A Spin-off Version of Jason for IoT and Embedded Multiagent Systems

Carlos Eduardo Pantoja[1,2][0000−0002−7099−4974],
Vinicius Souza de Jesus[1][0000−0002−4534−6078],
Nilson Mori Lazarin[1,2][0000−0002−4240−3997], and
José Viterbo[1][0000−0002−0339−6624]

[1] Institute of Computing – Fluminense Federal University (UFF),
Niterói - RJ, Brazil
[2] Federal Center for Technological Education Celso Suckow da Fonseca (Cefet/RJ),
Rio de Janeiro - RJ, Brazil
pantoja@cefet-rj.br, {vsjesus,nlazarin}@id.uff.br, viterbo@ic.uff.br

Abstract. Embedded artificial intelligence in IoT devices is presented as an option to reduce connectivity dependence, allowing decision-making directly at the edge computing layer. The Multi-agent Systems (MAS) embedded into IoT devices enables, in addition to the ability to perceive and act in the environment, new characteristics like pro-activity, deliberation, and collaboration capabilities to these devices. A few new frameworks and extensions enable the construction of agent-based IoT devices. However, no framework allows constructing them with hardware control, adaptability, and fault tolerance, besides agents' communicability and mobility. This work presents an extension of the Jason framework for developing Embedded MAS with BDI agents capable of controlling hardware, communicating, and moving between IoT devices capable of dealing with fault tolerance. A case study of an IoT solution with a smart home, a monitoring center, and an autonomous vehicle is presented to demonstrate the framework's applicability.

Keywords: Internet of Things · Multi-agent Systems · Edge Computing

1 Introduction

The Internet of Things (IoT) promotes the use of devices that sense physical environments in various application domains. However, they generate a considerable amount of raw data stream that needs to be transmitted to be processed [2]. Pervasive Computing allows the development of distributed cognitive devices capable of extracting relevant information and making decisions directly at the edge computing layer, reducing the dependence on connectivity [10].

One of the fields of Distributed Artificial Intelligence (DAI) is Multi-agent Systems (MAS), which are composed of multiple autonomous and proactive entities with decision-making capacity and social abilities that can collaboratively interact to achieve a common goal for the system [28]. In this research area, an

integration of hardware and software that allow agents to sense and act in a real-world environment using sensors and actuators is named Embedded MAS [3]. These systems can also contribute to reduce dependency on connectivity since a cognitive agent embedded in an IoT device can process the raw data received from sensors and act immediately, thus accelerating decision-making [9].

One of the most well-known cognitive agent architecture, the Belief-Desire-Intention (BDI) model [7], is based on the knowledge that an agent can have from the environment (perceptions), other agents, or itself [21]. When applied embedded in devices, it provides decision-making at edge level by using perceptions and beliefs captured from the real world and other devices [1, 5, 9]. We performed a mapping review, where we found some works that present a framework [8, 12, 15, 25] or provide some features to construct IoT agent-based devices using BDI frameworks, such as: extension to provide interoperable between cyber-physical and IoT systems using fuzzy logic [16]; approaches to reconfiguring agent's goals on the fly [11] and developing MAS with IoT objects [6], or architecture to Ambient Intelligence with IoT [23]. However, none of these solutions simultaneously meets all the needs of an agent-based IoT device, such as the hardware control, the communicability with other MAS or devices, the mobility of the agents between different MAS or devices, fault tolerance, and adaptability.

This paper presents the Jason Embedded – a spin-off extended version of the Jason framework [4] – that provides autonomy, pro-activity, social ability, adaptability, and fault-tolerance to IoT devices. It allows programming agents dedicated to exchanging KQML messages [14] between different devices and agents dedicated to controlling sensors and actuators and capable of deciding when to perceive the environment or yet can define a strategy to gather perceptions from sensors. Besides, allow the programming of Open MAS [26], where agents can move from one system to another using an IoT network.

For this, the reasoning cycle of a standard Jason agent was modified to create two novel specific types of BDI agents to program Embedded MAS. In the first, the agent can perceive or act in the environment and monitor the connection status with wired microcontrollers, allowing fault tolerance and adaptability. In the second, agents can exchange messages with other MAS and handle agents' arrival and departure into its MAS. Then, we integrated and adapted the ad-hoc solutions [17, 19, 24, 27] in a single distribution. Despite the modifications, all agents maintain the original features of Jason's agents. Jason Embedded allows the designer to abstract some aspects of hardware interfacing and communication, focusing only on programming the MAS. The contribution of this work is an extended framework to allow hardware controlling, communicability, and agent mobility to be integrated into a single framework. To demonstrate this, we build and present a case study that implemented an IoT solution for home monitoring integrated with a central control and an unmanned autonomous vehicle.

This paper is structured as follows: an analysis of related works is presented in Section 2; in Section 3 we present the Jason Embedded and the newly available behavior of agents; we demonstrate an Embedded MAS in a case study in Section 4; finally, a discussion and future work are presented in Section 5.

2 Related Works

A mapping review was conducted to find works that use the BDI framework for the development of IoT systems based on cognitive agents. We followed these steps: first was to define the search string[3] for Google Scholar which returned 246 results; next, the results were filtered, ignoring duplicity and results that were not a thesis or a paper, remaining 215; the third step considered only works published after 2017, remaining 171 works; the fourth step ignored surveys or reviews, remaining 153 works. After this, 56 works with the words *framework, tool, architecture, library, hardware, IoT, cyber-physical, things, or ubiquitous* in the title were considered for the next step.

Finally, it was rejected the works that were not directly related to IoT and BDI (i.e., simulation, data mining, and others were discarded); finally, 11 works remained. Three works [1, 5, 9] only present an IoT implementation based on cognitive agents. Four works [6, 11, 16, 23] feature an extension to a BDI framework, however, they do not meet all the needs of an agent-based IoT device, such as the hardware control, the communicability with other MAS or devices, the mobility of the agents between different MAS or devices, fault tolerance, and adaptability. Thus, only the below four works [8, 12, 15, 25] present a framework for programming IoT agents.

The first [8] try to provide connectivity to the Web of Things using an agent-oriented visual programming IDE using a framework endowed with a REST endpoint. The IDE is a web-based solution for reducing that expertise in adopting the BDI. The solution allows agents to discover and connect to Things using internal structures, but it does not offer embedded solutions to direct control hardware, being dependent on the IoT infrastructure. Our solution employs specialized agents to control hardware and access IoT gateways in an extended version of Jason. The hardware control is independent of the IoT infrastructure, which is used for the communicability and mobility of agents.

Similarly, the second [12] provides a declarative language in a BDI-like style for programming MAS for the IoT using microcontrollers in Python. However, microcontrollers are limited in processing capacity and memory and need other components to be connected to any IoT Gateway. Withal allows high-level agents to connect and control these microcontrollers if only they are available and connected to the IoT. The PHIDIAS is an improved version of PROFETA, and it presents ways of developing intelligence in devices, including IoT, but does not allow agent mobility between different MAS or Environments. PHIDIA allows communication using HTTP. The communication between microcontrollers and agents is performed using wireless devices. The microcontrollers, sensors, and actuators are not part of the MAS. Besides, one agent is responsible for each edge device. The third [25] try reduce the gap between theory and practical applications in cybersecurity using BDI agents and MAS. In their framework, agents can control IoT devices (remotely or hosted in the device), utilizing API to con-

[3] *("embedded" OR "embodied") AND ("multiagent system" OR "multi-agent system") AND "belief-desire-intention" AND "framework" AND "internet of things"*

nect to IoT gateways or simulated software. In both, the dependence on IoT gateways avoid agents from properly working even if embedded. The proposed in this paper allows agents to directly control hardware using serial communication (without network connection) and access the IoT network to exchange information with other MAS.

The fourth [15] try to bring together academic development and industry by offering ways of developing MAS using recent technologies (e.g., Node.JS). Similarly, our approach aims to provide and facilitate ways of developing embedded and IoT systems to be adopted in domains such as education, academia, and industry. In addition to the mapping review, the SPADE 3 [22], in its most recent version, uses XMPP and has an extension for BDI agents. However, it does not allow agent mobility between different MAS. The authors even argue that mobility between different physical or logical nodes is an interesting and desired functionality, however not contemplated by SPADE.

As previously presented, none of the works embed a MAS in IoT devices with BDI agents capable of communicating and moving between different Embedded MAS and also adapting to some faults that may arise in hardware malfunctioning, for example. Therefore, our spin-off version of the Jason framework presented in this work has customized architectures of BDI agents able to control hardware, communicate, and move between MAS embedded in devices using an IOT network. Besides, the agents that interface hardware can be fault tolerant if the hardware becomes unavailable and adapt themselves to find another way to comply with their goals.

3 Jason Embedded

Jason Embedded[4] is an extended version of Jason [4] that supports the development of Embedded MAS to provide autonomy and communicability to IoT devices and mobility and adaptability to agents. In this paper, we define autonomy as a characteristic an Embedded MAS provides to a device that does not need external architectures to control and manage it. Communicability is the agent's ability to communicate with other agents in its MAS or another MAS. We define mobility as the agents' ability to move from one MAS to another. Finally, Fault Tolerance and Adaptability are defined as the ability of an agent to identify if a physical resource is absent or not answering commands and to overcome this situation by modifying its goals.

Jason Embedded also provides new types of agents to control hardware devices and to communicate with agents hosted in other MAS. Besides, our extended version allows agent mobility between Embedded MAS, where one agent or a group of agents can move from one system to another. Then, it specializes the MAS by creating agents dedicated to certain functionalities without modifying the core of Jason, maintaining all of its original functionalities. Jason Embedded is an initiative employed by our research group in embedded solutions

[4] https://jasonembedded.chon.group/

to facilitate the development and teaching of practical MAS. The overall picture of our framework is depicted in Figure 1.

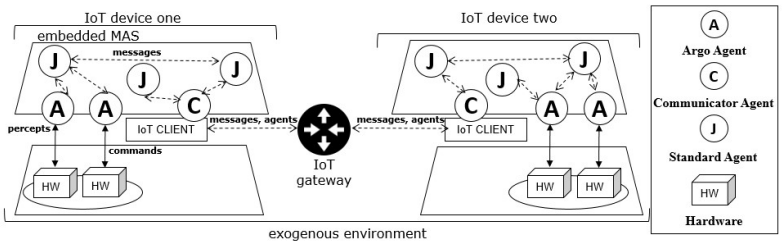

Fig. 1: The overall picture of the framework architecture.

In this approach, the developer can employ Standard, Argo, and Communicator agents to develop the Embedded MAS. Each type of agent has specific abilities to deal with hardware interfacing, communicability, and mobility using an IoT network. Argo agents use hardware interfacing to capture perceptions from the exogenous environment and send commands to activate actuators. The Communicator agent has an IoT client middleware for connecting to an IoT gateway to send messages or agents from one IoT device to another. Standard agents can communicate and exchange information only with agents from their system.

For example, in a scenario concerning three different IoT devices hosting Embedded MAS, one is responsible for managing resources in a house, one is responsible for a monitoring center, and the last is an emergency vehicle. An Argo agent monitors the house's sensors, and if something is detected, it informs the house's Communicator agent to send an alert to the monitoring center. Consequently, the Communicator agent from the monitoring center searches for an available vehicle and sends a Standard agent to help the Argo agent to drive the vehicle to attend the house's emergency. The three devices are distributed, autonomous, and controlled by an Embedded MAS. They can communicate by sending messages using an IoT gateway available, and Argo agents perform all sensing in each device. Finally, agents can move from one system to another.

The new types of agents are based on the standard Jason's agent reasoning cycle. Their reasoning cycle is modified at specific points to create new behaviors without modifying any existing function. In this way, every extended agent is still a Jason agent. The agent's reasoning cycle comprises ten steps that complete a BDI decision loop [4]. In the first step, the agent senses the simulated environment using the Perceive method to gather all available perceptions. At this point, the agent can only perceive a simulated environment if it exists, and if it needs to interface with real sensors, this environment must be properly coded. The last step executes an intention by performing actions from a selected plan. These two steps represent how the agent can perceive and modify the environment, and agents could use them to interface sensors and actuators.

Then, the first step could allow agents to gather perceptions directly from sensors and process them as beliefs. Since this mechanism works as passive perception, the agent is aware of all available information, even if it does not need all of them to accomplish a goal. To overcome this issue, the agent could also decide whether or not to gather available perceptions and filter undesired perceptions at runtime. It allows the agent to indirectly perform active perception since it can decide when and what to perceive, but it always does it in the first step of its reasoning cycle. Similarly, the last step could allow agents to act upon actuators by adopting a protocol that runs over the serial port by sending activation commands [24]. The agent does not need to use Jason's simulated environment — the endogenous representation of the system's environment — since it can directly activate or deactivate actuators in the real world — the exogenous environment. This approach reduces the use of abstractions and layers to reflect actions in the device's hardware where the Embedded MAS is hosted.

In the third step of the reasoning cycle, an agent checks its mail, looking for messages that other agents from its MAS could have sent to it. Since Jason Embedded aims to provide a framework for programming autonomous and embedded solutions using agents, it is worthwhile to think of prototypes communicating with other prototypes. Considering that each prototype has an Embedded MAS, some agents could work as a communicator, receiving and sending messages to other communicator agents. Otherwise, the Embedded MAS will be autonomous and proactive but without social ability. Then, these agents could check an alternative mailbox with messages from other Embedded MAS. In the last step, agents could send messages to agents within and out of its system by addressing other communicators using an IoT middleware, for instance.

As sending a message occurs in the last step, it could also allow specific agents to move agents from one Embedded MAS to another. In Embedded MAS, agent mobility is based on bio-inspired protocols [27]. The protocols simulate natural behaviors that could be explored in collaborative tasks using IoT devices. One or more agents or even the whole MAS could move from one device to another to take control of the destination device (predation) or to use it as a temporary non-hazardous relationship (mutualism and inquilinism) if both parties accept the protocol. For communicability and mobility, agents must connect to an IoT middleware to redirect the messages and agents to an agent of the target Embedded MAS. The description of all new internal actions, their behaviors, and requested parameters can be seen in Table 1.

Below are described the new characteristics allowed by the framework presented in this work. Jason Embedded offers two types of extended agents: one for interfacing hardware named Argo and another for communication named Communicator, responsible for all communications and mobility from outside its MAS. Besides, Jason Embedded maintains the standard agents present in the Jason. It is important to remark that the option for specializing agents leads to well-defined responsibilities, allowing agents to focus on their purpose to minimize some drawbacks. For example, when interfacing hardware, it is interesting that the agent can deliberate, considering the perceptions available at that mo-

Agent	New Action	Description
Argo	.port(S);	Defines a serial communication port with an IoT device.
	.limit(N);	Defines an interval for the cycle of environmental perception.
	.percepts(open|close);	Listens or not the environmental perceptions.
	.filter(add | remove, c, P, C);	Defines or not an environmental perception filter.
	.act(O);	Sends an order to the microcontroller to execute.
Communicator	.connectCN(T,G,E,U,K);	Joins an IoT network.
	.sendOut(D,f,M);	Dispatches a message to another MAS.
	.moveOut(D,b,A);	Carries over the agents to other MAS.
	.disconnectCN;	Leaves the IoT network.

Where:
A is one, all, or a set of agents (i.e., all, agent or $[agent_1, agent_2, agent_n]$).
C is an optional field representing the necessary context for applying the filter.
D It is a literal that represents the identification of the recipient MAS.
E is a number that represents the network port of an IoT gateway.
G is a literal that represents the FQDN or the network address of an IoT gateway.
K is an optional field representing the device's credentials in communication technology.
M is a literal that represents the message.
N is a positive number (N>0) that represents an interval in milliseconds.
O is a literal that represents an order for the microcontroller to execute.
P is one or a set of environmental perceptions (i.e., perception or $[perception_1, perception_2, perception_n]$).
S is a literal that represents a serial port (i.e., ttyACM0).
T is a literal that represents the communication technology.
U is a literal that represents the identification of the device in the IoT network.
V is an optional field that represents a context to apply the filter.
b is a bio-inspired protocol (inquilinism | mutualism | predation).
c is a filter criterion (all | comply | except | only).
f is a illocutionary force (tell | untell | achieve | unachieve).

Table 1: The Jason Embedded's internal actions

ment. Suppose the agent is compromised to some other task as communication or trying to achieve another intention. So, it could affect the agent's reaction in some applications and domains where the response time is essential — i.e., object deviation.

3.1 Controlling Hardware

Argo is a customized agent architecture built on the Jason framework that extends Jason's standard agents by adding the ability to control microcontrollers [24]. Argo agents interact with the physical environment, capturing information from the environment using sensors and acting by sending commands to the microcontroller to activate the actuators. Sensors' information is automatically processed as perceptions in the agent's belief base. Argo agents have four internal actions to control microcontrollers: port, limit, percepts, act, and filter. These actions define which port the agent is accessing, a time limit for accessing them, if they open or close the flow of perceptions, activate actuators, or filter the incoming percepts, respectively. Argo architecture was modified to work properly along with all existing architectures since it was initially designed to be an ad-hoc solution. Furthermore, the behavior of swapping resources [17] at runtime was added to avoid stopping the Embedded MAS during its execution.

3.2 Communicability

The Communicator agent is another customized agent architecture built on Jason's Standard agent by adding the ability to communicate with agents from other MAS, which also have Communicator agents [23]. It allow agents from different MAS to interact, exchange knowledge and even collaborate between them. The Communicator agent can interact by exchanging knowledge and even collaborating between them using KQML performatives [14]. To use these new capabilities, the communicator agent has specific internal actions: connect, disconnect, and sendOut. These actions connects and disconnects from the IoT Gateway, and send a message to another Embedded MAS. The Communicator agent is a client instance of an IoT middleware [13] that needs an IoT Gateway to communicate. The agent can send a message to another MAS using this IoT middleware even if the target is disconnected. Once it connects, the message is redirected.

In this case, the original communicator architecture was totally modified to allow the connection in the IoT Gateway generically and work with Argo agents in the same Embedded MAS. The connect and disconnect are new functionalities once the agent sometimes can deliberate whether to be offline.

3.3 Mobility

Subsequently, the Communicator architecture was extended with bioinspired protocols [27] following ecological relations concepts that allow agents to be transferred between different MAS. They can be used to preserve the MAS knowledge if the physical device is damaged since it is subject to the unpredictability of the real world. The bioinspired protocols currently have three ecological relations implemented: Predation, Inquilinism, and Mutualism: in Predation, all agents are transferred to the target MAS to prey and dominate. It eliminates all agents of the target and the origin MAS (after moving them) to prevent unwanted access to any residue of its knowledge; in the Inquilinism, it sends all its agents to the target MAS to preserve its knowledge, but they do not interfere in its activities and existence. However, similar to the predation protocol, the origin MAS is eliminated to prevent unwanted access to any residue of knowledge; the Mutualism sends an agent, a group of agents, or the entire MAS to interact, learn, and transmit knowledge to a target MAS. Agents using this protocol can return to the origin MAS or move to any other if allowed.

The Communicator has the moveOut internal action for the mobility behavior. It uses the IoT middleware instance to move agents from one system to another. In this version, Mutualism was modified to send all agents except the Communicator since the agents need a way to return to the origin MAS. In the Inquilinism, we modified the protocol to drop all desires in the target MAS of the moving agents since they cannot interfere with the MAS functioning. We modified the protocols to guarantee the departure and arrival of agents to avoid communication problems and that agents do not get lost during the agents' transference.

4 Case Study

In the case study, we consider a scenario, shown in Figure 2, that integrates a smart home, a central control, and an unmanned vehicle, all controlled by Multi-agent Systems. The MAS running in the central control has a Communicator to forward service calls and two Jason agents for decision-making in case of calls. The vehicle has an Argo agent for driving and a Communicator agent for receiving agents to use the vehicle. The MAS running in the house has a Communicator agent responsible for interacting with the central and an Argo agent controlling the sensors and actuators.

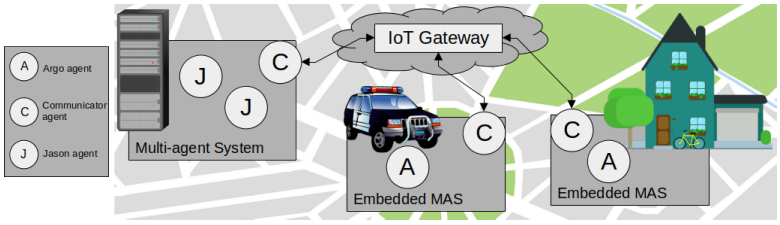

Fig. 2: The scenario of the case study proposed.

In this scenario, when the Argo agent in the House's Embedded MAS perceives a change in a specific sensor, it will inform the Communicator agent to send a message to central control. In the central control, the message is forwarded to a coordinator agent, who will send an agent to the house to analyze it. After completion, the sent agent will send a message to the coordinator requesting a patrol. The coordinator agent will migrate to an unmanned vehicle to execute the mission. When in the vehicle, the coordinator will inform the driving agent about the path.

To fulfill the proposed scenario, we built three Embedded MAS: the first – Batcave project – represents the house, hosted in a Raspberry Pi with an Arduino Board and has an LDR (light dependent resistor) sensor and a LED (light-emitting diode) as the actuator; the second – WayneMansion project – represents the control center, hosted in a Raspberry Pi; the third – BatMobile project – represents the unmanned vehicle, hosted in a ChonBot 2WD [18] prototype.

When the Argo agent in the house MAS (batCave.mas2j) notices a change in the environment (batSignal(true)) it activates an actuator (led(red)) and requests the Communicator agent to forward an alert to the center control MAS (wayneMansion.mas2j). The Figure 3 shown the code of the BatCave project.

When this alert arrives at the destination, it is forwarded to the Jason agent responsible for handling occurrences (bruce.asl); when analyzing the alert, the agent Bruce decides to designate other Jason agent (alfred.asl) to verify the situation (at MAS BatCave). The agent Alfred requests the Communicator agent of WayneMansion MAS to transfer itself to BatCave MAS; after this, the Communicator of BatCave MAS receives the agent Alfred in the system. Alfred requests

Fig. 3: The implementation of the BatCave project.

the Argo agent for a new check of sensors and to change the alert state to yellow (led(yellow)); finally, Alfred requests to Communicator of BatCave MAS to send a message to agent Bruce. The Figure 4 shown the code of the WayneMansion project.

Fig. 4: The implementation of the WayneMansion project.

The Communicator of WayneMansion MAS forwards the message to Bruce, who decides to patrol near the house. In this way, it requests the Communicator to transport itself to the autonomous unmanned vehicle (batMobile.mas2j); once it arrives in the BatMobile MAS, the agent Bruce requests to Argo agent to pilot the vehicle (goAhead). Figure 5 shown the implementation of the BatMobile project. Finally, all projects' Argo and Communicator agents have standard plans, beliefs, and intentions shown in Figure 6.

Fig. 5: The implementation of the BatMobile project.

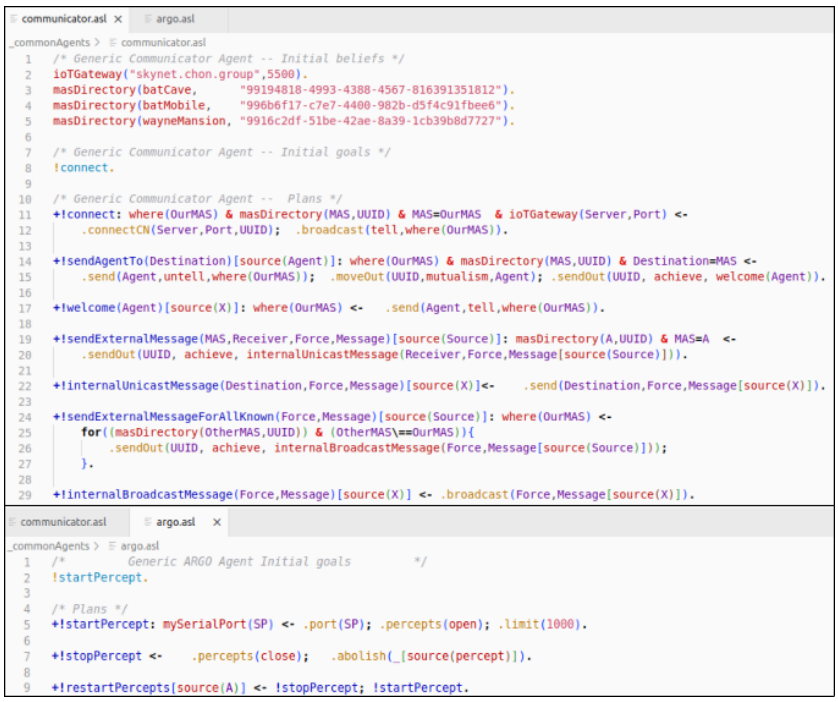

Fig. 6: The common source of the Argo and Communicator agents.

Aiming to enable the reproducibility of what is presented in this work, the framework and its source code, the implementation of the case study, and a demonstration video using prototypes representing the house, the control center, and the unmanned vehicle of the proposed scenario are available[5].

[5] http://bracis2023.chon.group/

5 Discussion

This paper presented an extended version of the Jason framework named Jason Embedded to program autonomous devices using BDI agents. It provides two new types of agents for allowing hardware interfacing, IoT-based communication, and mobility between different Embedded MAS. Besides, Jason's agents can deal with the MAS's internal issues where it lives. We also presented a study case to cover the new functionalities provided by the extended framework.

Using an Embedded MAS could bring advantages compared to adopting just one agent to interface all prototype's functionalities. One agent could be overloaded depending on how much information it gathers or which goals it is pursuing. The decision to use a framework that allows specialized agents to deal with certain functionality leads to an internal distributed solution to optimize the response to stimuli. If one agent has to deal with both perception gathering and external mail checking, it can eventually get overloaded. A prototype embedded with a MAS is autonomous and proactive by its capabilities of perceiving and acting upon the real world. However, when dealing with prototypes, it is important that these autonomous devices could collaborate somehow. Jason Embedded allows these prototypes to exchange messages using the Communicator agents. Then, devices with different Embedded MAS can negotiate, coordinate and exchange information without being part of the same physical architecture or network. They only need the Internet to connect to the IoT server.

Every Embedded MAS is an Open MAS if it adopts at least one Communicator agent. An Open MAS allows agents to enter and leave the system whenever necessary. So, in our extended version, the Communicator is responsible for moving agents (and itself) to other systems by invoking bio-inspired protocols. Depending on the domain, agents can move and dominate the destiny system or co-exist within the new one. In some cases, the agents can come and go from a system (mutualism). These two contributions now allow communicability and mobility in BDI agents using Jason by using some new internal actions since all background technologies are abstracted from the designer.

Jason Embedded could run on any platform or IDE that the designer chooses. But to exploit the most of it, it can work along the ChonOS and ChonIDE, which are an OS distribution and an IDE, to support the embedding process and develop a MAS using Jason Embedded. The OS comprises all the necessary technological dependencies in a single distribution, including the IDE to program the firmware and the MAS, start and stop the Embedded MAS, verify the prototype's outputs (logs), and inspect the mind of any agent. We remark that it has been an initiative of our research group to develop embedded solutions during the past years. In this paper, we consolidate several solutions that used to work alone into a single framework to facilitate their use and adoption.

In future works, we will provide an alternative to program artifacts that interfaces hardware using a serial interface and CArtAgO [20]. Then, the designer can choose between Argo agents and artifacts. The former is one agent dedicated to interface sensors and actuators, while the latter could be available to all agents existing in the system. Furthermore, the Communicator agent needs a different

format for identifying itself in the system since it uses a non-intuitive hexadecimal number. One possibility would be to use an email address to identify the Communicator agent. Besides, two new protocols could help in the development process of Embedded MAS: cloning could be used to send a copy of one or more agents to another system without killing these agents in origin; the cryogenics could be used to dump all agents into files and then restart them in the future. From the technology point of view, we intend to match the Jason Embedded version with the most recent Jason framework. This version uses version 1.4.1 since some single-board computers only run Java 8, and the recent Jason framework uses Java 17. Another point is to exploit alternative communication infrastructures such as SMTP/POP3 or XMPP to offer more reliability and privacy since ContextNet uses UDP, and the message is transmitted without cryptography.

References

1. Akhtar, S.M., Nazir, M., Saleem, K., Mahfooz, H., Hussain, I.: An Ontology-Driven IoT based Healthcare Formalism. International Journal of Advanced Computer Science and Applications **11**(2) (2020). https://doi.org/10.14569/IJACSA.2020.0110261
2. Baccour, E., Mhaisen, N., Abdellatif, A.A., Erbad, A., Mohamed, A., Hamdi, M., Guizani, M.: Pervasive ai for iot applications: A survey on resource-efficient distributed artificial intelligence. IEEE Communications Surveys Tutorials **24**(4), 2366–2418 (2022). https://doi.org/10.1109/COMST.2022.3200740
3. Barnier, C., Aktouf, O.E.K., Mercier, A., Jamont, J.P.: Toward an embedded multiagent system methodology and positioning on testing. In: 2017 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW). pp. 239–244 (2017). https://doi.org/10.1109/ISSREW.2017.57
4. Bordini, R.H., Hübner, J.F., Wooldridge, M.: Programming Multi-Agent Systems in AgentSpeak Using Jason (Wiley Series in Agent Technology). John Wiley Sons, Inc., Hoboken, NJ, USA (2007), https://dl.acm.org/doi/10.5555/1197104
5. Brandao, F., Nunes, P., de Jesus, V.S., Pantoja, C.E., Viterbo, J.: Managing natural resources in a smart bathroom using a ubiquitous multi-agent system. In: Proceedings of the 11th Workshop-School on Agents, Environments, and Applications (WESAAC 2017). pp. 101–112. FURG, São Paulo (2017)
6. Brandão, F.C., Lima, M.A.T., Pantoja, C.E., Zahn, J., Viterbo, J.: Engineering Approaches for Programming Agent-Based IoT Objects Using the Resource Management Architecture. Sensors **21**(23), 8110 (Dec 2021). https://doi.org/10.3390/s21238110
7. Bratman, M.: Intention, Plans, and Practical Reason. Cambridge: Cambridge, MA: Harvard University Press (1987)
8. Burattini, S., Ricci, A., Mayer, S., Vachtsevanou, D., Lemee, J., Ciortea, A., Croatti, A.: Agent-oriented visual programming for the web of things (2022), https://www.alexandria.unisg.ch/handle/20.500.14171/109205
9. Souza de Castro, L.F., Manoel, F.C.P.B., Souza de Jesus, V., Pantoja, C.E., Pinz Borges, A., Vaz Alves, G.: Integrating Embedded Multiagent Systems with Urban Simulation Tools and IoT Applications. RITA **29**(1), 81–90 (Jan 2022). https://doi.org/10.22456/2175-2745.110837

10. Chander, B., Pal, S., De, D., Buyya, R.: Artificial Intelligence-based Internet of Things for Industry 5.0, pp. 3–45. Springer International Publishing, Cham (2022). https://doi.org/10.1007/978-3-030-87059-1_1
11. Ciortea, A., Mayer, S., Michahelles, F.: Repurposing manufacturing lines on the fly with multi-agent systems for the web of things. In: Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems. p. 813–822. AAMAS '18, International Foundation for Autonomous Agents and Multiagent Systems, Richland, SC (2018), https://dl.acm.org/doi/10.5555/3237383.3237504
12. D'Urso, F., Longo, C.F., Santoro, C.: Programming intelligent iot systems with a python-based declarative tool. In: Proceedings of the 1st Workshop on Artificial Intelligence and Internet of Things co-located with the 18th International Conference of the Italian Association for Artificial Intelligence (AI*IA 2019), Rende (CS), Italy. CEUR Workshop Proceedings, vol. 2502, pp. 68–81. CEUR-WS.org (2019)
13. Endler, M., Baptista, G., Silva, L.D., Vasconcelos, R., Malcher, M., Pantoja, V., Pinheiro, V., Viterbo, J.: Contextnet: Context reasoning and sharing middleware for large-scale pervasive collaboration and social networking. In: Proceedings of the Workshop on Posters and Demos Track. PDT '11, Association for Computing Machinery, New York, NY, USA (2011). https://doi.org/10.1145/2088960.2088962
14. Finin, T., Fritzson, R., McKay, D., McEntire, R.: Kqml as an agent communication language. In: Proceedings of the Third International Conference on Information and Knowledge Management. p. 456–463. CIKM '94, Association for Computing Machinery, New York, NY, USA (1994). https://doi.org/10.1145/191246.191322
15. Kampik, T., Nieves, J.C.: JS-son - A Lean, Extensible JavaScript Agent Programming Library. In: Dennis, L.A., Bordini, R.H., Lespérance, Y. (eds.) Engineering Multi-Agent Systems, vol. 12058, pp. 215–234. Springer International Publishing, Cham (2020). https://doi.org/10.1007/978-3-030-51417-4_11
16. Karaduman, B., Tezel, B.T., Challenger, M.: Enhancing bdi agents using fuzzy logic for cps and iot interoperability using the jaca platform. Symmetry **14**(7) (2022). https://doi.org/10.3390/sym14071447
17. Lazarin., N., Pantoja., C., Viterbo., J.: Swapping physical resources at runtime in embedded multiagent systems. In: Proceedings of the 15th International Conference on Agents and Artificial Intelligence - Volume 1: ICAART,. pp. 93–104. INSTICC, SciTePress (2023). https://doi.org/10.5220/0011750700003393
18. Lazarin, N., Pantoja, C., Viterbo, J.: Towards a toolkit for teaching ai supported by robotic-agents: Proposal and first impressions. In: Anais do XXXI Workshop sobre Educação em Computação. pp. 20–29. SBC, Porto Alegre, RS, Brasil (2023). https://doi.org/10.5753/wei.2023.229753
19. Lazarin, N.M., Pantoja, C.E.: A robotic-agent platform for embedding software agents using raspberry pi and arduino boards. In: Proceedings of the 11th Workshop-School on Agents, Environments, and Applications WESAAC 2015. pp. 13–20. Niteroi (2015)
20. Manoel, F., Pantoja, C.E., Samyn, L., de Jesus, V.S.: Physical Artifacts for Agents in a Cyber-Physical System: A Case Study in Oil & Gas Scenario (EEAS). In: The 32nd International Conference on Software Engineering and Knowledge Engineering, SEKE 2020. pp. 55–60. KSI Research Inc. (2020)
21. Michel, F., Ferber, J., Drogoul, A.: Multi-Agent Systems and Simulation: a Survey From the Agents Community's Perspective. In: Danny Weyns, A.U. (ed.) Multi-Agent Systems: Simulation and Applications, p. 47. Computational Analysis, Synthesis, and Design of Dynamic Systems, CRC Press - Taylor & Francis (May 2009)

22. Palanca, J., Terrasa, A., Julian, V., Carrascosa, C.: Spade 3: Supporting the new generation of multi-agent systems. IEEE Access **8**, 182537–182549 (2020). https://doi.org/10.1109/ACCESS.2020.3027357
23. Pantoja, C., Soares, H.D., Viterbo, J., Seghrouchni, A.E.F.: An Architecture for the Development of Ambient Intelligence Systems Managed by Embedded Agents. In: The 30th International Conference on Software Engineering & Knowledge Engineering. pp. 215–249. KSI Research Inc, San Francisco Bay (Jul 2018). https://doi.org/10.18293/SEKE2018-110
24. Pantoja, C.E., Stabile, M.F., Lazarin, N.M., Sichman, J.S.: Argo: An extended jason architecture that facilitates embedded robotic agents programming. In: Engineering Multi-Agent Systems. pp. 136–155. Springer International Publishing, Cham (2016). https://doi.org/10.1007/978-3-319-50983-9_8
25. Rafferty, L.: Agent-based modeling framework for adaptive cyber defence of the Internet of Things. PhD Thesis, Faculty of Business and IT, University of Ontario Institute of Technology, Oshawa, Ontario, Canada (2022)
26. da Rocha Costa, A.C., Hübner, J.F., Bordini, R.H.: On entering an open society. In: XI Brazilian Symposium on Artificial Intelligence. vol. 535, p. 546 (1994)
27. Souza de Jesus., V., Pantoja., C.E., Manoel., F., Alves., G.V., Viterbo., J., Bezerra., E.: Bio-inspired protocols for embodied multi-agent systems. In: Proceedings of the 13th International Conference on Agents and Artificial Intelligence - Volume 1: ICAART. pp. 312–320. INSTICC, SciTePress (2021). https://doi.org/10.5220/0010257803120320
28. Wooldridge, M.: An Introduction to MultiAgent Systems. Wiley (2009)