in a MAS, together with a means of deployment that brings both flexibility and versatility. This architecture is compared to the most popular frameworks in the MAS domain.

***Index Terms*—software agents, multi-agent systems, deployment framework, Internet of Things**

I. Introduction

Multi-Agent Systems (MAS) are seen by many researchers as an adequate paradigm for the implementation of applications and, more often, middleware, in several domains, such as Ambient Intelligence (AmI) [1], the Internet of Things (IoT) [2], Active Assisted Living (AAL) [3], vehicular communication systems, edge computing [4], robots and swarms, and many others. In all of these fields, entities – devices, platforms, users, computing behaviors – are heterogeneous both as origin and as platform and capabilities; have a high degree of autonomy; and interact with other entities. Modeling such scenarios as multi-agent systems helps researchers and developers in simulations, as well as in the actual deployment, by leveraging MAS characteristics such as loose coupling, heterogeneity and orientation towards the perspective of individual entities.

Current research regarding the implementation of Multi-agent Systems (MAS) addresses two important, but different, aspects: the aspect of *programming the agents*, namely configuring the mental states of the agent and the parameters of its reasoning cycle, usually using specialized agent-oriented programming *languages*; and the aspect of agent organization and *programming the environment*, focusing on how the various actors in the deployment should be organized and how they should interact. Some implementations address both levels, offering both the enclosing framework, as well the means to model the agent internally, either by using an AOP language which is integrated with the framework (as in the example of JaCaMo with Jason [5]), or by offering methods to model the agent's behavior (as in the example of Jade [6]).

The general goal of this research is to help developers create systems in which each entity has its own code and model, but is able to easily integrate itself in a system and communicate with other entities. We wish to decouple the implementation and implementation language for the agent's behavior (such as by using the BDI model or the Jason language, etc) from the functionality offered by the agent platform / development framework, such as communication, addressing, mobility, monitoring and control.

There are currently several frameworks that allow developers to focus on programming the behaviour of the agent, letting the framework take care of agent management, communication, mobility, and discovery. Examples of existing frameworks are Jade[1], Jiac[2] and Jadex[3]. Some of these also offer the means to implement agents using the BDI model. Jason and the JaCaMo framework[4] also combine AOP language and execution platform. However, we found that each of these platforms lack the means to easily deploy certain types of scenarios. In JaCaMo it is difficult to discover services and artifacts through the tools provided to the AOP language, which is needed in IoT scenarios; in Jade there is no support for entities above the agents, implementing groups and organizations, which is needed in complex scenarios; in Jiac it is difficult to move away from the Java EE-inspired model, which may be needed on more restrictive platforms.

Another class of multi-agent system related tools are modeling and simulation frameworks. They allow the simulation of large numbers of agents on the local machine or on high-performance computing clusters. However, they don't offer support for distributed deployment and communication

This research was supported by grant PN-III-P1-1.2-PCCDI-2017-0734 and grant NETIO — 1268/22.01.2018.

[1]The final publication can be found on IEEE Xplore
A. Olaru, A. Sorici and A. M. Florea, *A Flexible and Lightweight Agent Deployment Architecture*, 2019 22nd International Conference on Control Systems and Computer Science (CSCS), Bucharest, Romania, 2019, pp. 251-258, doi: 10.1109/CSCS.2019.00048.
https://ieeexplore.ieee.org/document/8744845

[1]Java Agent DEvelopment Framework (http://jade.tilab.com/, accessed March 2019)

[2]Java-based Intelligent Agent Componentware (http://www.jiac.de/, accessed March 2019)

[3]Jadex Active Components (https://www.activecomponents.org, accessed March 2019)

[4]JaCaMo Multi-Agent Programming Framework (http://jacamo.sourceforge.net/, accessed March 2019)

between arbitrary devices. Our intention with this research is to provide an architecture that allows for the same agents to be deployed both on distributed, real-life setups, but also to be part of fast local simulations.

This paper addresses the needs of researchers and developers who need to easily develop, quickly deploy and painlessly monitor and control multi-agent systems for a variety of scenarios, with an effort that is only proportional to the complexity of the scenario. The goal is to offer the means to focus on programming the behavior of agents, without the need to think about how the agents communicate or move from one machine to another, but at the same time offer the possibility of choosing the appropriate method of communication for each part of the system, and switch methods at run-time if necessary, without affecting the implementation of the agents. Envisaged scenarios also include the simulation of (large numbers of) agents on the same machine. Another objective is to enable deployment, control, and monitoring using only the command line (if needed), as in many complex deployments most nodes (if not all) do not feature graphic terminals.

While the main application domain of this research is IoT and the proposed architecture tries to deal primarily with challenges in the Internet of Things [7], our intention is to create a framework that can be also used in AmI, AAL, vehicular networks, robot systems, and MAS simulation.

To this end, we propose a model that comprises two elements:

- a generic, open architecture for multi-agent systems, which enables a modular and lightweight implementation and an easy deployment, both for pre-defined and for application-specific entities.
- a means to quickly create and/or completely configure a MAS deployment from the command line and/or using an XML file, as well as control and monitor the MAS at run-time from the command line.

We have named the implementation of this proposed model FLASH-MAS, with "FLASH" being an acronym for "Fast Lightweight Agent Shell". The implementation, using the Java language, is open-source and licensed under the GPLv3[5].

We have designed FLASH-MAS with the following principles in mind:

- the model should support the implementation and management of agents, the environment, artifacts, groups, and organizations.
- the developer should only write the code for actual functionality that he/she needs, with minimal overhead related to deployment.
- the structure of entities offered by the deployment framework (nodes – communication infrastructures – agents – agent components) should only serve as a guideline and the developer should be able to replace as much of it as possible, both the implementation and the structural elements themselves. For instance, the developer should be able to implement the environment as an entity, or compose the environment of several artifacts, as reactive entities offereing services [8].
- the framework should respect the FIPA Message Structure Specification [9] and, in the future, the FIPA Abstract Architecture Specification [10].
- the framework must support a variety of communication methods, such as TCP/IP, WebSockets[6], HTTP-REST[7], publish-subscribe, etc.
- the framework should be deployable to a variety of computing platforms, both powerful and resource-constrained.
- the model should attempt to fuse the speed of agent-based simulation with the power of network-scale (and, if possible, larger scale) distributed MAS deployments.

With such ambitious goals in mind, it is clear that development of the FLASH-MAS platform and its associated model should focus on the essential: being flexible and lightweight.

The paper continues with reviewing related work in the field of MAS frameworks, followed by the presentation of the proposed architecture for FLASH-MAS in Section III and provided deployment tools in Section IV. Section V presents a comparison between FLASH-MAS and other similar, popular frameworks. The last section draws the conclusions.

II. Related Work

Frameworks for multi-agent systems are generally directed towards one of two goals: deployment as a distributed system across the network, or local simulation of large numbers of agents.

In the domain of agent development frameworks enabling distributed deployment, we see two types of frameworks. Some, such as Jade, Jiac, Jack and Jadex rely on Java agent implementations and offer various functionalities and APIs for agent management, communication and security, also helping with the agent lifecycle by means of behaviors or similar constructs. These are the frameworks most used for business and industrial applications. Other frameworks, such as Jason, Agent Factory and JaCaMo, rely on some agent-oriented programming (AOP) language, usually Jason / AgentSpeak for the implementation of agents, reasoning, and planning.

JADE [6] is by far the most popular agent development framework. It is a powerful and easy to deploy and it offers agent management, mobility, and communication for agents implemented in Java. Its main disadvantage is performance – for large numbers of agents, Jade becomes slow. The development of FLASH-MAS has been inspired by Jade, by means of previous iterations of the model – tATAmI-1 and tATAmI-2 [11], [12].

Jadex [13] extends Jade agents with reasoning capabilities (using a BDI model), agent components, and business-oriented features. However, developing agents in Jadex requires combining Java and XML in advanced specifications that may be

[5]FLASH-MAS implementation (https://github.com/andreiolaru-ro/FLASH-MAS, accessed March 2019)

[6]WebSocket.org (https://tools.ietf.org/html/rfc6455, accessed March 2019)

[7]Representational State Transfer (https://en.wikipedia.org/wiki/Representational_state_transfer, accessed March 2019)

difficult to grasp by the agent developer. In our model we offer a variety of means for specifying the elements in the agent system, which are also suitable for less experienced users.

JACK Intelligent Agents [14] is a production-grade framework for building BDI agents, building on the experience of PRS [15] and dMARS [16]. It is very well suited for building agent systems quickly and with relative ease, however it does not feature the flexibility we are trying to offer in choosing the messaging components, protocols, and concrete agent implementations.

JIAC [17] is another production-grade framework for developing complex agent systems, on both workstations (JIAC-V) and mobile/embedded devices (microJIAC). Its focus on industrial applications, by offering features for security, management and scalability. However, its architecture is based on the Java EE model, which restricts the possibilities of implementation for agents, and also forces the use of some particular base classes for agent components.

Siebog [18], is a Java EE-based framework in which agents execute in the browser and communicate exclusively through web protocols. The reliance on Java EE, however, implies a high cost for the initial deployment and while communication using web protocols supports an open, heterogeneous system, it somewhat limits the deployment scenarios.

Jason [19] is based on AgentSpeak and is an AOP language with its own special syntax to define goals, conditions, and plans. Jason agents can be run on top of Jade when a distributed setup is required. Jason is however constraining the developer into a particular way of programming agents, which some may find difficult to grasp, especially for developers not familiar with agent theory. JaCaMo [5] is a framework that combines the Jason programming language with the Moise organizational model and with the CArtAgO artifact and environment infrastructure. JaCaMo is probably one of the most suited frameworks for deploying MAS, however creating a JaCaMo deployment involves considerable work and requires a great deal of background theoretical knowledge. Performing even common agent-related tasks is not easy to do.

Agent Factory [20], together with its implementation for mobile devices – Agent Factory Micro Edition, includes flexibility for implementing agents in various programming languages, but only contains development kits only for Jason-based implementations, and development appears to have stopped some time ago.

Abar et al survey a great number of frameworks for multi-agent system simulation [21]. Such frameworks are directed towards the domain of complex systems and support large numbers of very agents. Since they handle simulation, generally all agents execute on the same machine or in a HPC environment.

The most notable MAS simulation frameworks are RePast[8] – with its flavors Repast Symphony and Repast HPC [22] – and Aimpulse Spectrum[9]. An analysis only on the scalability of various MAS frameworks, also in the context of the need for simulating large numbers of agents, is done by Lorig et al [23]. The difference between local simulation frameworks and distributed deployment frameworks is shown, especially when comparing Aimpulse Spectrum to Jade. Jade is very slow because of its communication infrastructure, while Aimpulse Spectrum is very fast, but does not support distributed deployments.

There are many agent-based implementations for IoT, both for specific applications and also as middleware for IoT. Many such implementations are based on Jade, thanks to Jade offering FIPA compliance and being very popular.

The ACOSO (Agent-Oriented Cooperative Smart Objects) Methodology specifies how to model IoT systems in an agent-oriented manner [24]. The implementation, JACOSO is Jade-based, showing that, although Jade is not the most adequate framework for IoT, there are no alternatives. Moreover, since Jade uses TCP/IP communication, the communication between agents and between agents and sensors must be done in different manners.

Calvaresi et al present another agent-based framework for IoT. They show how the agent-oriented paradigm is adequate for implementing heterogeneous systems as in the case of IoT. The proposed implementation relies, again, on Jade, for reasons of popularity and FIPA compliance.

The Sol agent platform [25] proposes a solution that is quite close to our research, in that it allows the agent to communicate using various communication protocols, depending on the sensors that devices need to communicate with or the network used by the devices. It uses *plugins* for different communication methods and modes and supports discovery, management and directory services, as well as group messaging. However, it appears that the configuration of the communication infrastructure cannot be done dynamically, and the inter-operation between different types of communication must be done by specialized proxy agents.

Self-OSGi attempts to combine a BDI architecture with the OSGi component standard, in order to ensure fault tolerance and automatic reconfiguration [26]. This is related to our initiative to include components (that we call shards) in agents. However, our goal of creating a lightweight, portable framework is not compatible with the heaviness of OSGi. We also don't need at this point the advanced features of automatic reconfiguration offered by OSGi.

III. Proposed Architecture

The proposed architecture is inspired by the concepts in the Jade framework, but is focused on allowing the greatest possible flexibility in implementing communication between agents, agent components, and the agents themselves.

This means that the same system must be able to contain agents communicating in a centralized manner by connecting to a server through TCP/IP or WebSockets, or through protocols allowing non-centralized communication [27], and even agents communicating only locally with other agents. This

[8]The Repast Suite (https://repast.github.io/, accessed March 2019)

[9]Aimpulse Spectrum Developer page (https://developer.aimpulse.com, accessed March 2019)

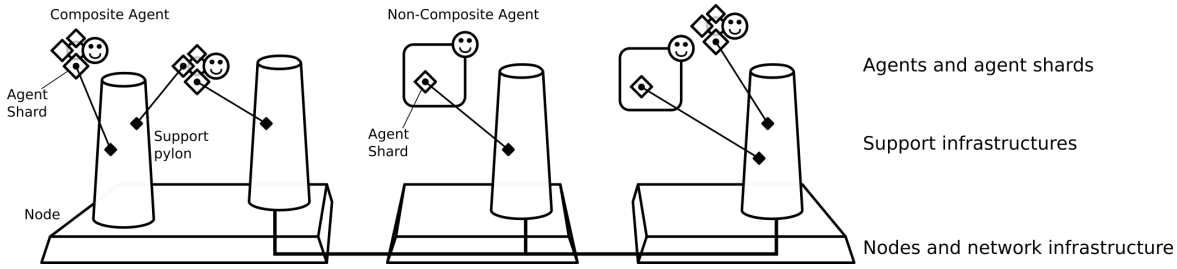

Figure 1. A graphical view on the relations between agents, agent shards, pylons, and nodes. Agents contain agent shards, which assist agents in connecting to pylons. Pylons deployed across nodes for support infrastructures, ensuring communication and discovery services in the system.

must be done transparently to agents. The agent implementation must be the same even if the communication method changes.

Similarly, the system must be able to support different implementations for agents. This means that Jade agents could co-exist with ad-hoc agents implemented in plain Java, and with agents implemented using other agent development tools, such as Jiac.

High flexibility means that some configuration is needed. However, it is our intention that the effort to specify the configuration must be proportional with the complexity of the system itself, without the need to create complex deployment files just to deploy a few agents.

The architecture of FLASH-MAS is centered on the concept of *Entity*. An entity is a part of the system that exists for a relatively long time during the system's execution (is persistent). All entities have some common features:

- an Entity has a *Configuration*, which is specified at deployment of the system or at run-time;
- an Entity is *loaded* (instantiated) by a *Loader*, using the given configuration; a default loader is offered, which just instantiates the class of the entity;
- an Entity has a *state*, which can be *running* or *not running*; the entity can be *started* and *stopped*;
- an Entity has a *context* within which it executes, which is formed by a set of Entities which contain it;
- an Entity may have a *name*, which can be used to identify it, but that is not mandatory.

The predefined entities are the following:

- *Nodes* – each machine that runs FLASH-MAS agents. The nodes loads all other entities that run initially on that node, and they execute in its context.
- *Pylons* – implementations for support services such as communication, discovery, mobility, remote deployment, etc. Different instances of the same type of pylon deployed across the system form a *support infrastructure* and are able to communicate with each other (e.g. a WebSocket server and its clients, an underlying Jade instance, etc). Each pylon executes in the context of a node.
- *Agents* – entities that act autonomously according to the implementation given by the developer. FLASH-MAS also offers an implementation for agents – the *Composite Agent* – which is composed of Agent Shards that process and post events on an event queue[10]. Each agent executes in the context of a node and zero, one, or several pylons.
- *Agent Shards* – for Composite Agents, components that implement various features, such as messaging, monitoring, control, mobility, etc. Each shard executes in the context of an agent. Agent Shards can also be used by non-Composite Agents in order to interact with pylons. Agent shards are characterized by their *designation*, which indicates what services they can offer (e.g. messaging, discovery, knowledge management, etc). The designation is associated to the interface presented by the shard to the agent.

Figure 1 shows the relations between entities in a system, and how agents use shards in order to interact with the pylons executing on the same node. Figure 2 shows how the various entities in a FLASH-MAS deployment are organized, as an UML diagram.

The essence of the FLASH-MAS architecture is the relation between agents, shards, and pylons. For instance, let us take the case of an agent A who wishes to use the services offered by the support infrastructure S (e.g. a discovery service). Agent A executes on node N, where there exists also a pylon P belonging to support infrastructure S. In order to use the services offered by the infrastructure, agent A must include an agent shard C, which is specific to infrastructure S. Implementation-wise, agent shard C is derived from an abstract shard offering discovery services, so the actual implementation of the agent does no change. Agent A must not necessarily be implemented as a Composite Agent; it may have any implementation, and can contain an instance of shard C as a member. If we wish to change the support infrastructure S with a different implementation S', all that must be done, from the perspective of agent A, is to replace the reference to shard C with a reference to another instance C', with the same designation and the same interface. The transition is seamless.

A FLASH-MAS deployment is organized as a hierarchy of contexts.

- all nodes execute in the context of a virtual, network-scale deployment;
- pylons execute in the context of the local node;

[10]For further details on Composite Agents, please consult our work on tATAmI-2 [12].

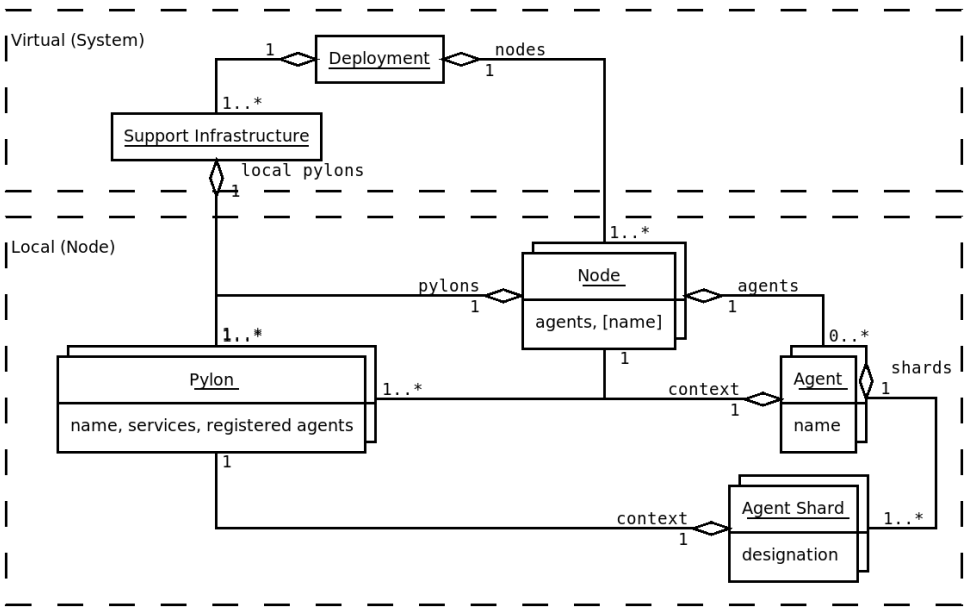

Figure 2. UML diagram of a FLASH-MAS deployment, containing a virtual level and an actual level. Entities only exist (execute) at the local level, but they belong to virtual structures which spread on multiple physical machines. While the entities are presented in the diagram as objects, they are actually implementations of the respective interfaces (`Node`, `Pylon`, `Agent`, `AgentShard`).

- pylons belong to virtual, network-scale support infrastructures;
- agents execute in the context of one or more pylons (and, by transitivity, in the context of the local node);
- agent shards execute in the context of agents (and, by transitivity, in the context of their corresponding pylon).

In order for agents to load the appropriate shards at run-time, the agent can ask the pylon (which is its context) which is the *recommended* shard implementation for a specific designation, and dynamically load the instance accordingly.

A. Interaction

There needs to be a standard API for each agent shard designation that we wish to be dynamically and seamlessly changeable, which all implementations need to respect. In order to achieve that, several invariants exist.

First, all interaction between entities is done by means of message-like structures called *waves*[11]. A *wave* has one or more destinations, can contain elements of metadata, and has a content. A wave is directly translatable to and easily readable as plain text. The metadata in the wave is specified as key-value pairs. A destination of the *wave* is a list of identifiers, in which the first identifier is unique throughout the system (e.g. the identifier of an agent), the second identifier is unique inside the entity identified by the first, etc.

Messages are particular instances of *waves*. For instance, if an agent $agentA$ contains a shard $monitoring$, which contains another application-specific module $statistics$ (also an entity), a message could be addressed just to agent $agentA$, or it could be addressed directly to module $statistics$ by specifying `agentA/monitoring/statistics` as a destination.

Beside the use of *waves* for interactions, there are some other invariants:

- all agents are identified by their name; agents are visible to other communication infrastructures by prefixing the name of the infrastructure to the name of the agent, with a separator (in a manner similar to Jade). If the infrastructure has no name, its agents (or other entities therein) will not be visible to the outside.
- messages are addressed using the identifiers of agents; messages have at least one destination and a content, and can contain additional fields according to the FIPA standard [9].
- discovery of services offered by agents or artifacts is done (as specified by the FIPA standard [10]) by category (functioning in a similar manner with shard designations), and/or by name, and/or by specifying various key-value pairs for relevant parameters.

B. Open System

The core feature of FLASH-MAS is flexibility, and what differentiates it from other frameworks is how it tries as much as possible not force the user into a particular architecture for the implementation. As such, the set of entity types that can be part of a deployment is open. While nodes, pylons and agents need to exist, the developer is free to add other entity types, such as artifacts, groups, or others.

Artifacts can be included in the system in a similar fashion to agents, with the difference that they will not feature and pro-active behavior. Integration with the system will be ensured by adding messaging (or similar) shards to artifacts, so that they

[11] Based on one of the meanings of the word wave, which is to use a hand gesture towards to signal something to someone.

```
<?xml version="1.0" encoding="UTF-8"?>
<deployment xmlns="http://flash.xqhs.net/deployment-schema" [...]>
  <package>examples.composite</package>
  <loader for="agent:composite"/>

  <agent name="AgentA" kind="composite">
    <shard name="messaging" />
    <shard name="monitoring" classpath="MonitoringTestFeature" />
  </agent>
  <agent>
    <parameter name="name" value="AgentB" />
    <parameter name="kind" value="composite" />
    <shard name="messaging" />
    <shard name="monitoring" classpath="MonitoringTestFeature" />
  </agent>
</deployment>
```

Figure 3. An example deployment file specifying two agents, both containing a messaging shard and a monitoring shard.

can receive messages (or other types of *waves*) from agents. For developers that wish to see the environment as a whole, a pylon representing the environment can be deployed on every node, in the context of existing communication pylons, and as a context of agents; each agent would feature a shard that allows the agent to interact with the environment, through actions and perceptions represented as *waves*.

Groups of agents (e.g. agent arrays or organizations) can be included as entities in the context of pylons (and the greater context of nodes), and as a context to agents. When the deployment is loaded, it will be the job of the agent array to load and configure the agents according to its own rules. However, agents will continue to benefit from the FLASH-MAS structure, by having access to the pylons in the context of which they have been created. If we talk about creating organizations, in the context of pylons supporting communication, an additional layer of pylons can be created that support the assignment of roles to agent and handle the distributed aspect of the organization (such as communication restrictions).

Some scenarios may need to include entities that are not able to run FLASH-MAS agents at all. Examples include very simple sensors needing gateways, or assistive robots with proprietary software, communicating by means of ROS[12]. In these case, special application-specific entities will need to be deployed to interface with the devices. Although such entities need to be built, they can be easily integrated into the deployment and they can automatically have access to services offered by support infrastructures, by means of the context mechanisms.

It is important to point out that any new entities can be introduced without modifying the core code of the framework. Their contextual placement is established by the deployment configuration, and their implementation is loaded dynamically based on their configuration parameters.

IV. DEPLOYMENT TOOLS

One of the main goals of FLASH-MAS is to be easy to deploy for simple scenarios, but also allow the deployment of complex scenarios.

A deployment is specified by combining two structures: an XML deployment file and command line arguments. Both structures affect the same configuration, with command line arguments taking precedence. An example deployment file is specified in Figure 3. The deployment can be specified entirely programmatically as well, through the use of a hierarchical Multimap structure[13].

Both the deployment file and the command line describe the deployment as a hierarchy of contexts, containing the descriptions of nodes, pylons, agents and shards and any other needed entities. Each entity is configured using key-value parameters. Only agents need to be explicitly specified. If no nodes are specified, a default node is created, containing all other entities in the deployment; if no pylons are specified, the default support infrastructure is used, supporting only communication inside the local node.

A complex system of class finding is used in order to find the implementation of entities and then loading them dynamically, using their type and name, without the need for the developer to specify the classpath for each entity. The developer may specify root packages where implementations may be located. For instance, with the `impl` additional package specified, the implementation for a Comp Agent will be searched at the classpaths:

`core.CompAgent`, `core.agent.CompAgent`, `core.agent.comp.CompAgent`, `core.agent.comp.Agent`, `impl.CompAgent`, `impl.agent.CompAgent`, `impl.agent.comp.CompAgent`, and `impl.agent.comp.Agent`.

```
CLI arguments    ::= deployment-file? category-descr*
category-descr   ::= '-' category element element-descr
element          ::= kind ':' name | kind ':' | name
element-descr    ::= (par ':' val | par)* category-descr*
```

Figure 4. The grammar of the FLASH-MAS command line.

Since the command line is linear, a hierarchical behavior is obtained by "navigating" through the levels of the hierarchy.

[12]Robot Operating System (https://en.wikipedia.org/wiki/Robot_Operating_System, accessed March 2019)

[13]Multimap (https://en.wikipedia.org/wiki/Multimap, accessed March 2019)

The level is changed with every new category description (see Figure 4. A category can be either a predefined name (such as for the deployment itself, the deployment schema, etc) or the name of an entity (predefined or new). If the category is predefined, the level is switched to that category (which can be below or above the current level); if the category is not predefined, a new instance of that entity is created as a subordinate of the current entity.

Entities can have a name, or a kind (a subtype of the entity, e.g. *composite* agent), or both. Entities are configured through parameters given as key-value pairs, although some parameters may not need an actual value.

For example, a very simple deployment consisting of two named Java agents can be obtained using only the command line:

```
flash -package simple -agent agentA -agent agentB
```

A more complex deployment can be with the command line:

```
flash -loader agent:composite -pylon websocket:
server:ws.org -agent -shard messaging -shard
knowledge -pair state:initial
```

In the deployment described above, a Composite Agent is launched in the context of a WebSocket pylon that has its server at `ws.org`; the agent contains two shards: one for messaging, as recommended by the pylon, and one for knowledge management, with an initial knowledge item stating that the agent is in its initial state.

The command line arguments can also complete the configuration in the deployment file, identifying entities through a combination of hierarchical navigation and name identification. For instance, take the command line:

```
flash deployment.xml -package additional.goal
-agent agentA -shard goalOriented -goal survive(15,
seconds) -agent agentB -monitoring server:ws.org
```

In the example above, consider that file `deployment.xml` has the content in Figure 3. The CLI arguments will add a package, a shard for `agentA` (presumably from the added package) and will set some configuration for the shard, and will also add a new parameter value for the `monitoring` shard of `agentB`.

V. Elements of Comparison

The purpose of FLASH-MAS is to provide a lightweight framework for deploying flexible multi-agent systems in an effortless manner.

In contrast with Jade, the intention is to easily change the communication infrastructure that the agents use, without changing the code of the agent. Moreover, more freedom is allowed in implementing the agent, since agents only need to define a method for receiving events from shards, as opposed to working in a behavior-driven manner. While command-line deployment is also possible in Jade, FLASH-MAS offers a more powerful CLI and a complementary XML file.

In contrast with Java EE-based implementations, FLASH-MAS offers more choice in how the system is organized. While communication through web services is elegant and modern, that can be covered in FLASH-MAS as well. WebSocket communication is available in a centralized manner in which a node is considered the server and all other pylons send messages through that node. Moreover, pylons could each deploy a lightweight web server and use a discovery algorithm to locate the nodes for each agent. Several such algorithms exist [27]. The goal of FLASH-MAS is also to use only plain Java, even without an absolute need for reflection and dynamic class loading, in order to support a great variety of platforms with the same code.

In contrast with multi-agent simulation frameworks, we attempt to offer an architecture and implementations that enable a distributed deployment, on a heterogeneous system (not in a HPC environment). However, the goal is to use FLASH-MAS for fast simulation of many agents as well. Since the communication method is not fixed, agents can also communicate using local infrastructure offering immediate message delivery. Moreover, although Composite Agents use an event processing thread for each agent, the behavior that is desired to be simulated can be easily embedded in agents that are run sequentially by nodes. This is thanks to the fact that the FLASH-MAS architecture does not force a particular structure on the agents.

VI. Conclusion and Future Work

This paper proposes an architecture for a multi-agent system framework supporting a lightweight and effortless deployment of systems which are open in terms of the types of entities that can be part of the system.

The proposed architecture relies on a set of predefined entities, with the possibility of introducing any number of new entities just by specifying them in the deployment and allowing the framework to locate their implementation. All entities exist in the context of one another, allowing subordinate entities to access the features offered by the entities containing them.

The framework features a flexible, easy to use deployment model, that can use both XML deployment files as well as the CLI in order to specify the entire deployment of the system.

Future work consists of improved integration, implementation of additional features for the framework, such as monitoring and remote control of entities, groups, organizations, and artifacts. Experiments will be performed comparing the performance of FLASH-MAS with that of other frameworks, such as Jade, Jiac and Repast.

References

[1] D. Tapia, A. Abraham, J. Corchado, and R. Alonso, "Agents and ambient intelligence: case studies," *Journal of Ambient Intelligence and Humanized Computing*, vol. 1, no. 2, pp. 85–93, 2010.

[2] D. Calvaresi, M. Marinoni, A. Sturm, M. Schumacher, and G. Buttazzo, "The challenge of real-time multi-agent systems for enabling iot and cps," in *Proceedings of the international conference on web intelligence*. ACM, 2017, pp. 356–364.

[3] D. Calvaresi, D. Cesarini, P. Sernani, M. Marinoni, A. F. Dragoni, and A. Sturm, "Exploring the ambient assisted living domain: a systematic review," *Journal of Ambient Intelligence and Humanized Computing*, vol. 8, no. 2, pp. 239–257, 2017.

[4] T. Ogino, S. Kitagami, T. Suganuma, and N. Shiratori, "A multi-agent based flexible iot edge computing architecture harmonizing its control with cloud computing," *International Journal of Networking and Computing*, vol. 8, no. 2, pp. 218–239, 2018.

[5] O. Boissier, R. H. Bordini, J. F. Hübner, A. Ricci, and A. Santi, "Multi-agent oriented programming with JaCaMo," *Science of Computer Programming*, vol. 78, no. 6, pp. 747–761, 2013.

[6] F. Bellifemine, A. Poggi, and G. Rimassa, "Developing multi-agent systems with JADE," *Intelligent Agents VII Agent Theories Architectures and Languages*, pp. 42–47, 2001.

[7] A. H. Ngu, M. Gutierrez, V. Metsis, S. Nepal, and Q. Z. Sheng, "Iot middleware: A survey on issues and enabling technologies," *IEEE Internet of Things Journal*, vol. 4, no. 1, pp. 1–20, 2017.

[8] A. Ricci, M. Viroli, and A. Omicini, "Give agents their artifacts: the a&a approach for engineering working environments in mas," in *Proceedings of the 6th international joint conference on Autonomous agents and multiagent systems*. ACM, 2007, p. 150.

[9] FIPA, "FIPA ACL message structure specification," December 2002. [Online]. Available: http://www.fipa.org/specs/fipa00061/SC00061G.html

[10] ——, "FIPA abstract architecture specification," December 2002. [Online]. Available: http://fipa.org/specs/fipa00001/SC00001L.html

[11] A. Olaru, M.-T. Benea, A. El Fallah Seghrouchni, and A. M. Florea, "tATAmI: A platform for the development and deployment of agent-based ami applications," in *Proceedings of ANT-2015, the 6th International Conference on Ambient Systems, Networks and Technologies, June 2-5, London, United Kingdom*, ser. Procedia Computer Science, E. Shakshuki, Ed., vol. 52. Elsevier, June 2015, pp. 476–483. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1877050915008182

[12] A. Olaru, "tATAmI-2 – a flexible framework for modular agents," in *Proceedings of AgTAmI 2015, the International Workshop on Agent Technology for Ambient Intelligence, the 20th International Conference on Control Systems and Computer Science, May 27-29, Bucharest, Romania*, I. Dumitrache, A. M. Florea, F. Pop, and A. Dumitrascu, Eds., vol. 2. IEEE Computer Society, May 2015, pp. 703–710. [Online]. Available: http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=7168503

[13] L. Braubach and A. Pokahr, "Jadex active components framework-BDI agents for disaster rescue coordination." *Software Agents, Agent Systems and Their Applications*, vol. 32, pp. 57–84, 2012.

[14] N. Howden, R. Rönnquist, A. Hodgson, and A. Lucas, "Jack intelligent agents-summary of an agent infrastructure," in *5th International conference on autonomous agents*, 2001.

[15] M. P. Georgeff and A. L. Lansky, "Reactive reasoning and planning." in *AAAI*, vol. 87, 1987, pp. 677–682.

[16] M. d'Inverno, M. Luck, M. Georgeff, D. Kinny, and M. Wooldridge, "The dMARS architecture: A specification of the distributed multi-agent reasoning system," *Autonomous Agents and Multi-Agent Systems*, vol. 9, no. 1-2, pp. 5–53, 2004.

[17] M. Lützenberger, T. Küster, T. Konnerth, A. Thiele, N. Masuch, A. Heßler, J. Keiser, M. Burkhardt, S. Kaiser, and S. Albayrak, "JIAC V: A MAS framework for industrial applications," in *Proceedings of the 2013 international conference on Autonomous agents and multi-agent systems*. International Foundation for Autonomous Agents and Multiagent Systems, 2013, pp. 1189–1190.

[18] D. Mitrović, M. Ivanović, M. Vidaković, and Z. Budimac, "A scalable distributed architecture for web-based software agents," in *Computational Collective Intelligence*. Springer, 2015, pp. 67–76.

[19] R. H. Bordini, J. F. Hübner, and M. Wooldridge, *Programming multi-agent systems in AgentSpeak using Jason*. John Wiley & Sons, 2007, vol. 8.

[20] S. Russell, H. Jordan, G. M. O'Hare, and R. W. Collier, "Agent factory: a framework for prototyping logic-based AOP languages," in *Multiagent System Technologies*. Springer, 2011, pp. 125–136.

[21] S. Abar, G. K. Theodoropoulos, P. Lemarinier, and G. M. O'Hare, "Agent based modelling and simulation tools: a review of the state-of-art software," *Computer Science Review*, vol. 24, pp. 13–33, 2017.

[22] N. Collier and M. North, "Repast hpc: A platform for large-scale agentbased modeling," *Large-Scale Computing Techniques for Complex System Simulations*, pp. 81–110, 2011.

[23] F. Lorig, N. Dammenhayn, D.-J. Müller, and I. J. Timm, "Measuring and comparing scalability of agent-based simulation frameworks," in *German Conference on Multiagent System Technologies*. Springer, 2015, pp. 42–60.

[24] G. Fortino, W. Russo, C. Savaglio, W. Shen, and M. Zhou, "Agent-oriented cooperative smart objects: From iot system design to implementation," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, no. 99, pp. 1–18, 2017.

[25] I. Ayala, M. Amor, and L. Fuentes, "The sol agent platform: Enabling group communication and interoperability of self-configuring agents in the internet of things," *Journal of Ambient Intelligence and Smart Environments*, vol. 7, no. 2, pp. 243–269, 2015.

[26] M. Dragone, "Component & service-based agent systems: Self-OSGi," in *ICAART 2012 - Proceedings of the 4th International Conference on Agents and Artificial Intelligence, Volume 1 - Artificial Intelligence, Vilamoura, Algarve, Portugal, 6-8 February, 2012*. Citeseer, 2012, pp. 200–210.

[27] A. Rawat, R. Sushil, and L. Sharm, "Mobile agent communication protocols: a comparative study," in *Computational Intelligence in Data Mining-Volume 1*. Springer, 2015, pp. 131–141.