applied sciences

Article

ARPS: A Framework for Development, Simulation, Evaluation, and Deployment of Multi-Agent Systems

Thiago Coelho Prado and Michael Bauer *

Department of Computer Science, Western University, London, ON N6A 3K7, Canada; tprado2@uwo.ca
* Correspondence: bauer@uwo.ca; Tel.: +1-519-661-3562

Received: 18 September 2019; Accepted: 18 October 2019; Published: 23 October 2019

Abstract: Multi-Agent Systems (MASs) are often used to optimize the use of the resources available in an environment. A flaw during the modelling phase or an unanticipated scenario during their execution, however, can make the agents behave not as planned. As a consequence, the resources can be poorly utilized and operate sub-optimized, but it can also bring the resources into an unexpected state. Such problems can be mitigated if there is a controlled environment to test the agents' behaviour before deployment. To this end, a simulated environment provides not only a way to test the agents' behaviour under different common scenarios but test them as well in adverse and rare state conditions. With this in mind, we have developed ARPS, an open-source framework that can be used to design computational agents, evaluate them in a simulated environment modelled after a real one, and then deploy and manage them seamlessly in the actual environment when the results of their evaluation are satisfactory.

Keywords: multi-agent systems; discrete event simulator; interoperability; agent and multi-agent applications

1. Introduction

In environments where resource management is critical, software agents can be employed to optimize those resources. When the task to accomplish this is too complex to be carried out by a single agent, multiple interacting agents can be used to achieve effectiveness. The Multi-Agent System (MAS) approach has been successfully applied in many domains, including: helping controllers in Air Traffic Control for making decisions based on the aircrafts' fuel availability, flight plan, weather, and any other relevant data [1]; optimizing distributed generators in smart grids for energy production, storage, and distribution [2]; or finding better arrangements for the components in manufacturing plants to increase the throughput and optimize material usage [3].

Many frameworks and toolkits enable the implementation of a MAS. They are not designed to provide seamless means of evaluation of the outcome of the agents in the system under certain scenarios before deployment. A flaw during the design and implementation of the agents, or an unpredicted state of the environment where they are acting, can not only interfere with the achievement of their goals, but can also bring the environment into a unexpected situation with unforeseen consequences.

Some work with MASs has made use of simulations to aid deployed agents to update their plans according to the current state of the environment. This means that the simulation is used as a planner tool, meant to help the agent to make a decision under a specified near future scenario rather than using the simulation in the process of designing the agent behaviour before deployment. This strategy has been applied to the field of multiple Unmanned Aerial Vehicles (UAV) [4], where communication is required for coordination and failures related to it can make the entities unreachable. The system simulates possible scenarios where communication is unavailable and an action by the UAV is expected. Another example is the simulation component used to detect conflicts and inconsistencies of resource allocation during the high-level planning in a manufacturing plant [5].

There is no general purpose MAS framework, to the best of our knowledge, that integrates the process of validation of the agents in a simulated environment before their deployment. To address this, we have developed ARPS, an open-source framework available at https://gitlab.com/arps/arps/ under MIT LICENSE [6], to seamlessly design, implement, assess the agents, and deploy the MAS after the results meet established criteria. ARPS stands for some of the core properties of agents: autonomy, reactivity, proactivity, and social ability. We use the management of resources of a data centre to illustrate our approach and how it can be used in other domains.

In Section 2 we cover related work. Following this, Section 3 describes the background and architecture of the framework. The process of the implementation of the MAS to manage an experimental scenario is shown in Section 4. In Section 5 we discuss the findings, limitations, and future directions.

2. Related Work

There are multiple toolkits, platforms, and frameworks available for creating a MAS. In this section, we will describe a few of them. The works here by no means represent the only alternatives for creating MASs. For other options, refer to surveys on this topic, such as the one presented in [7].

The works reviewed here implement common aspects of a MAS to work in an actual environment, such as communication, interoperability, storage, security, and resource discovery. Therefore, the users can focus on the definition of the agents and their behaviour, how they are organized, and how they interact to solve a problem.

Among the popular MAS frameworks, JADE (also known as Java Agent Development Framework) [8] was created to address the problem of interoperability and provide an environment for the development of agents. It has no domain dependent requirement as seen in the other solutions. According to the authors, such dependencies were obstacles for the adoption of MAS technologies at the time of its conception. JADE simplifies the implementation of multi-agent systems through a middleware compliant with the FIPA (Foundation for Intelligent Physical Agents) specifications [9], a standard proposed for interoperability of agents. The JADE authors argue that is industry-driven and currently the most known FIPA-compliant agent platform in the academic and industrial community.

The A-Globe platform [10] is designed for testing experimental scenarios featuring agents' position that requires a Geographical Information System (GIS, though agents may suffer from communication inaccessibility either because of the spatial distance of the agents or by broken links. Because it is a closed-environment, interoperability is not one of the concerns of this platform. Hence, it is not fully compliant with the FIPA-specifications on inter-platform communication, albeit it provides compliance with the Agent Communication Language (ACL), a structure for composing messages exchanged by agents.

Based on the fulfillment of requirements such as robustness, security, and the ability to ensure that a partial solution can be executed when an optimized one is not found due to constraints, DARPA funded a project called Cougaar [11] (Cognitive Agent Architecture). This agent platform was created to offer specialized support for logistics-related problems. This platform is also not FIPA-compliant. It aims to facilitate the development of agent-based applications that are complex, large scale and distributed.

Other solutions offer more components built on top of the existing agent-based platforms. The Jadex BDI Agent System [12] follows the Belief Desire Intention (BDI) model [13] and facilitates intelligent agent construction over other middleware such as JADE. It has been used to build applications in different domains, such as simulation, scheduling, and mobile computing. The programming model of Jadex allows for designing an application as hierarchical decomposition of components interacting via services and thus helps to make complexity manageable. These components can be used in concurrent and dynamic distributed systems. Another example is JaCaMo [14]. It is an interpreter for an extended version of AgentSpeak, a BDI agent-oriented logic programming language [15]. It is a platform that integrates three projects with different MAS-related paradigm models for development: Jason [16] for agent development; Moise [17] for agent organization; and CArtAgO [18] for environment-oriented programming.

Lastly, the MadKit [19] offers a modular and scalable multiagent platform. Its central aspect is the ability to organize agents in groups and roles aiming at the development of artificial societies. It is closely related to our approach in the sense that has a simulator component. Nonetheless, the simulation and evaluation are not seamless in the workflow, leaving the responsibility of this integration to the user.

Previous work has often presented MAS tools that provide a base infrastructure to implement agents. These approaches also looked to help address other specific problems, such as interoperability by defining standards, robustness and security. In some cases, the work sought to introduce approaches to enrich the design process, provide clear definition of how agents are organized and what their roles are. However, these solutions lack a component that would allow a developer to assess the behaviour of the agents in a simulated environment before actual deployment. Further, this step should be as seamless as possible, i.e., it should take no or few modifications to alter agents in the simulated environment in order to make them run in the actual environment. This is the gap that ARPS fills.

Agent-Based Modelling (ABM)

One of the main solutions available to simulate complex systems is known as Agent-based modelling (ABM). It is employed in domains such as social sciences, biology, ecology, engineering, and economics. There are many platforms to implement ABM [20,21]. Among them, we list a few widely adopted examples. The Swarm package [22] provides object-oriented libraries of reusable components for building models and analyzing, displaying, and controlling experiments on those models. NetLogo [23] is a software that provides packages to simulate multi-agent in environments. It has a significant user community, it is highly documented, and they provide many demos. MASON [24] is a discrete event multi-agent simulation toolkit. It was designed to serve as the basis for a wide range of multi-agent simulation tasks ranging from swarm robotics to machine learning to social complexity environments. The Repast platform, initially developed as a tool to be used in social sciences, is a family of agent-based modelling and simulation platforms. Currently, not only does it provide a package to be used in regular environments, such as desktops, and laptops [25], but also has an advanced version to be used in HPC environments [26] to simulate more demanding scenarios.

The ABM and MAS concepts are closely related to each other since both are agent-based. The difference is that the modelling in ABM aims to gain insight about emergent properties in complex adaptive systems while MAS focus on actual agents [27,28].

Our framework aims to provide an environment to combine ABM properties, such as the ability to model agents in a simulated environment and observe their behaviour, with MAS's ability to implement and deploy the agents evaluated during the simulation in the action environment. The integration of both approaches can yield positive results. The framework in [29], features this combination. It enables mobile agents to work simultaneously both in the actual and virtualized agent platform to enable large-scale simulation to observe unknown emergence behaviour. This is accomplished by having the agents in the actual environment collecting data in real-time, and improving the simulation, which in turn can have its outcome used in decision making. In our case, we need a separate simulator component to enable the study of the agents' behaviour without disrupting the environment before their deployment. This then enables a developer to leverage the construction of reliable physical agents by evaluating their interactions and the effects of their actions. Because simulation can involve the compression of time, it is possible to simulate many different scenarios efficiently. Also, during the simulation, it is possible to create unexpected scenarios to see how the agents perform. This can be desirable in areas that already employ ABM for resource management during a disaster, such as [30], that could be extended to have software agents that could direct resources where it is necessary. We are aiming to combine the characteristics of both simulation and development to offer an alternative for creating solutions in resource management using MAS.

Figure 1 illustrates our workflow to achieve this integration. A user/admin can create the environment, its resources, and the agents' models, and define how they are organized. The MAS environment is generated, and the user can simulate a scenario. After analyzing the simulation results, the user can decide to refine the models or apply the agents' models to the actual environment. The output of this environment can also be used as feedback to improve the existing models and create a more reliable simulation.

As we have discussed, none of the previoius MAS frameworks have a simulator component that allows this workflow without having to reimplement the concepts defined during the simulation into the actual environment.

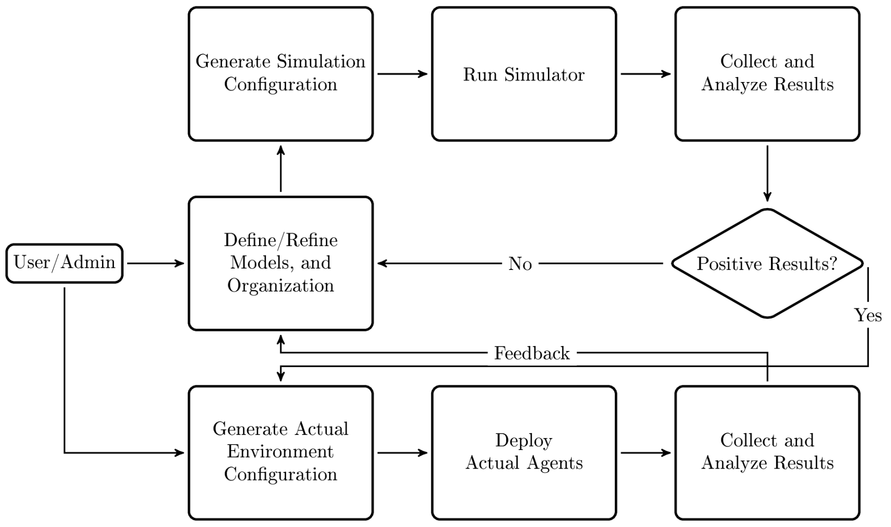
Generate Simulation Configuration
Run Simulator
Collect and Analyze Results
User/Admin
Define/Refine Models, and Organization
No
Positive Results?
Yes
Feedback
Generate Actual Environment Configuration
Deploy Actual Agents
Collect and Analyze Results

Figure 1. Workflow for MAS conception, evaluation, and deployment.

3. Our Approach

Below we describe the background of the framework, our design choices, and present the architecture.

3.1. Background

Data centres present complex dynamic environments where many resources are allocated to provide guaranteed, reliable services. These resources are not only related directly to the computer systems, such as processing power, storage capacity, and network, but also to the supporting equipment and environmental control. Data centre administrators face a daunting task in trying to manage them optimally. There are impacts when these resources are misconfigured or overallocated on the total cost of ownership because of excess power consumption or idle resources. Data centre operators can also incur financial penalties due to the broken guarantees related to service delivery.

One proposed way to tackle this problem is the adoption of autonomic computing architectures and strategies. An autonomic approach aims to embody the idea of self-management, which in turn can be realized by decomposing it into other sub-properties, such as self-configuration, self-healing, self-optimization, and self-protection [31]. This separation of concerns can be managed by decentralized autonomous agents that may interact with each other, optimizing local resources to achieve global optimization, as exemplified in [32], where a MAS manages the number of hosts available to process workloads depending on the demand.

One problem faced in the employment of MASs in data centres is the impracticality of having an actual data centre available to evaluate the effectiveness of the policies governing agents due to costs and security concerns. In some cases, data centre simulators, such as that described in [33], have been developed to evaluate the possible impacts of the autonomic agents policies in the data centre before implementing and deploying them in the real environment. Even so, there is a gap, however, in the development between the simulation step and the implementation and deployment of the actual agents. To address this problem, a framework was developed using initially the concepts of the aforementioned simulator with the additional feature of employing the assessed policies during the simulation to drive the software agents that will manage the actual resources in the environment.

Contrary to most of the works related to MAS platforms presented in the previous sections, our approach was first developed to solve a domain-specific problem focused on self-management of resources. During the development, the generic components were extracted to allow more flexibility during the implementation and the test of the proposed solutions. This resulted in the general purpose ARPS framework to enable MAS.

The generalization of the scenarios where our MAS framework can apply, although it is not limited by, is illustrated in Figure 2. As can be seen, there is a complex system with resources (R_n), grouped by environments ($Environment_m$), that are affected by external events (E). The events occur at a variable or fixed interval in this system. Without any management, these resources can be in a suboptimal state. To overcome this problem, agents (A_i), driven by policies, are employed to monitor resources or modify them using available touchpoints. The agents can be reactive, proactive. They can act in isolation, or they can communicate with each other for cooperation, coordination, or negotiation.

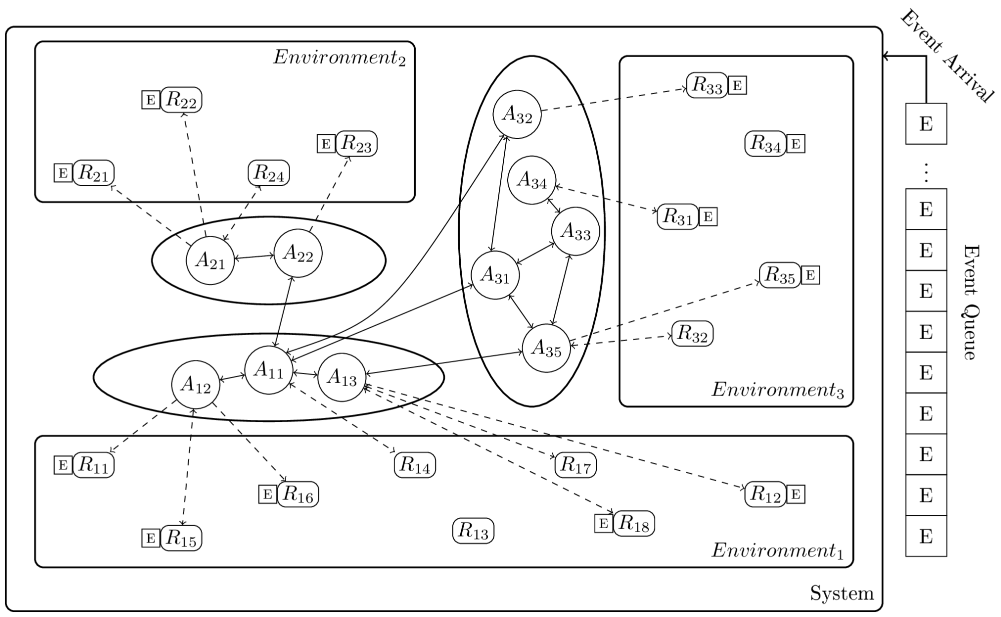

Figure 2. Scenario for resources being modified by external events while agents manage them.

The framework is being currently developed in Python 3, a high-level multiplatform language that can be deployed in myriad host platforms, including Internet of Things (IoT) devices [34,35]. The users can install the framework from source code or as a Python package, available at https://pypi.org/project/arps/. This means that, when implementing the agents for a specific domain, the user needs to implement all the components using Python. Albeit there is no single programming language that can be applied in every domain effectively, Python has been suggested as an alternative for a general programming language to be adopted by the scientific community and it has been used by researchers in areas not related to technology or engineering, such as psychology and astronomy [36,37].

3.2. Architecture

The ARPS framework is composed of four main components: agent manager, agents, discovery service/yellow pages service and Discrete Event Simulator (DES). The architecture and relationship of the first three are illustrated in Figure 3. The agent manager has three main aspects: the management of the availability of resources and policies related to an environment, agent life-cycle, and simulation when it is running for this purpose. It acts as a container for the agents. This container groups resources logically by some criteria, like resource similarity, or accessibility. Agent managers can be distributed across systems. The agent is an entity driven by policies that will manage one or more resources. It can interact with all other agents available by the discovery service. Since each agent manager can be deployed in distributed manner, the discovery service is a directory where its agents are registered when created and their location is made available to be discovered by other requesters, so the agent from one agent manager can exchange messages with agents from others agent managers. Lastly, the DES is a component available for the evaluation of the agents, and its integration with the others will be explained in the following sections.

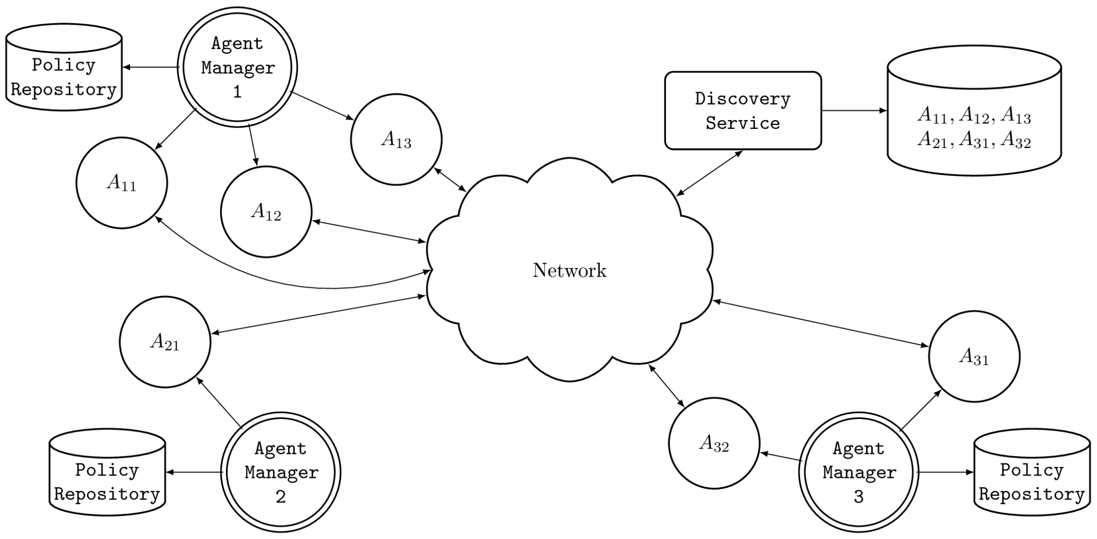

Figure 3. Architecture of the ARPS framework.

3.2.1. Interoperability

The agent manager, agents, and discovery service implement the RESTful architecture style for interoperability. This is done using HTTP methods, and the payload is in JSON format. This RESTful API employs uniform resource access using the format http://hostname:port/resource/, where the *hostname:port* are the host name and listening port to access the API respectively, and the *resource* is a web resource. The HTTP methods (GET, POST, PUT, and DELETE) are available for the clients. The payload of the POST and PUT methods, as well as the possible HTTP response codes, were omitted for simplification purposes. Their descriptions are available at https://gitlab.com/arps/arps/wikis/home. Below we present the API to perform the requisitions.

The agent manager API is intended to fulfill requests by the user. The summary in Table 1 shows how the policies that can be used to drive agents' behaviour can be listed, the agents that are currently running in its environment, the touchpoints that agents can monitor/control, and, when there are agents created just for the purpose of monitoring resources, a timestamped log containing the state of the resources can be retrieved.

The life-cycle of the agents is controlled by creation, modification, inspection, and termination methods, as seen in Table 2.

Table 1. REST API for environment.

HTTP Method	URI	Description
GET	`/list_agents`	show agents in the environment
GET	`/list_policies`	show policies available for agent creation/modification
GET	`/list_touchpoints`	show sensors and actuators available in the system for monitoring
GET	`/monitor_logs`	retrieve the state of the sensor and actuators monitored by the agents created to this end

Table 2. Agent Manager API for agent life-cycle.

HTTP Method	URI	Description
POST	`/agents`	Creates an agent; policies and the fixed interval is specified in the JSON body of the message
PUT	`/agents/[agent_id]`	Modify the agent's policies or relationship with other agents
GET	`/agents/[agent_id]?[params=values]`	Retrieve the internal state of the agent; parameters are used to specify the type of the content returned
DELETE	`/agents/[agent_id]`	Terminate the agent

Lastly, the REST API is available only when the agent manager is created to run the simulator is seen in Table 3.

Table 3. Agent Manager API in simulator mode.

HTTP Method	URI	Description
GET	`/sim/run`	Run simulation
GET	`/sim/stop`	Stop simulation
GET	`/sim/status`	Retrieve current status
GET	`/sim/result`	Retrieve result of the simulation in CSV format
GET	`/sim/save`	Retrieve the organization of the system

The agents have their API described in Table 4. It is not only accessed with the intent of message exchange among the agents, but it can also be called by users or other applications.

Table 4. Agent API.

HTTP Method	URI	Description
PUT	`/policy`	Add or remove a policy accordingly to the content of the message body
PUT	`/meta_agent`	Establishes or removes the relationship between two agents
PUT	`/action`	Requests a user defined action to be executed by the agent accordingly to the content of the message body
GET	`/info`	Request information regarding the agent, such as current policies, touchpoints, and relationships
GET	`/sensors`	Request current state of the sensors available to the agent
GET	`/actuators`	Request current state of the actuators available to the agent

The API provided by the discovery service, shown in Table 5, is used to register, unregister, and list the running agents in the system.

Table 5. Discovery Service.

HTTP Method	URI	Description
GET	`/agents`	List all agents
PUT	`/agents/[agent_id]`	Register the agent
GET	`/agents/[agent_id]`	Retrieve the location of the agent: hostname and port
DELETE	`/agents/[agent_id]`	Unregister the agent

We are aware of the existence of FIPA-HTTP to make compliant FIPA agents using HTTP [38]. This standard, however, only supports the POST method with the semantics of the message embedded in a multi-part content sent in the body of the message. This defeats the purpose of the semantics presented by the HTTP methods, and the already available HTTP status codes related to each HTTP method. Some studies that support this view of interoperability in MAS, using Resource Oriented Architecture (ROA), can be found in [39–41],

There are many advantages of using RESTful architecture style for interoperability. The system is open, so applications that the system designers did not take into account a priori can be integrated into the existing system since the web resources are accessed uniformly. Secure communication can be implemented over HTTPS. A cache can be used to save bandwidth and to optimize system communication. Visualization through different devices can be easily implemented by third-party entities across different devices. The support of the MAS for the REST API can also be used to extend the API and make it compliant with FIPA-HTTP if required.

3.2.2. Agent Model

The intelligent agent model, as described by Russel & Norvig [42], is used to define the agent. Thus, besides the conventional definition of the agent, as an entity perceiving the environment through sensors and modifying it through actuators, it has also the components to achieve reasoning. To this end, the agent's behaviour is driven by a set of policies that are executed by its control loop. A policy can be activated either by interaction with other agents, or internally by a fixed interval. Each policy has access to the touchpoints available in the environment provided by the agent during its creation. Another characteristic is that policies can be dynamically added or removed.

The policies can be reactive or proactive. The reactive policy follows the Event Condition Action (ECA) rule. This model is used to create simple reflex agents. Thus, the agent monitors the environment, and, when the current state matches a condition, a predefined action is performed. Alternatively, the agents can optimize the environment continuously. In this case, their behaviour can be defined in terms of proactive policies. To this end, the user can extend the interface to implement goal-based or utility-based policies.

Another characteristic is related to how agents are organized. The framework defines that an agent has a unidirectional relationship with other agents. This relationship is also dynamic and can be created or removed during the agent lifetime. Since this model does not impose any form of organization and agents communicate using a peer-to-peer architecture, they can be organized hierarchically, or horizontally. Agents can have a relationship established with any other agent available through the yellow pages service. Currently, there is no support to enforce groups or roles.

3.2.3. Simulation

During the simulation, only a single instance of an agent manager is necessary since it will act as the gateway to all other virtual agent managers that would be available in the system, as seen in Figure 4a. Similarly to the actual environment, the format used by the API is http://hostname:port/agent_manager/resource/, where *agent_manager* antecedes *resource* to identify the virtual agent manager that will perform the requisition.

In the actual environment, the transport system used to exchange messages between agents is implemented using the HTTP protocol, as seen previously. It relies on the physical network to function. During the simulation, however, this protocol is substituted by a global bus that will serve as a discovery service and communication layer at the same time. The content sent from one agent is put directly into the queue of messages of the receiver agent. The difference between real and simulated communication is seen in Figure 4b. This has the advantage of removing the overhead of communication. Additionally, it gives the freedom to the user to apply models into this layer to improve the reliability of the agents to evaluate them under intermittent communication, increased latency, or corrupted messages.

The resource is another component that has to be overridden by the models that describe how they behave in the real environment. For example, in our case, we can model the behaviour of the computational resources to have a certain load accordingly to the task that arrived in the system. As well, our model supports resources that affect other resources indirectly. Revisiting the previous example, we can create a model that increases the energy consumption based on the load in the system when the computational resources are utilized.

The main component of the DES is the events generator. It supports both deterministic and stochastic models. The former uses a log file containing the events—when the last event is completed the simulation is terminated—while the latter uses a stochastic generator implemented by the user—the termination needs to be invoked explicitly. Each event has two actions. The main action that is executed every step while the event is still unfinished, and a post-action, when the event exits the system. Figure 4c showing the resource being modified by an event during the simulation and in Figure 4d showing the queue of deterministic events illustrates the DES component working along with the agents

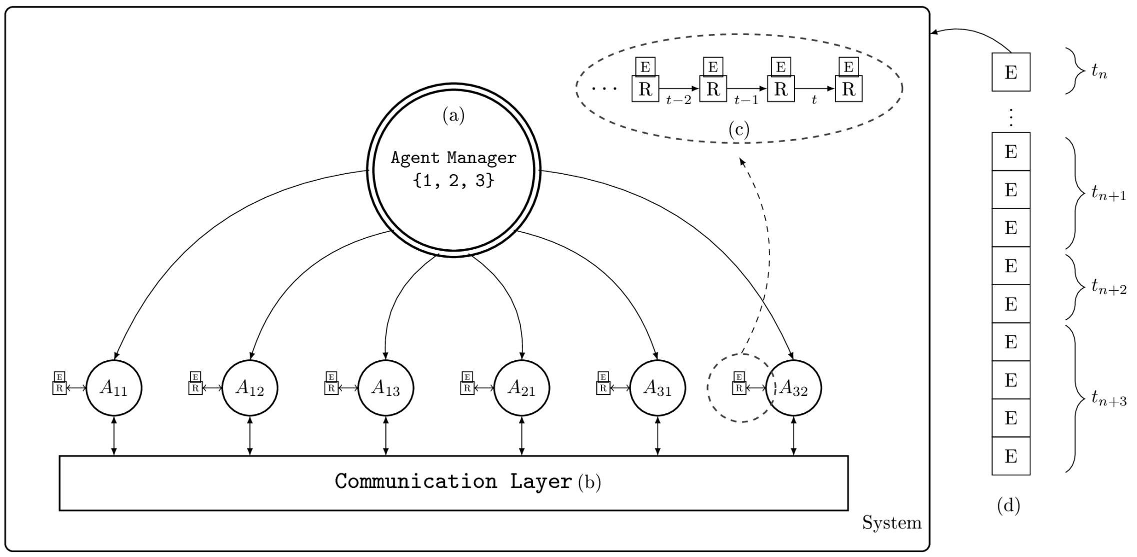

Figure 4. Agent Manager in Simulator Mode.

When finished, the results of the simulation are made available in the CSV format. We have chosen this format since it is supported by a myriad of statistical tools, such as Pandas (https://pandas.pydata.org/), R (https://www.r-project.org/), or spreadsheet-like apps such as LibreOffice (https://www.libreoffice.org/). We believe these third-party tools are better equipped to fulfill the needs of the user.

4. Demonstration of the Framework

In this section, we will illustrate how the ARPS framework can be used to solve the problem of resource management. In the next section, we describe the components needed to be implement to enable the MAS. Following this, we present an example in a specific domain: management of resources in data centres.

4.1. ARPS Framework Usage

The modelling of the MAS for resource management can be accomplished by following the steps:

1. Collect data about the resources in the environment using the available sensors.
2. Create a model of the environment, construct the resources behaviour, their relationship with each other.
3. Create the actuators and the policies to drive the agents based on the available resources.
4. Create a model of the events that will modify the environment.
5. Run the simulation, take a snapshot of the environment containing the agents deployed along with their policies, and collect the results in the CSV format.
6. Use more suitable external tools to evaluate the results.
7. Use a snapshot to deploy the agent managers and their agents in the real environment.
8. Modify the environment model, the policies, or the events and re-run step 4 to improve the agents in the actual environment.

The usage of the framework can be summarized as the implementation of some interfaces, and the creation of the configuration files needed by the agent managers and agents.

We have created the abstraction of the resources that provides the touchpoints to be accessed by sensors and actuators, seen in Figure 5.

Starting with the sensors and a minimal set of the configuration files (files that will be described in detail later in this section), it is possible to execute the agents in the actual environment to collect data since a set of monitoring policies related to the implemented resources are made available automatically. Therefore, it is possible to initially create agents with the only the purpose of gathering data periodically. This data provides insight about the actual resources . This will be essential during the modelling of the environment used by the simulator. This model can be later updated when comparing the simulated environment with the actual environment.

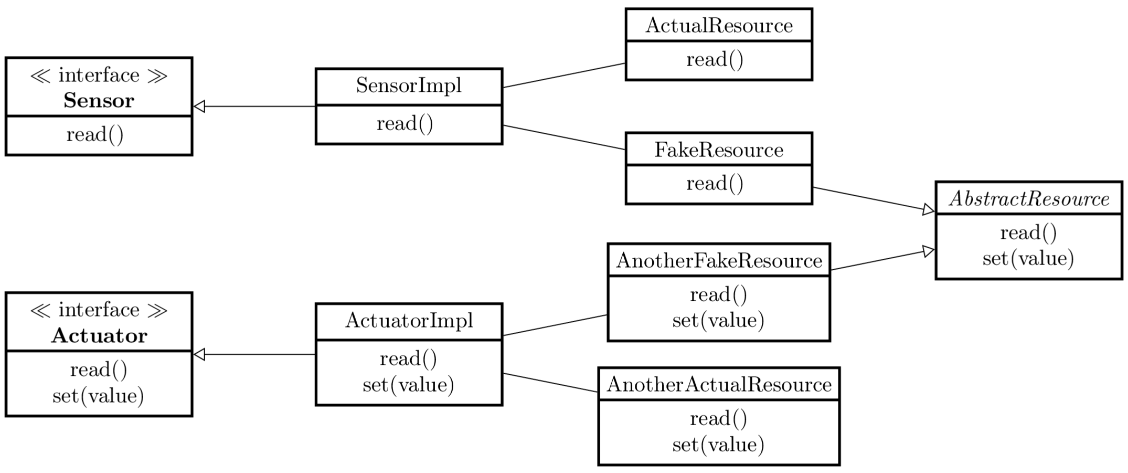

Figure 5. Resources, Sensors, and Actuators structure diagrams.

Once the resources and sensors are implemented, the next step is the implementation of the actuators. The actuators are used by the policies that drive the behaviour of the agents. These policies can be created by implementing the Policy interface, illustrated by a reflex policy in Figure 6. Since our approach is based on autonomic computing concepts [43], other types of policies, employing utility functions, or goal-based approaches can be implemented. The method *condition(event)* can be implemented to use optimization algorithms, defined by the user, that will compute the ideal parameters and then use the results to modify the resources through their actuators. Goal-based approaches would require the user to model the states and use the method *condition(event)* to search for the best action to perform to achieve a better state. Both policies would be executed as periodic policies, so the event is just a time event indicating that it is time for the policy to evaluate the current state and executes its actions based on the perceived environment.

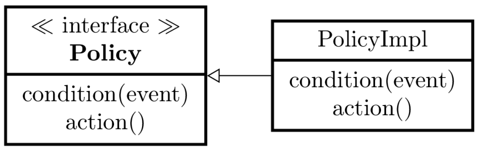

Figure 6. Policy structure diagram.

Since the policies encapsulate the perception and modification of the resources, and these resources can be modified by external events, the next step is the modelling of the these events. The events arrival in the system can be stochastic or deterministic and it can be created by implementing the interface EventQueueLoader. The events are encapsulated in the SimEvent class, which in turn contains the SimTask that will be executed during the simulation. The SimTask *main* method should provide the behaviour that will modify a resource, while the *pos* method will contain all the finalization process to be executed, like resource release, after the task is finished. The relationship between the DES components is illustrated in Figure 7.

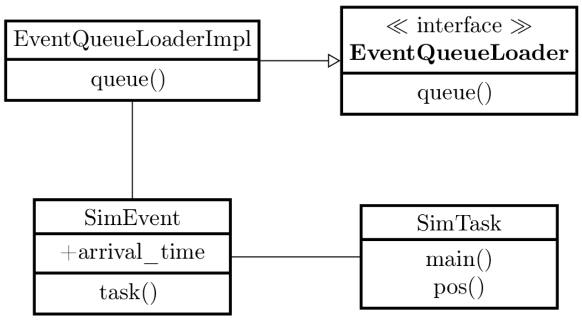

Figure 7. Simulator structure diagram.

Given the structure present in Figure 8, the final step before running the MAS in the actual or simulated environment is the creation of the configuration files, in JSON format, containing the implemented files previously described in this section. Both configuration files are similar in their structure. The file *simulation.conf* contains the paths of the files related to the fake resources, and the DES component classes, while the file *real.conf* contains only the actual resources. The remaining classes, like policies, sensors, and actuators, remain the same for both configuration files. Therefore, the transition of the agents from the simulated environment to the real environment can be done seamlessly.

After the instantiation of the agent manager in simulation mode, using the *simulation.conf* configuration file, the REST API presented in the previous section can be used to create the agents, organize them, and run, stop, and collect the result of the simulation.

Based on the result, only the policies classes need to be modified. Then, using the *real.conf* configuration file, the agents can be deployed in the actual environment. According to the behaviour observed in the actual environment, improvements can be made in models used by the simulated environment. Thus, it is only necessary to modify existing fake resources, simulation tasks and policies. Incrementally, new resources, touchpoints, and policies can be added into the MAS to improve the system.

```
src
├ resources
│  ├ ActualResource
│  ├ FakeResource
│  ├ AnotherActualResource
│  └ AnotherFakeResource
├ touchpoints
│  ├ SensorImpl
│  └ ActuatorImpl
├ policies
│  └ PolicyImpl
├ des
│  ├ EventQueueLoaderImpl
│  ├ SimEvent
│  └ SimTask
└ conf
   ├ simulation.conf
   └ real.conf
```

```
simulator.conf:
{
    ...
    'resources': ['resources/FakeResource',
                  'resources/AnotherFakeResource'],
    'touchpoints': ['touchpoints/SensorImpl',
                    'touchpoints/ActuatorImpl'],
    'policies': ['policies/PolicyImpl'],
    'des': ['des/EventQueueLoaderImpl',
            'des/SimTask', 'des/SimEvent]
    ...
}
-----------------------------
real.conf:
{
    ...
    'resources': ['resources/ActualResource',
                  'resources/AnotherActualResource'],
    'touchpoints': ['touchpoints/SensorImpl',
                    'touchpoints/ActuatorImpl'],
    'policies': ['policies/PolicyImpl'],
    ...
}
```

Figure 8. The minimal structure required.

4.2. Example: Data Centre Management

As previously discussed in Section 3, we are researching ways to address the problems in trying to optimize energy efficiency in the data centre while ensuring that other performance metrics are met using MAS. To illustrate this, we provide a minimal example to cover the basic setup of the ARPS framework. The source code is available at https://gitlab.com/arps/arps/tree/main/arps/examples/computational_resources_management.

This example by no means provides an in-depth analysis of resource management in data centres and only serves to illustrate how the MAS can be realized. To keep this example simple, we will manage only two resources from a single host: the CPU and the Energy Monitor. Additional resources, such as temperature sensors of the environment, cooling system, and multiple hosts, can be added iteratively as the need for better understanding the system arises.

The CPU provides means to both read its current utilization or modify its frequency. It is driven by two governors: performance and powersave. This enables dynamic frequency scaling. The maximum frequency is 2.9 GHz in each one of its four cores and when in performance mode, the frequency is adjusted dynamically. Conversely, in powersave mode, the frequency stays closer to the minimum frequency available. The Energy Monitor only provides an interface to read the estimated power consumption in watts using the tool *PowerTOP* [44]. This tool was developed by Intel, and provides estimates on power consumption.

To understand how energy consumption is affected when the CPU has a certain workload, we collected the data using the sensors. As described in the previous section, when a resource is implemented, a monitor policy related to it is made available by the Agent Manager. In this case, it is possible to create agents executing the CPUMonitorPolicy and EnergyMonitorPolicy policies periodically. The collected data were summarized into two charts, where the Y-axis represents the normalized value of the CPU workload and the estimated power. The X-axis is the time in seconds. In Figure 9, the CPU is using a performance governor, while in Figure 10 the CPU is using powersave governor. As expected, there is a correlation between workload and energy consumption in performance mode since the CPU adjusts its frequency during the execution. The same does not apply in the powersave mode, where the energy consumption almost never goes over a certain value.

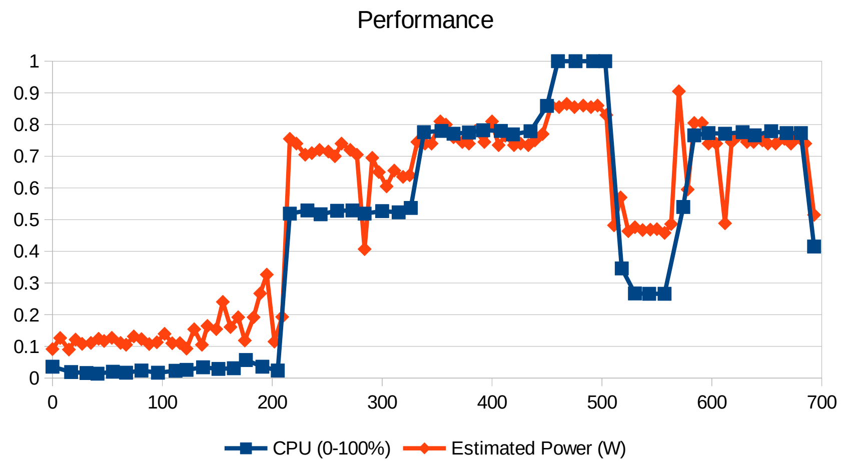

Figure 9. Correlation of energy consumption and CPU workload using performance governor.

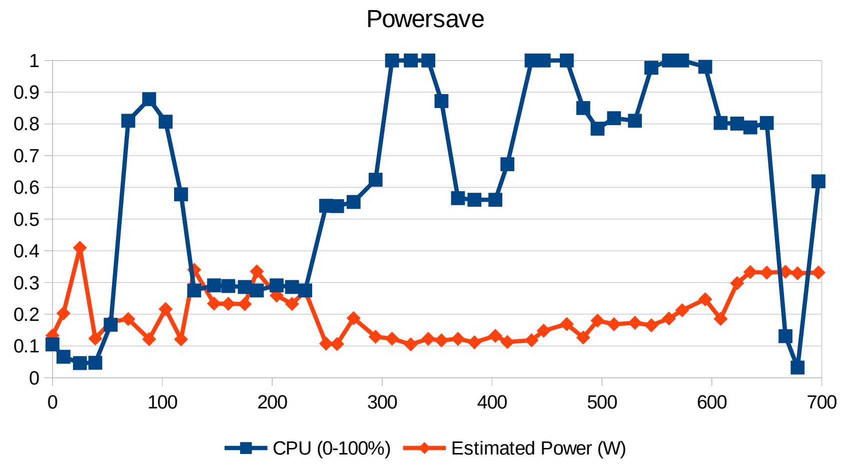

Figure 10. Correlation of energy consumption and CPU workload using powersave governor.

To reduce energy consumption, for example, we can use reactive policies implementing the method *condition(event)* of the Policy interface to evaluate the workload in the CPU. The action performed by the policy can be the adjustment of the CPU's frequency to a lower level. In this example, we considered two different strategies for saving energy. The strategies involve three policies; these policies are presented in Figure 11. The first adjusts the CPU to powersave mode when the utilization is over 80% while in the second sets the CPU to powersave mode when the utilization is over 50%. The third returns the CPU to performance mode when the utilization is under 50. The first strategy is composed of policies PowersaveDynamicScalingPolicy1 and PerformanceDynamicScalingPolicy; strategy two uses PowersaveDynamicScalingPolicy2.

We have chosen batch jobs to represent the external tasks that modify the environment to evaluate how the agents will manage energy consumption. In this case, we opted to use the deterministic model of the job arrival using an external file of events, where each row represents one event arriving in the system containing the arrival time, duration of the task, and workload. The SimTask *main* method, seen in Figure 12, allocates the CPU resource, and consequently causes additional power consumption proportionately to the workload when the CPU is performance mode and more steady when the CPU is in powersave mode, as mentioned previously. The SimTask *pos* method would release the resource as soon as the task is completed.

```
class PowersaveDynamicScalingPolicy1(Policy):
    def condition(self, event):
        return self.cpu.read() > 80

    def action(self):
        self.cpu_freq.set(governor='powersave', max_frequency=0.9)

class PowersaveDynamicScalingPolicy2(Policy):
    def condition(self, event):
        return self.cpu.read() > 50

    def action(self):
        self.cpu_freq.set(governor='powersave', max_frequency=0.9)

class PerformanceDynamicScalingPolicy(Policy):
    def condition(self, event):
        return self.cpu.read() <= 50

    def action():
        self.cpu_freq.set(governor='performance', max_frequency=2.9)
```

Figure 11. Example of implementation of a reactive policy.

```
class CPUTask(SimTask):
    def main(self):
        if not self.acquired:
            self.acquired = True
            self.cpu.value += self.wordload

        if self.cpu.frequency == 'powersave':
            estimate = self.powersave_estimate(self.cpu.value)
        elif self.cpu.frequency == 'performance':
            estimate = self.performance_estimate(self.cpu.value)
        self.energy_monitor.value = estimate

    def pos(self):
        self.cpu.value -= self.wordload
```

Figure 12. Example implementation of a task to update the energy monitor based on CPU workload.

The first analysis suggests that the average energy consumption with the first strategy is 9 Watts while for the second one it is 7.57 Watts. Without taking any other resources correlated to energy consumption and CPU, we can infer that the second one is better. However, other resources or metrics, such as a metric on Quality of Service (QoS), can be affected by the CPU dynamic frequency scaling of strategy two; these could be included in the model for further investigation.

When comparing the policies in the real environment, a similar behaviour is observed. The average energy consumption in the first policy is 13.75 Watts while in the second one is 12.34 Watts. These results come from a very limited set of experiments and comparisons and so no significance can be determined. Again, this was meant as example to illustrate how the ARPS framework can be used.

5. Discussion

We presented ARPS, a framework to enable evaluation of a MAS before deployment in an actual environment. This framework is the result of a general purpose MAS solution developed to be as flexible as possible to provide management of a data centre. It has proven useful in a specific domain, and we believe that others in the MAS community can benefit from this additional option when developing solutions to their problems. To this end, we illustrated how it could be applied using our specific case.

Due to the decision of using the RESTful architecture style, agents can be deployed manually by a user in an environment that would contain a single agent, without the need of an agent manager.

This unintended feature is possible because agents are self-contained and all the operations are made available through the API. The interaction can be done using web clients widely available on the Internet. The only requirement is that the platform can run Python applications. Thus, a user can instantiate an agent to control an autonomous robot developed in a platform such as a RaspberryPI, or it can deploy an agent in devices of an IoT environment.

We have identified other features that could enrich this framework. The inclusion of a Policy Management Tool can ease the process of creating new policies dynamically using a high-level language. Also, new resources and their touchpoints could be made available on the fly or dynamically removing unavailable resources. This could improve the integration of simulation and deployment even more. We plan to add a learning component to make agents more robust to complex adaptive environments. Further, reliability characteristics, such as fault tolerance, and security concerns, are being addressed and will be included in future versions.

Currently, the framework has some limitations related to features that were set aside due to low priorities. The framework does not come with a visualization component. Thus, it is not possible to visualize the running agents both during the simulation or within the real environment. Also, because we were aiming to have actual agents, we did not evaluate the usage of this framework applied to problems entailing only agent-based modelling simulation (ABMS). Since ABMS and MAS concepts overlaps, this MAS framework could be adapted to be used as ABMS tool. We do not know, however, how it performs when millions of agents are deployed in the environment. The framework does include statistical tools to analyze the results of the simulation or to assess the system in the actual environment since we believe that each user has their own needs and preference for tools. As mentioned, all the data gathered via the framework is made available in CSV format and so other applications that can provide more suitable tools can be used to extract insightful information from the data. Lastly, the framework is not FIPA compliant yet.

Author Contributions: Conceptualization, T.C.P. and M.B.; Methodology, T.C.P. and M.B.; Software, T.C.P.; Supervision, M.B.; Visualization, T.C.P.; Writing—original draft, T.C.P. and M.B.; Writing—review & editing, T.C.P. and M.B.

Funding: This research is partially supported by CNPq, Brazil grant number 249237/2013-0.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Wangermann, J.P.; Stengel, R.F. Optimization and coordination of multiagent systems using principled negotiation. *J. Guid. Control Dyn.* **1999**, *22*, 43–50. [CrossRef]
2. Roche, R.; Blunier, B.; Miraoui, A.; Hilaire, V.; Koukam, A. Multi-agent systems for grid energy management: A short review. In Proceedings of the 36th Annual Conference on IEEE Industrial Electronics Society (IECON 2010), Glendale, AZ, USA, 7–10 November 2010. doi:10.1109/iecon.2010.5675295. [CrossRef]
3. Hadeli; Valckenaers, P.; Kollingbaum, M.; Brussel, H.V. Multi-Agent Coordination and Control Using Stigmergy. *Comput. Ind.* **2004**, *53*, 75–96. doi:10.1016/s0166-3615(03)00123-4. [CrossRef]
4. Sislak, D.; Rehak, M.; Pechoucek, M. A-globe: Multi-Agent Platform with Advanced Simulation and Visualization Support. In Proceedings of the 2005 IEEE/WIC/ACM International Conference on Web Intelligence (WI'05), Compiegne, France, 19–22 September 2005. doi:10.1109/wi.2005.18. [CrossRef]
5. Pěchouček, M.; Mařík, V. Industrial Deployment of Multi-Agent Technologies: Review and Selected Case Studies. *Auton. Agents Multi-Agent Syst.* **2008**, *17*, 397–431. doi:10.1007/s10458-008-9050-0. [CrossRef]
6. Open Source Initiative. The MIT License. 1988. Available online: https://opensource.org/licenses/MIT (accessed on 20 November 2016).
7. Kravari, K.; Bassiliades, N. A Survey of Agent Platforms. *J. Artif. Soc. Soc. Simul.* **2015**, *18*, 1–11. doi:10.18564/jasss.2661. [CrossRef]

8. Bellifemine, F.; Bergenti, F.; Caire, G.; Poggi, A. Jade—A Java Agent Development Framework. In *Multi-Agent Programming*; Springer: Boston, MA, USA, 2005; pp. 125–147. doi:10.1007/0-387-26350-0_5. [CrossRef]
9. Poslad, S. Specifying Protocols for Multi-Agent Systems Interaction. *ACM Trans. Auton. Adapt. Syst.* **2007**, *2*, 15. doi:10.1145/1293731.1293735. [CrossRef]
10. Šišlák, D.; Rehák, M.; Pěchouček, M.; Rollo, M.; Pavlíček, D. A-globe: Agent Development Platform with Inaccessibility and Mobility Support. In *Software Agent-Based Applications, Platforms and Development Kits*; Springer: Birkhäuser Basel, 2005; pp. 21–46. doi:10.1007/3-7643-7348-2_2. [CrossRef]
11. Helsinger, A.; Thome, M.; Wright, T. Cougaar: A scalable, distributed multi-agent architecture. In Proceedings of the 2004 IEEE International Conference on Systems, Man and Cybernetics (IEEE Cat. No.04CH37583), The Hague, The Netherlands, 10–13 October 2004. doi:10.1109/icsmc.2004.1399959. [CrossRef]
12. Pokahr, A.; Braubach, L.; Lamersdorf, W. Jadex: A BDI Reasoning Engine. In *Multi-Agent Programming*; Springer: Boston, MA, USA, 2005; pp. 149–174. doi:10.1007/0-387-26350-0_6. [CrossRef]
13. Rao, A.S.; Georgeff, M.P.; BDI Agents: From Theory to Practice. In Proceedings of the ICMAS: International Conference on Multi-Agent Systems, San Francisco, CA, USA, 1995, Volume 95, pp. 312–319.
14. Boissier, O.; Bordini, R.H.; Hübner, J.F.; Ricci, A.; Santi, A. Multi-Agent Oriented Programming with Jacamo. *Sci. Comput. Program.* **2013**, *78*, 747–761. doi:10.1016/j.scico.2011.10.004. [CrossRef]
15. Rao, A.S. AgentSpeak(L): BDI agents speak out in a logical computable language. In *Lecture Notes in Computer Science*; Springer: Berlin/Heidelberg, Germany, 1996; pp. 42–55. doi:10.1007/bfb0031845. [CrossRef]
16. Bordini, R.H.; Hübner, J.F.; Wooldridge, M. *Programming multi-agent systems in AgentSpeak using Jason*; John Wiley & Sons: Chichester, WS, UK, 2007; Volume 8.
17. Hübner, J.F.; Boissier, O.; Kitio, R.; Ricci, A. Instrumenting Multi-Agent Organisations with Organisational Artifacts and Agents. *Auton. Agents Multi-Agent Syst.* **2009**, *20*, 369–400. doi:10.1007/s10458-009-9084-y. [CrossRef]
18. Ricci, A.; Piunti, M.; Viroli, M.; Omicini, A. Environment Programming in CArtAgO. In *Multi-Agent Programming*; Springer: Boston, MA, USA, 2009; pp. 259–288. doi:10.1007/978-0-387-89299-3_8. [CrossRef]
19. Gutknecht, O.; Ferber, J. MadKit. In Proceedings of the Fourth International Conference on Autonomous Agents—AGENTS ’00, Barcelona, Spain, 3–7 June 2000; pp. 78–79. doi:10.1145/336595.337048. [CrossRef]
20. Allan, R.J. *Survey of Agent Based Modelling and Simulation Tools*; Technical Report, DL-TR-2010-007; Computational Science and Engineering Department, STFC Daresbury Laboratory: Daresbury, Warrington, England, 2010.
21. Railsback, S.F.; Lytinen, S.L.; Jackson, S.K. Agent-Based Simulation Platforms: Review and Development Recommendations. *Simulation* **2006**, *82*, 609–623. doi:10.1177/0037549706073695. [CrossRef]
22. Minar, N.; Burkhart, R.; Langton, C.; Askenazi, M. *The Swarm Simulation System: A Toolkit for Building Multi-Agent Simulations*; Working Papers 96-06-042; Santa Fe Institute: Santa Fe, NM, USA, 1996.
23. Sklar, E. Netlogo, a Multi-Agent Simulation Environment. *Artif. Life* **2007**, *13*, 303–311. doi:10.1162/artl.2007.13.3.303. [CrossRef] [PubMed]
24. Luke, S.; Cioffi-Revilla, C.; Panait, L.; Sullivan, K.; Balan, G. Mason: A Multiagent Simulation Environment. *Simulation* **2005**, *81*, 517–527. doi:10.1177/0037549705058073. [CrossRef]
25. North, M.J.; Collier, N.T.; Ozik, J.; Tatara, E.R.; Macal, C.M.; Bragen, M.; Sydelko, P. Complex Adaptive Systems Modeling With Repast Simphony. *Complex Adapt. Syst. Modeling* **2013**, *1*, 3. doi:10.1186/2194-3206-1-3. [CrossRef]
26. Collier, N.; North, M. Parallel Agent-Based Simulation with Repast for High Performance Computing. *Simulation* **2012**, *89*, 1215–1235. doi:10.1177/0037549712462620. [CrossRef]
27. Niazi, M.; Hussain, A. Agent-Based Computing from Multi-Agent Systems to Agent-Based Models: A Visual Survey. *Scientometrics* **2011**, *89*, 479–499. doi:10.1007/s11192-011-0468-9. [CrossRef]
28. Drogoul, A.; Vanbergue, D.; Meurisse, T. Multi-agent Based Simulation: Where Are the Agents? In *Multi-Agent-Based Simulation II*; Springer: Berlin/Heidelberg, Germany, 2003; pp. 1–15. doi:10.1007/3-540-36483-8_1. [CrossRef]
29. Bosse, S.; Engel, U. Augmented Virtual Reality: Combining Crowd Sensing and Social Data Mining with Large-Scale Simulation Using Mobile Agents for Future Smart Cities. In Proceedings of the 5th International Electronic Conference on Sensors and Applications, Canary Islands, Tenerife, 25–27 September 2019; Volume 4, p. 49.

30. Azimi, S.; Delavar, M.; Rajabifard, A. Multi-agent simulation of allocating and routing ambulances under condition of street blockage after natural disaster. *Int. Arch. Photogramm. Remote Sens. Spat. Inf. Sci.* **2017**, *42*, 325–332. [CrossRef]
31. Kephart, J.; Chess, D. The Vision of Autonomic Computing. *Computer* **2003**, *36*, 41–50. doi:10.1109/mc.2003.1160055. [CrossRef]
32. Tesauro, G.; Chess, D.M.; Walsh, W.E.; Das, R.; Segal, A.; Whalley, I.; Kephart, J.O.; White, S.R. A multi-agent systems approach to autonomic computing. In Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems, New York, NY, USA, 19–23 July 2004; Volume 1, pp. 464–471.
33. Norouzi, F.; Bauer, M. Autonomic Management for Energy Efficient Data Centers. In Proceedings of the Sixth International Conference on Cloud Computing, GRIDs, and Virtualization (Cloud Computing 2015), Nice, France, 22 March 2015; pp. 138–146.
34. Schraven, M.; Guarnieri, C.; Baranski, M.; Müller, D.; Monti, A. Designing a Development Board for Research on IoT Applications in Building Automation Systems. In Proceedings of the International Symposium on Automation and Robotics in Construction (ISARC), Banff, AB, Canada, 2019; Volume 36, pp. 82–90. doi:10.22260/ISARC2019/0012. [CrossRef]
35. Andročec, D.; Tomaš, B.; Kišasondi, T. Interoperability and lightweight security for simple IoT devices. In Proceedings of the 2017 40th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO), Opatija, Croatia, 22–26 May 2017; pp. 1285–1291. doi:10.23919/MIPRO.2017.7973621. [CrossRef]
36. Perkel, J.M. Programming: Pick Up Python. *Nature* **2015**, *518*, 125–126. doi:10.1038/518125a. [CrossRef] [PubMed]
37. Ayer, V.M.; Miguez, S.; Toby, B.H. Why Scientists Should Learn To Program in Python. *Powder Diffr.* **2014**, *29*, S48–S64. doi:10.1017/s0885715614000931. [CrossRef]
38. Foundation for Intelligent Physical Agents (FIPA). FIPA Agent Message Transport Protocol for HTTP Specification. 2002. Available online: http://www.fipa.org/specs/fipa00084/SC00084F.html (accessed on 10 June 2019).
39. Ciortea, A.; Boissier, O.; Zimmermann, A.; Florea, A.M. Give Agents Some REST: A Resource-oriented Abstraction Layer for Internet-scale Agent Environments. In Proceedings of the 16th Conference on Autonomous Agents and MultiAgent Systems (AAMAS '17), São Paulo, Brazil, 8–12 May 2017; International Foundation for Autonomous Agents and Multiagent Systems: Richland, SC, USA, 2017; pp. 1502–1504.
40. Braubach, L.; Pokahr, A. Conceptual Integration of Agents with WSDL and RESTful Web Services. In *Lecture Notes in Computer Science*; Springer: Berlin/Heidelberg, Germany, 2013; pp. 17–34. doi:10.1007/978-3-642-38700-5_2. [CrossRef]
41. Pablo, P.V.; Holgado-Terriza, J.A. *Integration of MultiAgent Systems with Resource-Oriented Architecture for Management of IoT-Objects*; IOS Press: Amsterdam, The Netherlands; Volume 23: Intelligent Environments 2018; pp. 567–576. [CrossRef]
42. Russell, S.J.; Norvig, P. *Artificial Intelligence: A Modern Approach*; Pearson Education: Upper Saddle River, NJ, USA, 2009; Chapter 2.
43. Kephart, J.O.; Walsh, W.E. An artificial intelligence perspective on autonomic computing policies. In Proceedings of the Fifth IEEE International Workshop on Policies for Distributed Systems and Networks; Yorktown Heights, NY, USA, 9 June 2004. doi:10.1109/POLICY.2004.1309145. [CrossRef]
44. Intel Open Source. PowerTOP: Linux Tool to Diagnose Issues with Power Consumption and Power Management. 2007. Available online: https://01.org/powertop/ (accessed on 12 July 2019).