The entity-operation model for practical multi-entity deployment*

Andrei Olaru, Gabriel Nicolae, and Adina Magda Florea

Department of Computer Science and Engineering, University Politehnica of Bucharest, 313 Splaiul Independentei, 060042 Bucharest, Romania
{andrei.olaru,adina.florea}@upb.ro, gabriel.nicolae2907@stud.acs.upb.ro

Abstract. In the world of multi-agent system (MAS) frameworks, developers are many times forced into a fixed and reduced array of abstractions, with limited options in expressive modeling of all the components of a MAS. For instance, in JADE, the most popular agent framework, developers are limited to using agents as sole abstraction for all elements of the MAS. These limitations hinder interoperability, the deployment of open, heterogeneous systems, and the use of agents in complex scenarios involving a great variety of elements such as physical devices, context managers, services, and communication infrastructures.
We introduce the *entity-operation model* for multi-agent systems, as an approach to integrate all elements in the MAS deployment as first-class entities in the MAS model, to support heterogeneity and flexibility in the implementation, and to achieve context-aware access control to the functionalities offered by entities.
We present a formalization of the model, together with mechanisms for authorizing operations and for routing operation calls in the MAS. We discuss the entity-operation model in relation to other existing MAS frameworks, and we give insight into implementation challenges which arose when integrating the model with the FLASH-MAS framework.

Keywords: Multi-agent systems · Multi-agent frameworks · Communication infrastructure interoperability.

1 Introduction

Agents and multi-agent systems (MAS) are used in a great variety of domains, including cloud computing, networks security and routing, social networks, robotics, complex systems modeling, the Internet of Things (IoT), Ambient Assisted Living (AAL), smart cities, smart grids, and simulation [1,6,8].

A MAS framework is meant to save the developer from the task of implementing several functionalities such as inter-agent communication, resource discovery,

* This work was supported by a grant of the Ministry of Research, Innovation and Digitization, CNCS - UEFISCDI, project number PN-III-P1-1.1-TE-2021-1422, within PNCDI III. This work has been partially funded by UEFISCDI project Cornet (1/2018, PN-III-P3-3.6-H2020-2016-0120).

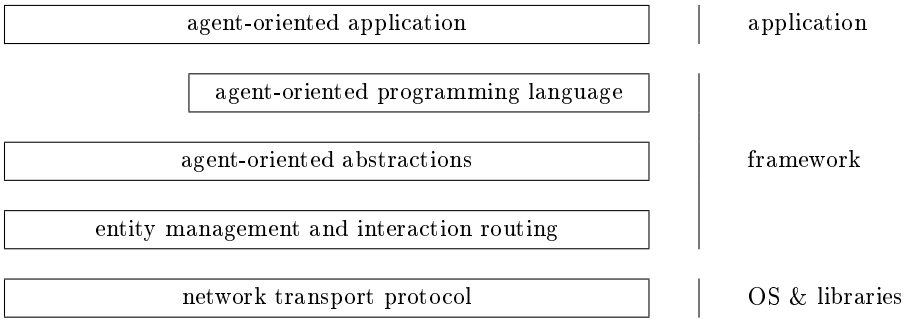

Fig. 1. A view on the layers existing in a MAS framework. Agent-oriented abstractions are entities accessible by the developer, such as agents, artifacts, and nodes.

agent mobility, internal agent event processing and internal agent organization, as well as deployment, control and monitoring of agents.

There are several frameworks which facilitate the design, modeling, and simulation or deployment of MAS [6,19]. Among them there are JADE, SPADE, JIAC, JACK, JaCaMo, PLACE, FLAME, MASON, Repast, of which some allow the deployment on distributed networks of devices, whereas others (such as the last three) are Agent-Based Modeling Simulation (ABMS) platforms, which enable high-performance simulation of large numbers of agents. Some frameworks, such as SARL and MET4FoF [23,9], support both high-performance local simulation, as well as distributed deployment.

A MAS framework normally offers to the developer an API that allows the creation and management of various agent-oriented abstractions (such as nodes, agents, artifacts), the definition of various aspects of the environment (such as a physical space, tools agents can use, services available), as well as the interaction between all those elements. For instance, JADE offers agents as sole abstraction for the persistent components in a MAS agentification, with containers available as references to the nodes in the deployment. JaCaMo implements the Agent & Artifacts (A&A) model and offers artifacts as abstractions for any aspect of the environment [22]. ABMS frameworks generally split the modeling between event-driven agents and a space-based environment model.

In terms of the infrastructure for communication and services, it is fixed in most frameworks, some, such as SARL and JaCaMo, offering the choice between local and network-distributed message or event routing. Figure 1 shows a perspective on the layers of a MAS framework: the MAS application is built on top of agent-oriented abstractions offered by the framework, and potentially an agent-oriented programming language (such as Jason or SARL); the framework deals with the management of entity lifecycle and with the interaction between entities.

This approach has some important limitations. First, any new type of entity that the developer may need to model must be built on top of the abstractions

offered by the framework, adding complexity andleading to two levels of modeling: the agent-oriented model of the framework and the agent-oriented model of the application. For instance, if the developer of a distributed AOP application implemented in JADE wishes to model artifacts, artifacts would have to be implemented as agents, leading to having "agent" agents and "artifact" agents. The alternatives are either to use JaCaMo, forcing the developer to use Jason for agents, or to interoperate CArtAgO with JADE, which brings a different framework into the mix. Similarly, in JaCaMo, organizations are managed via artifacts, such that an organization is not a first-class entity in itself. Other entities may be needed, such as context managers – entities managing an activity or a smart space – which need to perform some proactive actions, such as sending notifications or checking for overlaps, but do not have mental states, making them different from both agents and artifacts.

The second limitation relates to communication infrastructures in distributed network deployments, which are also related to discovery of resources. Most frameworks have the communication infrastructure fixed and it is impractical to change it. In a complex distributed deployment, using different communication infrastructures in different parts of the system brings challenges in modeling and forces the developer into creating multiple models for multiple frameworks, incurring overhead in interoperating them.

In this paper we introduce a model for the entity-interaction layer of a multi-agent system framework, which relies on a uniform representation of entities and their operations. We call it the *entity-operation model*. In this model, any persistent component in the multi-agent system is explicitly modeled as an entity, and its functionality is accessible via a number of operations, which can have arguments and restrictions.

This way, the model of the MAS can contain, as first-class entities, not only agents, artifacts, or organizations, but also communication infrastructures, directory services, context managers, entities at the sub-agent level (components of agents, such as behaviors), and so on. All of these would be accessible to other entities by means of a uniform underlying interface, while allowing each type of entity its own set of specific operations. By using a uniform approach to model any entity, a deployed system can welcome, at runtime, new entities of new types, supporting openness and heterogeneity.

Abstractions that make up existing agent-oriented models can be implemented using the entity-operation model. For instance, instead of implementing organizations via artifacts (as in JaCaMo), an organization can be implemented directly as an entity in its own right, potentially distributed over several nodes. A developer using an agent framework can use "standard" entity types and interactions (e.g., agents and artifacts) without needing to know about entities and operations, but also has the possibility to create new types of entities, to access other entities via operation calls, and to use the full capabilities of any of the entities in the framework. The entity-operation model is interoperable with other agent-oriented models but supports a flexible approach to the modeling of individual entities.

A natural addition to the entity-operation model was a means to control the access to operations. We have created a context-based access model, where access is authorized to entities having specific relations to other, known, entities.

We have successfully implemented the entity-operation model in FLASH-MAS[1] [17], reusing existing blocks and models and implementing a scenario demonstrating how the entity-operation model can be used for context-based access to elements in a smart environment.

We have devised an ambient intelligence scenario which we will use throughout the paper, in which a person interacts with entities in a smart building: Andreea is a master student at the Department of Computer Science and Engineering. She is also working as a teaching assistant for undergraduate students in the same department. It's the middle of January and Andreea is going to teach an Operating Systems lecture on a Monday morning in the new smart building in the campus. To reduce the energy bill, the heating is turned off over the weekend and it must be started by an authorized person in each room on Monday. Andreea lives quite far from the university, and it takes her about an hour to get there. Before leaving her home, she uses the mobile app to check the temperature of the classroom and she remotely turns on the heating. When she arrives in the building, she uses her smartphone to unlock the door to the lecture room and turns on the lights. Being a second-year master student, Andreea has a cloud computing class in another room in the same building later on. She resides in Room 308 where she has a desk. She prints some notes using the printer in Room 308 and then goes to her class. This time she is a student, so she won't be able to perform the same actions as before – for instance, she will not be able to unlock the room where she has classes as a student. She needs to present a semester project, so she temporarily receives control of the projector in the room, to show her slides.

A second use-case that we address is a platform in which agents are able to exchange pre-trained machine learning (ML) models, evaluate or train them further, and exchange information about their experiences. In this scenario, ML models and their descriptions are first-class abstractions, and act as sub-agent entities that agents can use in their activity and that agents can send or receive. They have a reactive aspect – answering to queries – but also a pro-active (but not autonomous) aspect, as they can report on the status of their training process or report problems with their functionality.

The paper is organized as follows. In the next section we discuss existing frameworks and models for distributed multi-agent systems. After the presentation of the model in Section 3 and implementation challenges in Section 4, we discuss the advantages and appropriateness of using the entity-operation model for multi-agent frameworks in Section 5. The last section draws the conclusions.

[1] The Fast and Lightweight Multi-Agent Shell. The source code is available at https://github.com/andreiolaru-ro/FLASH-MAS

2 Related Work

Pal et al [19] survey the current state of framework development, detailing both the application domains of frameworks and their implementation language, as well as their development status and distribution license. An important distinction is between open-source and commercial platforms. A related work lists only 16 projects as in-development general-purpose platforms, combining platforms for distributed deployment, platforms for ABM simulation, and AOP languages. Kravari and Bassiliades [14] survey the development status, license, adherence to standards, ease of deployment, and security for several agent frameworks, concluding that JADE remains the most popular framework. They observe that when choosing a framework for deploying MAS, developers and researchers must select and be limited by application domain, programming language, and learnability. This is why our intention is to develop a more general, easy to use MAS framework.

Agent-based simulation tools are surveyed by Abar [1], Rousset [24] and Lorig [15]. The most popular and giving a high level of performance are the Repast suite and D-MASON. JADE is shown to have very little applicability for ABMS, because of the lack of support for synchronization and for HPC-specific communication. More flexibility in the interaction model could have made JADE a valid option for AMBS.

Cardoso and Ferrando [6] provide a fresh systematic literature review on AOP languages, which are many times related to their respective frameworks. Most languages are based on AgentSpeak [20] (ASTRA, Jason and related languages) or on JADE.

JADE [3] is, by far, the most popular framework for distributed deployment of multi-agent systems. It offers communication, directory and discovery services, agent migration, and a specific, behavior-based structure for agents. In terms of communication, it relies on TCP/IP by default, but other communication methods are available at deployment time, with agent code changes necessary [7].JADE offers no abstraction other than the agents (potentially offering services), with the framework also abstracted as standard agent instances. Nodes, communication services, and directory services are accessed in different manners – nodes via direct methods, communication via methods in the agent, and directory services via FIPA-ACL messages with specific content [10]. Many other frameworks are based on or inspired by JADE and strive to be FIPA-compliant. Jadex [5] adds support for BDI agent modeling. SPADE [13] is developed in Python and uses XMPP/Jabber as a communication method, featuring a GUI for monitoring agents, giving the advantage of easier interoperation with ML libraries. PADE [16,26] is also implemented in Python and uses Twisted for communication, hence supporting multiple protocols. JACOSO is a JADE-based implementation of the ACOSO Methodology for the development of IoT systems [11]. Apart from Smart Objects, which it agentifies as JADE agents, it builds additional abstractions for other elements in IoT scenarios, such as tasks (as sub-agent entities), events, and a variety of managers and adapters whose properties do not fit in the agent model, showing the need to be able to integrate

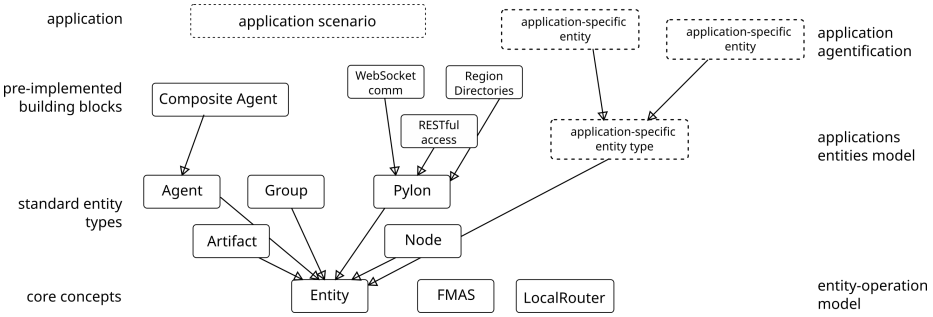

Fig. 2. A view on the object-oriented class hierarchy in a scenario using the entity-operation model. There are several layers of abstraction, with the most abstract at the bottom. See also Section 3.2. Dotted borders are used for application-specific models and entities defined by the MAS developer.

new abstractions in the agent framework without the need to create additional layers.

JaCaMo [4] is another popular MAS framework, which combines the Jason AOP language, based on AgentSpeak and Prolog, with the CArtAgO implementation of the Agents & Artifacts (A&A) model for the environment, and with the MOISE implementation for roles and organizations. JaCaMo rests on strong theoretical foundations and can be deployed both locally and in JADE-based distributed setups. The distinction made between agents and artifacts is very strong, and their development paths diverge from modeling phase. Workspaces exist as virtual entities spanning multiple nodes, but do not have an embodiment with its own code. No new abstractions can be created in JaCaMo, without basing them on existing entities. Initial steps have been taken towards making JaCaMo BDI compatible with ABM simulation, via JaCaMo-SIM [21].

Janus [12] is a language-independent platform, but targeted mainly at executing the SARL [23] AOP language. It supports event-driven interaction, and in deployments over a local computer network (via the Janusnode variant) it broadcasts these events to all agents. While combining the A&A approach with the distributed approach, it is not adequate for message-based applications, nor does it offer services such as directory or service discovery.

Met4FoF [9] is a recent, in-development MAS framework oriented towards streaming data from sensors, written in Python. It offers specialized modules for stream management, buffering, and redundancy. It offers only agents as abstractions for persistent entities, with additional data- and stream-related abstractions.

3 The entity-operation model

When implementing complex scenarios using an agent-based approach, a question that is raised frequently is "*what should this be modeled as?*". This is some-

what related to the question that arises in an open system when a new entity is introduced and the other entities in the system ask questions like "*what is this? how should I interact with this?*". When using a MAS framework, the set of possible entity types is fixed to what the framework offers. New entity types will have to be implemented via existing entity types.

A framework using the entity-operation model does not restrict the developer in this way and allows the deployment of any type of entity as a first-class entity. The framework itself is very *thin* – it only specifies an interaction model, with all the rest being modules that can have various implementations.

Entities. The model that we propose posits that all persistent elements in a MAS are represented as *entities*. The central principle is that entities are *persistent*. Secondly, entities need, in general, to be *accessible*; as such, they expose *operations*, which other entities may *call*; not every operation is available to any entity, as we will detail further on. Third, entities are *autonomous*, in that they can decide how they react to operation calls. All entities should have a unique identifier. In a complex deployment, although mechanisms for name shortening and caching can be used, any entity should be uniquely identified by its *URI*.

Entities may be *local* to (running inside) a physical node, or may be *distributed* across multiple nodes. Distributed entities must have a local *embodiment* on each node where they are present. For instance, in JADE, containers are the embodiment of the JADE platform.

Our goal is to use this model to describe all elements in a running, deployed MAS. That means that, beside agents and components representing aspects of the environment, elements such as nodes and communication infrastructures should also be implemented as entities; similarly, any interaction between entities, be it an interaction between two agents, but also an interaction between an agent and a node, or an agent and an organization, should be performed via operations. Communication infrastructures are also accessed via operations, allowing for more uniformity and for flexibility in the implementation of communication mechanisms.

We do not model in any way the *inside* of entities. We look for interoperability and mutual understanding, but an entity may work in various manners on the inside. Of course, one has the possibility to model the inside of an entity using other entities, as for instance some agents in FLASH-MAS are composed of shards, which are also modeled as entities. FLASH-MAS offers an implementation for *composite agents*, whose behavior is modeled by the shards that are added to the agent [17].

Using entities does not mean we forfeit existing models based on agents or on agents and artifacts. Rather, we offer a uniform underlying model (or metamodel) that underpins the actual model used by the application. This brings unity to the technical implementation of the entities and the opportunity to easily switch between different approaches to modeling.

In our scenario, we model as user's agent as an agent entity, the smart lock, the smart light, the heating appliance, and the temperature sensor as artifacts,

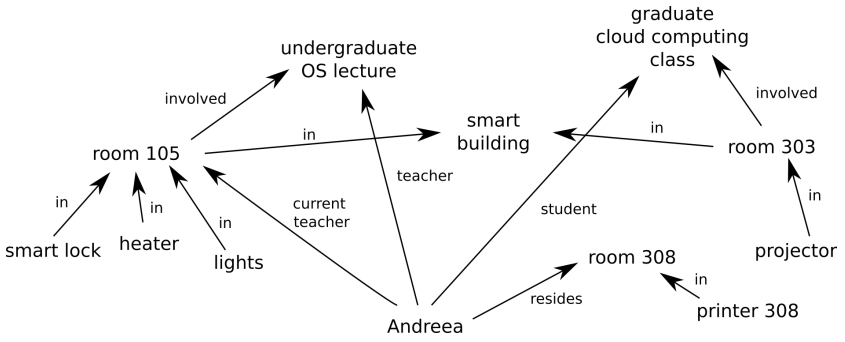

Fig. 3. A perspective on the relations in the running scenario, at the moment when Andreea teaches the undergraduate class on operating systems.

and the entities managing the room and the two teaching activities as context managers – non-autonomous agents with a more complex behavior.

Operations. The model that we propose posits that any interaction between two entities is performed via *operations*, with one entity *calling* an operation of the other entity. Moreover, any interaction between a core element of the framework and an entity should be performed via an operation of the entity.

Entities are expected to have a `list` operation, which returns a description of all the operations available to any other entity. While this can, in the future, be used to semantically search for an appropriate operation, it can currently be used to *duck type* an entity, based on its available operations.

In keeping with the concepts in the A&A and web services models, operations can also have *return values*, which are returned to the initiator of the operation call.

In our scenario, the user's agent can query the temperature sensor to find the temperature in the room, can instruct the smart lock to unlock the room, and can connect to a wireless projector. All of these, of course, if the user is authorized to perform the operations. For instance, the `print` operation of the printer in Room 308 may be available to people who are physically in the room, or are in general residing in that room.

Relations. Access to operations can be *restricted* by using *relations*. Relations link entities in a similar manner to semantic triples in RDF[2]. Relations can express, for instance, that a certain device is in a particular room, or that a user has a role in a particular activity, or that a service runs on a particular node. Once initiated, relations must be accepted by entities at both ends, and can be canceled by any of the two entities.

Relations can be used to restrict which entities are allowed to call a given operation. For instance, a door for a room in a smart building can only be unlocked by the personal agent of a user who is teaching the lecture taking place in that room in that given interval.

[2] Semantic triple https://en.wikipedia.org/wiki/Semantic_triple

In our example, relations describe the placement of devices, the location of the user, and the role of the user in the current context. See Figure 3 for a perspective on the relations in the scenario, at the moment when Andreea teaches the undergraduate class.

Context tokens. An entity calling an operation which has restrictions must *prove* that it indeed has the required relations to other entities. As such, we introduce the idea of *context tokens* – tokens which are a proof of context. Each token is a document containing the statement of a relationship, the timestamp of the document, and an expiration time. To ensure authenticity, tokens can be required by the callee to be cryptographically signed by an authorized entity involved in the relationship.

An entity will receive context tokens periodically from other entities, proving their relationship. As a caller of an operation, it will send, in the operation call, all relevant tokens, proving that it is indeed authorized to call the operation. The tokens must still be valid for successful authorization. The expiration period of context tokens is related to the nature of the relation and is proportional to the time a relation is expected to last. The quality of being employed by an organization can be re-certified (by the emission of a context token) once every month, whereas the property of being in a given room should be recertified once every minute. It is the entity managing a given context that decides the expiration period of context tokens.

For instance, in our scenario, there is a printer in Room 308. When a user with a device enters the room, a relation is created by the local access point between the user's agent / device and the entity managing the room. While the user remains in the room, the user's agent receives, periodically, a context token proving the relation. When the user wishes to call the `print` operation, the user's agent will know that the operation is restricted to users in the room, so the operation call that is sent will also contain the most recent context token proving that the user is in the room; the token is signed by the room manager. The printer already has the public key of the room manager, since one of the restrictions on one of the operations of the printer involves the room manager. It can check the context token and approve the operation.

In the example with Andreea unlocking the door as a teacher, the smart lock entity lists an operation `unlock`, with the restriction that the caller must have the role *current_teacher* in the current room and at the current time. In a different exchange, the entity managing the lecture informs the room who will be teacher in the current time slot. The entity managing the room creates a *current_teacher* relation with the agent of the user, periodically sending to the user a context token proving the relation. When calling the `unlock` operation, this context token will be included, and it will be verified by the smart lock.

3.1 Formalization sketch

From an omniscient point of view, we define a fully modeled multi-entity system, using the entity-operation model, as a tuple $\langle EE, RR \rangle$, where EE is the set of

entities and RR is the set of *relations*. We have:

$$EE = \{E \mid E = \langle ID_E, Operations_E \rangle\}$$

$$RR = \{\langle from, relation, to \rangle\}, \text{with } from, to \in EE$$

An *operation* $O \in Operations_E$, with $E \in EE$, is defined as:

$$O = \langle Name_O, Description_O, Arguments_O, Result_O, Restrictions_O \rangle$$

The tuple contains the name of the operation, its description, the description of the arguments, return value, and restrictions on the entities which may call it. The description of the operation can have any form, but a semantic description is more suitable. The description of the arguments is of the form $Arguments_O = \{\langle Name, Description \rangle\}$. In the simplest implementation of the model, the *Description* can be as simple as the type of the argument, and the description of the result the type of the returned value.

The restrictions on an operation are defined as a logical operation on relations. Take, for example, the printer located in Room 308 in our running scenario. It has one operation (`print`), which should be available to any entity E in the same room ($E \prec_{located} Room308$, and to anyone who is a resident in that room ($E \prec_{resides} Room308$). So, for an entity E to be allowed to call the `print` operation, it should be true that $E \prec_{located} Room308 \vee E \prec_{resides} Room308$.

The restrictions can be formalized as a disjunctive normal form on positive literals, each literal representing a relation, but replacing the formula $E \prec_{\langle\rangle} E1$ with the pair $(\prec_{\langle\rangle}, E1)$. That is, for an operation O:

$$Restrictions_O \subseteq \{Conjunction \mid Conjunction \subseteq \mathcal{R} \times EE\}, \text{with}$$

$$\mathcal{R} = \{relation \mid \langle *, relation, * \rangle \in RR\} - \text{the set of all relation names}$$

As such, the `print` operation has a restriction that looks like: $\{\{(\prec_{located}, Room308\}, \{(\prec_{resides}, Room308)\}\}$

Of course, no single entity in a distributed system has an omniscient view on all other entities (or else it would be a bottleneck and a single point of failure), so, in practice, the system is formed of the set of entities EE, each entity keeping track of its relation to other entities:

$$\forall E \in EE \,.\, E = \langle ID_E, Operations_E, Outgoing_E, Incoming_E \rangle, \text{with}$$

$$Outgoing_E = \{(relation, E_{to}) \mid \langle E, relation, E_{to} \rangle \in RR\}$$

$$Incoming_E = \{(relation, E_{from}) \mid \langle E_{from}, relation, E \rangle \in RR\}$$

An *operation call* is an object containing the *caller*, the *callee*, the name of the operation, the arguments for the operation, relevant information about the relations of the caller, and whether a return value should be sent back to the caller (if the operation supports it):

$$call = \langle E_{Source}, E_{Destination}, Name_{Op}, \{Arguments\}, \{Tokens\}, send\text{-}result \rangle$$

where $E_{Source}, E_{Destination} \in EE$, $Name_{Op}$ is an identifier for the operation, *Arguments* are the argument values for the operation, *send-result* is a boolean value, and *Tokens* are the *context tokens*.

3.2 Predefined entities and relations

As stated in the Introduction, a developer does not need to create new types of entities or to call operations directly. A layer of predefined entities may prove sufficient for many MAS applications (see also Figure 2). This has several advantages: (1) a framework based on the entity-operation model can be used exactly like a "normal", existing framework; (2) given enough predefined entities, a framework based on the entity-operation model can be used like *any* of the standard frameworks; and (3) is needed, the developer can still define new types of entities or new implementations of existing entities. Entities in existing MAS models can be represented in the entity-operation model as follows.

An **agent** has a `receive` operation, which allows it to receive messages from any other entity. The implementation of agents can define a `send` method which constructs an operation call directed at another agent. In the A&A model, the implementation can also contain methods for accessing artifacts and operations to allow notifications from artifacts. As there are many approaches to what an agent is, multiple "agent" entity types can be defined depending on the application, each with its own set of operations, defining what an agent is in that approach.

An **artifact** works just as in the A&A meta-model, exposing operations to any entity in the appropriate workspace. Workspaces are distributed entities, and entities are bound to workspaces by means of dedicated relations.

A **pylon** that is the embodiment of a *communication infrastructure*, can receive `route` operation calls from any entity, and it can attempt to route the operation call to its target entity, which may be on a different machine.

A **pylon** that offers directory and discovery services presents the `register` and `search` operations.

A **node** offers a `load&start` operation to any entity which *executes* on the node or which has *authority* over the node. Nodes supporting migration offer a `receive_agent` operation which enables agents to migrate to that node.

Sub-agent entities, for instance **shards** in FLASH-MAS [17], must have a fast two-way means of interaction with their container agent. As such, a shard must offer a `signal_agent_event` operation to receive events from the agent, and an agent supporting shards must offer a `post_event` operation to receive events from its shards. The same mechanisms can be used to build holonic systems. In a model similar to Jade, behaviors could also be modeled as sub-agent entities.

Machine learning models (pre-trained) can be represented as entities having a `get_result` operation. As sub-agent entities, they can be sent from one agent to another, they can be cloned, or they can migrate with an agent from one node to another, using the same migration mechanism as agents. They can have a pro-active aspect which allows them to notify other entities about the status of the training process.

Entities are expected to have a `list` operation, which returns the names and descriptions of operations available to other entities.

Any entity should offer a set of operations which allows it to interact easily with the framework, while also abiding to the model. These operations – `start`,

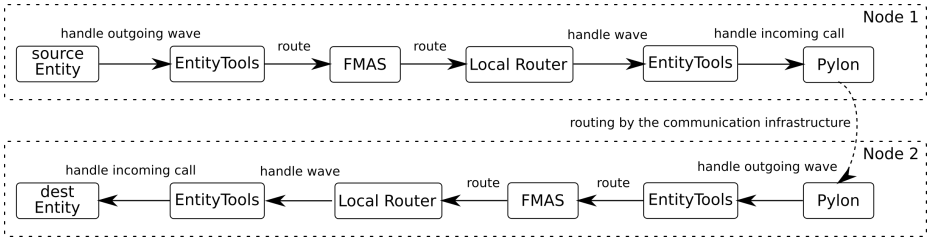

Fig. 4. The path taken by an operation call between entities situated on different nodes.

`stop`, and `isRunning`, should only be available to entities authorized to *control* that entity (e.g., the node on which the entity is executing, or the *owner* of the entity).

4 Implementation challenges and results

We have implemented the entity-operation model in FLASH-MAS, a Java-based framework which offers tools for the deployment of complex, distributed multi-agent scenarios, in which the implementation of any entity is customizable. We have re-written the core code of FLASH-MAS so that all entities use the entity-operation model. As such, some challenges arose, and we will present in this section how we solved them.

To abide to the entity-operation model, any object which represents an entity must implement the `EntityAPI` interface, which specifies a minimal number of methods:

- `connectTools` gives the entity a reference to the `EntityTools` instance which will connect it to the framework. Using the `EntityTools` instance, the entity can register (or obtain) and ID with the framework.
- `getID` returns the ID of the entity.
- `handleIncomingOperationCall` is called whenever an operation call is sent to the entity. The method is called by the `EntityTools` instance associated with the entity, which has previously checked if the operation is correctly accessed.
- `handleRelationChange` is called by the `EntityTools` instance whenever a relation involving this entity is created or destroyed.

While representing all persistent things in a MAS as entities, and since entities can be implemented in any way by the developer, there is a need for something to bind the entities together, help manage them, and ensure that operation calls reach their intended destination. Hence, on any JVM where FLASH-MAS is running there is a singleton object called `FMAS`, representing the framework. To ensure correct encapsulation and to restrict access to powerful `FMAS` functionality we take example from the internal implementation of JADE and create, for each entity integrated into the system, an instance of the `EntityTools` class, which helps entities interface with the framework. The `EntityTools` instance manages

the list of the entity's operations, the access to those operations, and the relations incoming to or outgoing from the entity.

Entities interact via operation calls. There are, however, other types of interactions, which are not direct interactions between entities, and cannot be represented as operation calls. These are: the return value sent as a result of an operation call; the initiation or removal of a relation between two entities; and the acceptance or rejection of a new relation between entities. All these interactions have a destination and a source or a return path, so we can model them collectively as a concept that we call *wave*.

Waves must be routed so that they reach their destination, sometimes across the local network or the Internet. In keeping with the flexibility offered by FLASH-MAS, there is no restriction on the communication method used, as long as the communication infrastructure can deliver a wave to the node where a destination entity is located. In line with the principles of the entity-operation model, interaction infrastructures are embodied by pylons, which are also entities. Routing waves is handled by an entity which is directly linked to the `FMAS` instance, called the *Local Router*.

To route a wave, the Local Router uses the following algorithm:

- if the destination is registered with the local `FMAS` instance, the wave is routed directly to its destination (via the `EntityTools` instance associated with the destination);
- otherwise, the list of local entities which offer the `route` operation is used to look for an appropriate router;
- the wave is sent via the first of these entities that executes the `route` operation successfully.

A detailed view of this process is shown in Figure 4. While the wave passes through several entities, the only decision points are in the *Local Router* instances, the rest being only method calls. The advantage of this process is, however, that routing waves can be done using no matter which communication infrastructure. Currently, FLASH-MAS has implementation for communication via WebSocket, RESTful web services, distributed region-based mesh, and ROS. Having an implementation for entities which is agnostic to the communication mechanism opens the path towards using the same codebase for performing high-performance simulations and deploying entities in a distributed setup. In FLASH-MAS we have already performed experiments with using MPI-based communication.

In FLASH-MAS we have strived not only to have the ability to select the communication infrastructure at deployment, and make agent (and, in general, entity) code agnostic of the mechanism used for communication, but also to be able to support the deployment and interoperation of multiple communication infrastructures in the same system. This is also possible in the entity-operation model. Following the principles in previous work [18,25], *bridge* entities can offer the `route` operation, the same as pylons of communication infrastructures. A bridge will register in two (or more) communication infrastructures and will act in each one as a sink for the other(s), ensuring waves can travel between any

two nodes in the bridged infrastructures, in a transparent manner for the other entities.

Some MAS frameworks, including JADE, support mobile agents, which can migrate from one node to another. This poses a particularly difficult challenge to the framework, as the agent needs to interrupt its activity, get serialized, transferred to another node, deserialized, and resume execution. In FLASH-MAS, entity mobility needs support from both *inside* and *outside* the entity. The entity must ensure that it suspends its activity correctly, and serialization is also done inside the entity. Once serialized, the entity sends the package as the argument of an operation call to the destination node, which, if it supports migration, deserializes the entity and registers it with the local `FMAS` instance, leaving to the entity to resume its activity in the correct manner.

It is arguable that, when using cryptographically signed context tokens, checking the tokens can incur a significant performance penalty. We have developed, however, a mechanism to avoid this penalty for the cases where performance is essential. First, many scenarios do not require at all that access to the most used operations is controlled, for instance in a scenario using agents which exchange messages. Secondly, the most important performance penalty is brought when operation calls are routed inside the same node. This is particularly of issue in the case of sub-agent entities (shards in FLASH-MAS) to which only their container agent has access, but which are expected to exchange calls with their agent frequently. Let us take the example of a shard which should post an event to its agent. The shard calls the agent's operation and attaches a context token proving that the shard belongs, indeed, to that agent; but the context token has been generated by the agent, on the same node. This means that the local `FMAS` instance can check the token only by hashing it and comparing the result against a list of active tokens and their hashes, without needed to verify the signature. Full verification is still needed, however, when calls are exchanged between different network nodes.

We have validated the viability of the entity-operation model by implementing the scenario presented in the Introduction. Our goal was to verify that we can use the entity-operation model to implement all the described processes and to perform functional testing of the context-based access model. This stage of validation was successful.

In the implementation of the scenario, we have created agents for Andreea and other students, context managers for the two lectures, for the smart building, and for the three rooms, and artifacts for the various devices – smart lock, heater, lights, and printer. Relations have been created, especially the ones representing the role of Andreea as a teacher for one course and as a student for the other course. When, in simulated time, Andreea's lecture as a teacher approached, a relation was created by the course manager between Andreea and the room manager. She was now authorized to control some of the devices in the room. When a relation was created between Andreea and the smart building, the `unlock` operation of the smart lock became available to her. After the end of the lecture,

her relation with the room was removed, so the operations became unavailable again.

The implementation shows that, indeed, a variety of entities can be implemented using the proposed model, more properly than just implementing them all as "agents". It showed that the context-based access model can be used to limit the availability of operations.

5 Discussion

The idea of having objects distributed across the network, offering operations that can be called, is not new. However, our model is directed specifically towards autonomous entities. Compared, for instance, to Java RMI, the entity-operation model, and the various implementations for actual communication between nodes, increases flexibility as it avoids reliance on a single interaction mechanism.

Having entities and operations is similar, and can be replicated by, having agents which send messages from one to another. It offers, however, a lot more expressivity in modeling the entities in the system, reserving the agent abstraction for truly pro-active, autonomous entities, without abusing them to implement any of the persistent entities in a deployment.

Especially when using web service communication, a deployment using the entity-operation model can be likened to a set of web-services offering various operations. Our approach, however, allows various interaction methods, some of which can be more lightweight than deploying a web server on every node.

Essentially, what we strive to offer with the entity-operation model is choice and expression power. A MAS developer should not be forced into making the choice of implementing an entity as a framework-offered abstraction that is not appropriate for that entity, and this choice should not lead to a development path that is so far from the other types of abstractions that it is difficult to return to the decision point. The developer should be able to choose from a wide array of available abstractions and, when needed, to be able to create their own first-class abstractions, and then use that set of abstractions for the entities in the applications. Let us take a few examples.

A *context manager* handles the interactions between other entities and a smart space or a smart activity [2]. For instance, a context manager keeps track of the entities which are a part of that context, e.g., which are physically in that space, or are part of that activity. In our scenario, the context manager of the room keeps track of the users in the room, or users authorized to control the room devices; the context manager of the course (as an activity) sends updates to the entities involved in the activity. A context manager is not a proper agent, in terms of an entity which has goals, and which achieves goals by executing actions in a plan; it rather manages aspects of the environment. However, it cannot easily be implemented as an artifact (in the sense of the A&A model) because an artifact cannot create relations between it and agents proactively, because agents need to first *focus* on the artifact. When the teacher (or some faculty staff) adds

students to a course, some agent would have to send a message to the students' agents, and then the agents would have to focus on the context manager. In the entity-operation model, a context manager entity is able to create relations to agents (that they can approve or not), even if it is not modeled as an agent.

A *broadcast group* (similar to a mailing list) relays messages sent by one of its members to all the other members in the group. Again, a broadcast group would not be properly modeled as an agent. However, it cannot be modeled as an A&A artifact either – artifacts can notify agents only of observable properties or via signals. For a broadcast group, members of the group would not receive the messages in the group as signals, not as messages, needing a different processing path inside the agent. In the entity-operation model, a broadcast group is implemented as an entity which agents can `join` and then it can simply call the `receive` operation of agents each time it needs to broadcast a message.

6 Conclusions and Future Work

We introduce the entity-operation model as a practical approach to the uniform implementation of the various abstractions offered by a MAS framework. The model has been created with the desire to both offer to the MAS developer an array of available abstractions, but also to allow the developer to change previous choices regarding the modeling of the scenario, and to create new types of abstractions, if one needs it. The model is enriched with a context-based access model for operations.

Using the entity-operation model brings a series of advantages, like the possibility to interact with all types of abstractions in a MAS and to model explicitly the communications and services infrastructures. Another advantage is the ability to create sub-agent or supra-agent entities, such as shards and organizations, respectively, or to create holonic structures.

A current development direction is to fully integrate machine learning models as sub-agent entities, while interoperating, using the principles of the entity-operation model, with ML frameworks written in Python.

The next steps in this research are to build entity implementations that use the entity-operation model and are compatible with JADE, JaCaMo, and other popular MAS frameworks. Our short-term goals are to be able to run JADE agent code on other communication infrastructures, to use various agent implementations with the JADE communication infrastructures, to support Jason as an AOP language, and to interoperate with CArtAgO and MOISE.

References

1. Abar, S., Theodoropoulos, G.K., Lemarinier, P., O'Hare, G.M.: Agent based modelling and simulation tools: a review of the state-of-art software. Computer Science Review **24**, 13–33 (2017)
2. Baljak, V., Benea, M.T., El Fallah Seghrouchni, A., Herpson, C., Honiden, S., Nguyen, T.T.N., Olaru, A., Shimizu, R., Tei, K., Toriumi, S.: S-CLAIM: An agent-based programming language for AmI, a smart-room

case study. In: Proceedings of ANT 2012, The 3rd International Conference on Ambient Systems, Networks and Technologies, August 27-29, Niagara Falls, Ontario, Canada. Procedia Computer Science, vol. 10, pp. 30–37. Elsevier (2012). https://doi.org/10.1016/j.procs.2012.06.008, http://www.sciencedirect.com/science/article/pii/S1877050912003651

3. Bellifemine, F., Poggi, A., Rimassa, G.: JADE - a FIPA-compliant agent framework. In: Proceedings of PAAM. vol. 99, pp. 97–108. Citeseer (1999)
4. Boissier, O., Bordini, R.H., Hübner, J.F., Ricci, A., Santi, A.: Multi-agent oriented programming with JaCaMo. Science of Computer Programming **78**(6), 747–761 (2013)
5. Braubach, L., Pokahr, A.: Jadex active components framework-BDI agents for disaster rescue coordination. Software Agents, Agent Systems and Their Applications **32**, 57–84 (2012)
6. Cardoso, R.C., Ferrando, A.: A review of agent-based programming for multi-agent systems. Computers **10**(2), 16 (2021)
7. Curry, E., Chambers, D., Lyons, G.: A jms message transport protocol for the jade platform. In: IEEE/WIC International Conference on Intelligent Agent Technology, 2003. IAT 2003. pp. 596–600. IEEE (2003)
8. Dorri, A., Kanhere, S.S., Jurdak, R.: Multi-agent systems: A survey. IEEE Access **6**, 28573–28593 (2018)
9. Dorst, T., Eichstädt, S., Schneider, T., Schütze, A.: Propagation of uncertainty for an adaptive linear approximation algorithm. SMSI 2020-System of Units and Metrological Infrastructure pp. 366–367 (2020)
10. FIPA: FIPA ACL message structure specification (December 2002), http://www.fipa.org/specs/fipa00061/SC00061G.html
11. Fortino, G., Russo, W., Savaglio, C., Shen, W., Zhou, M.: Agent-oriented cooperative smart objects: From iot system design to implementation. IEEE Transactions on Systems, Man, and Cybernetics: Systems (99), 1–18 (2017)
12. Galland, S., Rodriguez, S., Gaud, N.: Run-time environment for the SARL agent-programming language: the example of the janus platform. Future Generation Computer Systems **107**, 1105–1115 (2020)
13. Gregori, M.E., Cámara, J.P., Bada, G.A.: A jabber-based multi-agent system platform. In: Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems. pp. 1282–1284 (2006)
14. Kravari, K., Bassiliades, N.: A survey of agent platforms. Journal of Artificial Societies and Social Simulation **18**(1), 11 (2015)
15. Lorig, F., Dammenhayn, N., Müller, D.J., Timm, I.J.: Measuring and comparing scalability of agent-based simulation frameworks. In: German Conference on Multiagent System Technologies. pp. 42–60. Springer (2015)
16. Melo, L.S., Sampaio, R.F., Leão, R.P.S., Barroso, G.C., Bezerra, J.R.: Python-based multi-agent platform for application on power grids. International Transactions on Electrical Energy Systems **29**(6), e12012 (2019)
17. Olaru, A., Sorici, A., Florea, A.M.: A flexible and lightweight agent deployment architecture. In: 2019 22nd International Conference on Control Systems and Computer Science (CSCS), Bucharest, Romania, 28-30 May 2019. pp. 251–258. IEEE (2019). https://doi.org/10.1109/CSCS.2019.00048, https://ieeexplore.ieee.org/abstract/document/8744845/
18. Olaru, A., Florea, A.M.: A framework for integrating heterogeneous agent communication platforms. In: Proceedings of ACSys 2015, the 12th Workshop on

Agents for Complex Systems, in conjunction with SYNASC 2015, the 17th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing, Timisoara, Romania, September 21-24. pp. 399 – 406. IEEE Xplore (2015). https://doi.org/http://dx.doi.org/10.1109/SYNASC.2015.66
19. Pal, C.V., Leon, F., Paprzycki, M., Ganzha, M.: A review of platforms for the development of agent systems. arXiv preprint arXiv:2007.08961 (2020)
20. Rao, A.S.: AgentSpeak (L): BDI agents speak out in a logical computable language. In: European workshop on modelling autonomous agents in a multi-agent world. pp. 42–55. Springer (1996)
21. Ricci, A., Croatti, A., Bordini, R., Hübner, J., Boissier, O.: Exploiting simulation for mas programming and engineering-the jacamo-sim platform. In: 8th International Workshop on Engineering Multi-Agent Systems (EMAS 2020) (2020)
22. Ricci, A., Viroli, M., Omicini, A.: 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. p. 150. ACM (2007)
23. Rodriguez, S., Gaud, N., Galland, S.: SARL: a general-purpose agent-oriented programming language. In: 2014 IEEE/WIC/ACM International Joint Conferences on Web Intelligence (WI) and Intelligent Agent Technologies (IAT). vol. 3, pp. 103–110. IEEE (2014)
24. Rousset, A., Herrmann, B., Lang, C., Philippe, L.: A survey on parallel and distributed multi-agent systems for high performance computing simulations. Computer Science Review **22**, 27–46 (2016)
25. Suguri, H., Kodama, E., Miyazaki, M., Kaji, I.: Assuring interoperability between heterogeneous multi-agent systems with a gateway agent. In: 7th IEEE International Symposium on High Assurance Systems Engineering, 2002. Proceedings. pp. 167–170. IEEE (2002)
26. Tom, R.J., Sankaranarayanan, S., Rodrigues, J.J.: Agent negotiation in an iot-fog based power distribution system for demand reduction. Sustainable Energy Technologies and Assessments **38**, 100653 (2020)