Managing Natural Resources in a Smart Bathroom Using a Ubiquitous Multi-Agent System

Fabian Brandão[2] , Palloma Nunes [2] , Vinicius Souza de Jesus[2] , Carlos Eduardo Pantoja[1,2] , José Viterbo[1]

[1] Universidade Federal Fluminense (UFF)

[2]Centro Federal de Educação Tecnológica Celso Suckow da Fonseca (CEFET/RJ)

fabiancpbm@gmail.com, pallomapit@hotmail.com, souza.vdj@gmail.com

pantoja@cefet-rj.br, viterbo@ic.uff.br

Abstract. *This work presents a prototype using a Multi-Agent System for managing a smart bathroom model that will help in the economy of natural resources (electrical energy and water) using perceptual information from sensors and low-cost controllers. The MAS was developed using Jason framework and ARGO agents and the bathroom uses an ATMEGA328 controller and several sensors. The prototype is able of controlling certain available functions such as shower, discharge or sink for water resources and it chooses between commercial and renewable energy. Besides, there is an emergency system for helping people inside the bathroom. Some performance tests were executed to explore the use of robotic agents programmed with Jason and ARGO.*

1. Introduction

The Ambient Intelligence (AmI) is an autonomous environment that has electronic components interconnected working together with software and its goal is to make up daily activities imperceptible [Aarts and De Ruyter 2009]. In turn, ubiquitous systems are autonomous computer programs able to contact with humans in an invisible way. The Multi-Agents System (MAS) is an autonomous system composed of autonomous agents able to make decisions based on its beliefs, interaction with others agents and environment's perceptions [Wooldridge 2009]. Many authors suggest that the MAS approach can be considered in the development of AmI applications and ubiquitous systems [Chaouche et al. 2014].

Some works [Martins and Meneguzzi 2014][Andrade et al. 2016] uses MAS mostly applied in simulated environments, thereby when applying these MAS in the real world, there is no guarantees of their effectiveness, since the real world is a dynamic environment which generates constant perceptual information that can compromise the MAS performance. There are some works about MAS applied in a real environment too such as [Villarrubia et al. 2014],[Hagras et al. 2004],[Cook et al. 2003][Kazanavicius et al. 2009], however, these works present an architecture that was structured only for a particular domain or it is not reusable.

Jason [Bordini et al. 2007] is a framework that allows the development of MAS based on the Belief Desire Intention (BDI) [Bratman 1987] architecture, using an AgentSpeak [Rao 1996] interpreter in Java where agents can be situated in an environment, usually simulated. ARGO [Pantoja et al. 2016] is a customized architecture of Jason agents

that provides communication with low-cost controllers, actuators and sensors allowing an interaction with the real world. ARGO is an attempt to provide an uncoupled architecture where the high-level language is programmed without being coupled to the hardware to facilitate the development and prototyping of ubiquitous MAS and AmI using Jason. One of ARGO's differential is to generalize and facilitate the construction of MAS regardless of the hardware assembled and it is possible to use popular controllers, such as Arduino.

The objective of this work is to employ the MAS approach applied in a physical intelligent bathroom that is able to manage hydric and electrical resources for economy and sustainability. It is represented as a prototype composed of sensors and actuators that one or more agents can use to interact with the real world. The MAS was programmed using Jason and ARGO where intelligent agents could control the resources based on their beliefs and decide the actions to be taken in a near real-time situation. Besides, an emergency verification process is presented. It is started with the bathroom empty and if somebody decided to enter, he or she is able of interacting with the devices (for example shower, sink and discharge) pervasively. Once inside, if the person is not feeling well and the presence sensor does not capture movements, a sound would be emitted as an alert. If there is no presence after some seconds, another sound would be emitted continuously and the door would be opened for someone to help. The prototype constructed is physically connected to only one micro-controller in order to explore the situation where multiple functions need to be handled by a MAS but the hardware is limited (this situation can lead to undesirable delays).

The expected contributions of this work are: to provide a smart bathroom model applied and tested in a physical prototype; explore the use of ARGO agents for prototyping ubiquitous Multi-Agent System (uMAS); and an analysis of some strategies for programming agents which interact and control hardware devices. This work is structured as follows, in section 2 some background of MAS development can be seen; section 3 presents the uMAS approach for a bathroom in a smart home; section 4 presents the prototype of the smart bathroom; some related works are discussed in section 5, and finally, conclusion and future works are shown.

2. Ubiquitous Multi-Agent Systems

In this section, will be presented the technological background that allowed to create an uMAS applied in a smart bathroom model. The Jason framework allows creating a MAS, whereas, the ARGO customized architecture allows a MAS programmed in Jason to perceive and realize actions in the real world using controllers.

2.1. Jason Framework

The Jason Framework is a Java-based interpreter of AgentSpeak(L), which provides the creation of cognitive agents based on the Belief-Desire-Intention (BDI) architecture and the Procedural Reasoning System (PRS). The BDI allows creating cognitive agents based on three concepts: Beliefs, that are information acquired by others agents of the SMA, by environment's perceptions and mental notes; Desires, that are the interest and motivation to achieve a goal; and Intentions, that are deliberated actions. Furthermore, the Jason agent has a reasoning system, based on the PRS, which is a mechanism to perform the decision-making according to beliefs and perceptions. The PRS allows the agents to

reason in order to accomplish complex tasks immersed in a dynamic environment with several agents.

Every Jason agent has a reasoning cycle, which is initialized by perceiving the environment or by another agent's messages received, and across a succession of internal events, the result is actions to be taken or messages sent. However, the Jason agent can only perceive the simulated environment and, depending on the number of perceptions processed, it can generate delays on the reasoning cycle. For this purpose, there is a customized architecture of agents that can perform a communication with the real world through the use of devices and can also filter unnecessary perceptions.

2.2. ARGO Agents

The ARGO is a customized architecture of Jason agents, which enables the communication between agents and controllers. The architecture uses a communication protocol named Javino for interacting with devices and can use perception filters (to improve its performance when reasoning with information from sensors).

Javino [Lazarin and Pantoja 2015] is a middleware that allows the communication between high-level languages and controllers that have a serial connection (ATMEGA in Arduino, for example). This middleware has an error identification structure that prevents loss of data in messages sent at runtime. Moreover, Javino is prepared to deal with the sending and reception of messages equally from Java to the controller, and from the controller to Java. The perception filter [Stabile Jr. and Sichman 2015] is a mechanism that makes the agent block environment perceptions based on filters designed by the developer at design time. These perception filters (used as XML files) are named according to the name of the agent who uses the filter and to the filter methodology types. Besides, the filters can be changed at runtime by the agent who owns the file.

There are two kinds of agents, the standard agent of Jason and the customized ARGO agent. The standard agent makes decisions in an autonomous way in simulated environments. In turn, the ARGO agent has the same abilities of a standard agent, but it has the function of communicating with hardware devices (controllers). ARGO agents are programmed using 5 internal actions: *.port()*, where the programmer pass the identification of the serial port where the controller is connected, by parameter; *.percepts()*, which is responsible for allowing to get perceptions if open is the parameter, or not if block is the parameter; *.limit()*, which delimits (passing the milliseconds per parameter) the interval of time that an agent will perceive from the environment; *.act()*, which sends a message, per parameter, from the agent to the controller; and *.filter()*, which invokes the XML file of a perception filter by passing its name by parameter.

3. The MAS Approach in Home Systems

In this section, it is presented the idea of the intelligent bathroom and its architecture. The aim is to save and reuse hydric and electrical resources in an autonomous way showing how people can manage resources applying MAS using the bathroom. The save of resources can happen when the water used in the shower and in the bathroom sink are placed on a specific box of water instead of being wasted; the rainwater is captured and placed on the second box of water; and when the solar boards capture the energy and charge some rechargeable batteries. The reuse of these resources happens when the sanitary discharge

uses the water of the first or the second box of reused water mentioned before; when the shower and the bathroom sink use the rainwater in the second box mentioned; or when the electrical energy used is the energy present in the batteries.

Moreover, the proposed intelligent bathroom is able to manage also the security of the user inside the bathroom. In the bathroom, the person only can get in if it is empty or if someone is not feeling well inside the bathroom. A sound is emitted to report that something is wrong with the person which occupy the bathroom. Another issue is that the bathroom's door has an automatic operation in the idealized bathroom.

To indicate when someone is not feeling well, is necessary first to understand the logical status of the bathroom. There are six status mode: *free*, *busy*, *attention*, *wait*, *alert*, and *in help*, which constantly change (according to the perceptions of the presence sensor, door's actions, and previous status). It begins with the *free* status, which means that nobody is inside the bathroom and anyone can enter. The *wait* status happens all the times that the door's button is pressed and the person has not entered yet. The *busy* status imply that someone is inside the bathroom and this person is feeling well. When the previous status is busy and the sensor captures no presence, the status changes immediately to attention and remains in this mode, during a certain time. While the sensor continues to identify no movements, internal sound notices are sent while the*attention* status remains the same. But, passing a certain time in the *attention* status without capturing no movements, it changes its status to *alert* (in this moment the buzzer is turned on in an external sound notice, meaning that something is not good with the person who is inside the bathroom). Finally, the *in help* status happens when someone is not feeling well inside the bathroom and another person enters in it. It is important to remember that if, at a certain moment that the status is *attention* or *alert*, the presence sensor came back to perceive someone, the status changes to *busy*. This process can be seen in Figure 1.

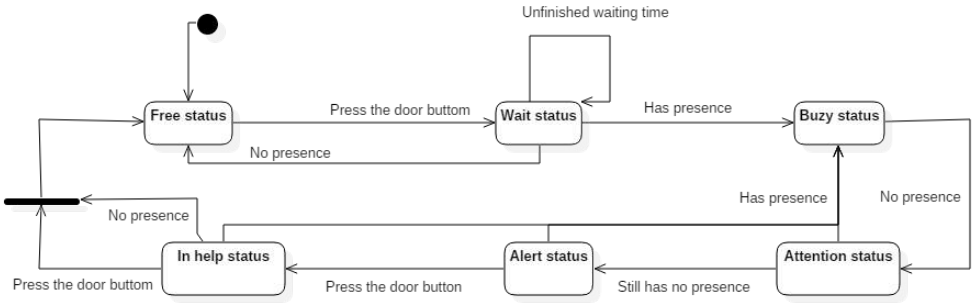

Figure 1. The logical status of the bathroom for an emergency situation.

4. The Prototype

In order to test the MAS approach, it was developed a prototype of a smart bathroom assembled with sensors and actuators. In this section, it is explained the physical structure of this prototype, which counts with a wood structure that looks like a reduced real bathroom; three boxes of water (one with rainwater and other with water bought, located on top of the part which the user access, and the last one with water already used in the shower and in the bathroom sink, below the structure); a miniature of a shower; the sanitary discharge with the bathroom sink; valves to control the water flow; electrical wires;

one power supply; hoses; buttons (to turn on or turn off the shower, the sanitary discharge, the bathroom sink, the door, and the use of clean energy); 3 water pumps; sensors (a presence sensor and a level sensor built with a potentiometer and a styrofoam ball); buzzer; 2 solar boards; rechargeable batteries and; a circuit employed with an ATMEGA328 controller. The prototype can be seen in Figure 2.

Figure 2. The Smart Bathroom Prototype.

An important issue to remark is the difference in the operation of the door idealized and of the door of the prototype: the door idealized works automatically when someone appears in front of it, and the door of the prototype works with an answer of a button pressing. The controller, mentioned before, is dedicated to one ARGO agent named *Manager*, which is responsible for capturing the perceptions from sensors and to action upon the real environment based on messages received from 9 standard agents.

In this MAS, standard agents work together with the ARGO agent where they are responsible for making decisions based on their beliefs, perceptions and message exchanged between agents and the former one is responsible for capturing the perceptions and send to them and to receive messages with actions to be executed by the actuators. The standard agents are: *door*, *shower*, *discharge*, *sink*, *light*, *status*, *reservoir*, *energy* and *pump*, that, intuitively, are responsible for managing each component and the status of the bathroom, where *reservoir* and *pump* are, respectively, responsible for defining the use of adequate reservoir and to maintain perceptions of the level of the rainwater box. The code for agent *manager* can be seen in Figure 3.

```
!start.

/* Plans */

+!start : true <-
    .port(COM3);
    .percepts (open);
    .percepts (block);
    !start.

+!open_door : true <-
    .act(openDoor);
    .wait(3700);
    .send (door, achieve, trigger).

+!close_door : true <-
    .act(closeDoor);
    .wait(4600);
    .send (door, achieve, trigger).

+!turnOn_light : true <-
    .act(turnOnLight).

+!turnOff_light : true <-
    .act(turnOffLight).
```

```
+!return_water : true <-
    .act(turnOnPumpReturn);
    .wait(15000);
    .act(turnOffPumpReturn);
    .wait(1000).

+!turnOn_shower : true <-
    .act(turnOnShower).

+!turnOff_shower : true <-
    .act(turnOffShower).

+!turnOn_discharge : true <-
    .act(turnOnDischarge).

+!turnOff_discharge : true <-
    .act(turnOffDischarge).

+!turnOn_sink : true <-
    .act(turnOnSink).

+!turnOff_sink : true <-
    .act(turnOffSink).

+!potable_water: true <-
    .act(potableWater).
```

```
+!rain_water: true <-
    .act(rainWater).

+!turnOn_cistern_pump: true <-
    .act(turnOnCisternPump).

+!turnOff_cistern_pump: true <-
    .act(turnOffCisternPump).

+!turnOff_all : true <-
    .act(turnOffShower);
    .act(turnOffSink);
    .act(turnOffDischarge);
    .act(turnOffRainPump);
    .act(turnOffPotablePump);
    .act(turnOffLight).

+!turnOn_low_buzzer : true <-
    .act(lowAlarm).

+!turnOff_low_buzzer : true <-
    .act(turnOffLowAlarm).

+!turnOn_high_buzzer : true <-
    .act(highAlarm).

+!turnOff_high_buzzer : true <-
    .act(turnOffHighAlarm).
```

Figure 3. The agent manager.

So, the intelligent agents, are responsible for the security and decision-making related to the economy of the bathroom resources. For this, the security and supervision actions work with a logical status mode as mentioned before; that is assumed by the *status* agent according to the perceptions of the presence sensor, door's action, and the previous status. With the bathroom initially empty, a user can press the button to open the door. In this case, the agent *door* opens it and the user can get in. Once inside the bathroom, if the agent *light* perceives somebody inside, the light turns on and the logical status changes from *free* to *wait*, and after a few seconds to *busy* (at this moment, if another user tries to enter, the agent *door* does not open the door). Therefore, an example of the security action happens when the status change to *attention* if the presence sensor does not identify the presence of a person inside the bathroom and after that changes to *in alert*. In this moment, the buzzer is turned on by the agent *status* [1].

Before continue with the case, it is important to reinforce two things: first that every time that the door is open or close, an intermediate status *wait* happens until the door is totally opened or closed and; second that every time that is mentioned that the door is opened with the presence of someone in front of it, means that the door button was pressed to simulate the attempt of opening the door (with the presence of someone in front of it, in this case). Therefore, when the status is *in alert*, the buzzer is turned on by

[1]The video of this example running can be seen at https://youtu.be/h-xC9pk2ri4

the agent *status* and anyone that appear in front of the door can make it open automatically. Moreover, the prototype has a particular problem that is to have just one controller. This problem can cause delay by the number of requisitions done for several simple agents to just one ARGO agent and, consequently, the loss of pervasiveness expected of an AmI. Because of this, it was necessary to make tests to see which strategy was appropriated.

Besides, when the water pump and the valves, necessary to supply the components that use water are turned on, the comparisons of levels are done (with the level sensor) by agent *reservoir* and the box of water related with the water bought from some distributer are the last one to be used (being used only if the others do not have enough water for use). It is important to remember that even the shower and the bathroom sink (agents *shower* and *sink* respectively) can only receive water from the two boxes of water located on top of the prototype, but the *discharge* agent, verifies the box located below the prototype, and then, if this box is without enough water for use, it uses one of the boxes located on top. Besides, the economy of electricity happens putting the energy absorbed from the solar boards in rechargeable batteries; when it is necessary to use the electricity, a comparison circuit checks (done by agent *energy*) and if there is enough energy in the batteries and, whenever possible, it uses this one first.

Once inside the bathroom, if the person turns the shower on, in this moment, the agent *shower* checks if there is water enough in the rainwater box; if yes, the water pump related to the rainwater box and the valve related to the shower are turned on; if no, the water pump related to the bought water box and the valve related to the shower are turned on. The person can decide to use the bathroom sink too, for example. The difference between them is only the button and the agent responsible for the functioning (agent *sink*), the subsequent processes are the same as was explained before. The code for agent *sink* can be seen in Figure 4.

```
/* Initial beliefs and rules */

sink_status(off).

/* Initial goals */

!start.

/* Plans */

+!start : true <-
    .send(manager, askOne, sink(Status), Reply);
    -+Reply;
    !trigger;
    !start.

+!trigger : sink(1) & sink_status(off) <-
    .send (manager, achieve, turnOn_sink);
    -sink_status(off);
    +sink_status(on).

+!trigger : sink(0) & sink_status(on) <-
    .send (manager, achieve, turnOff_sink);
    -sink_status(on);
    +sink_status(off).

-!trigger : true <-
    .print("sink").
```

Figure 4. The agent sink.

To use the sanitary discharge, the user presses the button but, in this time, the agent *sanitary* checks first if there is enough water in the box located below the bathroom; if

yes, the water pump related to this box and the valve related to the sanitary discharge are turned on; if no, this same agent checks if there is enough water in the rainwater box; if yes, the water pump related to the rainwater box and the valve related to the sanitary discharge are turned on; if no, the water pump related to the box of bought water and the valve related to the sanitary discharge are turned on [2].

4.1. Results

Some tests were performed considering the response time of agents in each interaction with the prototype (for example, buttons and sensors). In the first test, each standard agent was picked for working along with the ARGO agent. The response time was measured considering from 1 to 8 agents plus the ARGO agent in every test. Three repetitions were done for each case. It as used the average and the standard deviation to verify if exist variations in the measures. In this test, the emergency process (*status* agent) were excluded. In general, the results showed, that as the number of agents was increased the response time was increased as well. It can happen because of the use of several simple agents asking for perceptions and actions from the same ARGO agent and because the prototype is employed with only one ATMEGA328. The ARGO agent has a stack of requisitions to answer that increases depending on the number of agents of the MAS. This stack can generate delays in the execution time. It is important to remark that the best approach is to use one controller for a few functions of the prototype. In this paper, we focus on test the performance of the MAS when the number of controllers is limited (one controller in this case). To evaluate this performance it is shown, in figure 5, some of the results considering the number of agents and the response time in seconds.

We also performed throughput tests considering the number of requests for perceptions per second that were done by agents (without limiting the interval between these requests). It is also verified the number of perceptions that were discarded by the agent. As results, it was verified that, in average, the agent in these tests shows a throughput 5 requests per second, where 100% of them arrived without being discarded by Javino, who for this work it is very good. However, since there are 8 simple agents trying to communicate with the ARGO agent, the perceptions are not always updated in each reasoning cycle (since the agent can block or open the flow of perception at runtime).

In the test that was done with the *status* agent we considered the time of execution from *free* status until it comes back to *free* status again). Again, it was performed three repetitions for each of the following two cases: the agent *status*, the ARGO agent and more 2 simple agents; and the agent status, the ARGO agent and more 7 simple agents. The results show the same pattern from the former results of basic functions.

To address the limitation of having just one controller, two tests were realized: the first one the manager agent (that was programmed in a reactive manner) is responsible for the sensing and to distribute these perceptions to other agents while the remaining agents are responsible for the reasoning and; the second one is similar to the former one, however the manager agent was programmed using belief plans and not achievement plans. As a result, the first test had a better performance comparing with the second test, seeing that the first presented an answer time smaller than the second test. Therefore, the first strategy, only one agent communicates with the controller and the other agents has been adopted

[2]The video of the basic functions running can be seen at https://youtu.be/cqi9QGvMvuA

Figure 5. The results for some of the agents of the smart bathroom (seconds per number of agents employed in tests).

(sending to these other agents, perceptions coming from the controller; and receiving commands from these same agents and sending to the controller). This chosen strategy shows that the approach can be significant in applications of AmI because it reduces the loss of pervasiveness if it is compared to the second approach (second test).

And the last test was performed to evaluate the MAS managing the reservoirs (potable, rainwater and cistern), pumps, status logic, energy sources (electric and solar) and manual drives. Thereby, it can be observed that there are many functionalities for a single controller to manage, however the different strategies made the response time acceptable or not, depending on the strategy adopted. Moreover, the functions of reusing and saving natural resources remained functional independent of the strategy, since the *energy* agent uses the maximum of clean energy that it obtains and the *reservoir* agent gives priority to use rainwater and has a policy of reuse.

5. Related Work

In this section, it is discussed some related works and they are compared with the smart bathroom presented in this work. Many papers show methods of using MAS for AmI

in Smart Homes, for example, the MavHome [Cook et al. 2003] is a Smart Home which works like an intelligent agent, that is, according to the necessities of the residents it offers comfort in day-by-day tasks improving the time of the residents, reducing operation costs, without forgetting of the security offered. The MavHome uses the MAS with agents divided into four layers named: decision, information, communication, and physical. The Ambient Intelligence Environment Using Embedded Agents [Hagras et al. 2004] uses concepts of ubiquitous computing environments, Ambient Intelligence, and embedded agents. In this way, it was created an environment named iDorm which was controlled by an embedded agent that uses fuzzy logic, inside an architecture of different networks. Without forgetting that the iDorm uses different kinds of embedded computational artifacts. Both of them present solutions that do not use an agent-oriented programming language and are specific and not reusable. Unlike, the smart bathroom presented here uses Jason as a programming language and also offers reductions in operation costs and security. Besides, ARGO is independent of the solution presented here and can be used in different domains.

In [Lim et al. 2009], an example of a project using MAS is the Smart Home with context aware, that uses communication and interaction protocols and the information about the ambiance are passed to the agents. In turn, the agent's task is to observe user actions, predict and analyze the risk of the task. To do this, it is important to remember that there are sub-agents which receive the sensors' catches and give to a super-agent (which, in turn, is connected with all the sub-agents). However, connecting a super agent to all sub agents can generate delays and the solution does not provide an independent platform able of programming uMAS in different domains. The proposed smart bathroom uses a similar approach, where one agent is responsible for getting perceptual information from sensors and it manages the requests of others agents to act upon the real world. Besides, ARGO is independent of domain, as stated before, and counts with mechanisms such as perception filters and blocking perceptions at runtime, which try to improve the performance of robotic agents.

6. Conclusion

This paper presented an uMAS of a smart bathroom model with the purpose of saving and reusing natural resources (water and energy). A prototype was constructed in order to simulate a real bathroom and the uMAS was developed using Jason and the ARGO architecture. Applying BDI agents in robotic platforms is not a simple task since problems can arise depending on the number of perceptual information. ARGO provides mechanisms such as perception filters and internal behaviors that block and unblock the flow of perceptions from the real world and another one which specifies a time interval for getting perceptions (both behaviors stop the internal process of capturing perceptions using Javino at runtime). These characteristics help in the development of uMAS with an acceptable response time depending on the programming strategy adopted. Then, in this work, it was explored a strategy using a centralized ARGO agent responsible for managing actions and perceptions interacting with one controller and 8 agents responsible for specific functions of the proposed smart bathroom negotiating actions with the centralizer.

It is important to remark that when employing more agents that controllers, the agents have to negotiate hardware resources and the performance can be compromised in actions that the time is a critical variable. For instance, verification of each energy to be

used (renewable or bought) is a function that does not impact the user of the bathroom if it delays 5 or 10 seconds. However, if the shower takes too long to be turned on after somebody presses the button, one can question the effectiveness of the system. The best scenario for prototyping an uMAS using ARGO agents is to provide an agent ARGO dedicated for each controller employed whenever possible and communicating with each other exchanging perceptual information or delegating to standard agents the reasoning of a specific function. Considering this, the experiments conducted showed that using a centralized agent when the number of functions are high and specialized per agents generates a bottleneck of requisitions (messages) that could delay in seconds the execution of certain functions.

For future works, an uMAS will be developed in a real room for managing the resources of a laboratory. Besides, a communication mechanism using a middleware for the internet of things will be tested using Jason and ARGO agents. This middleware aims to provide a way of two different MAS communicate. For example, a MAS responsible for one smart home will be able of communicating with a MAS responsible for a smart car.

Referências

Aarts, E. and De Ruyter, B. (2009). New research perspectives on ambient intelligence. *Journal of Ambient Intelligence and Smart Environments*, 1(1):5–14.

Andrade, J. P. B., Oliveira, M., Gonçalves, E. J. T., and Maia, M. E. F. (2016). Uma Abordagem com Sistemas Multiagentes para Controle Autônomo de Casas Inteligentes. In *XIII Encontro Nacional de Inteligência Artificial e Computacional (ENIAC)*.

Bordini, R. H., Hübner, J. F., and Wooldridge, M. (2007). *Programming Multi-Agent Systems in AgentSpeak using Jason*. John Wiley & Sons Ltd.

Bratman, M. E. (1987). *Intention, Plans and Practical Reasoning*. Cambridge Press.

Chaouche, A.-C., Seghrouchni, A. E. F., Ilié, J.-M., and Saïdouni, D. E. (2014). A higher-order agent model with contextual planning management for ambient systems. In *Transactions on Computational Collective Intelligence XVI*, pages 146–169. Springer.

Cook, D. J., Youngblood, G. M., Heierman III, E. O., Gopalratnam, K., Rao, S., Litvin, A., and Khawaja, F. (2003). Mavhome: An agent-based smart home. In *PerCom*, volume 3, pages 521–524.

Hagras, H., Callaghan, V., Colley, M., Clarke, G., Pounds-Cornish, A., and Duman, H. (2004). Creating an ambient-intelligence environment using embedded agents. *IEEE Intelligent Systems*, 19(6):12–20.

Kazanavicius, E., Kazanavicius, V., and Ostaseviciute, L. (2009). Agent-based framework for embedded systems development in smart environments. In *Proceedings of International Conference on Information Technologies (IT 2009), Kaunas*.

Lazarin, N. M. and Pantoja, C. E. (2015). A robotic-agent platform for embedding software agents using raspberry pi and arduino boards. In 9^{th} *Software Agents, Environments and Applications School*.

Lim, C., Anthony, P., and Fan, L. (2009). Applying multi-agent system in a context aware. *Borneo Sci*, 24:53–64.

Martins, R. and Meneguzzi, F. (2014). A smart home model using jacamo framework. In *2014 12th IEEE International Conference on Industrial Informatics (INDIN)*. IEEE.

Pantoja, C. E., Stabile, M. F., Lazarin, N. M., and Sichman, J. S. (2016). Argo: An extended jason architecture that facilitates embedded robotic agents programming. In Baldoni, M., Müller, J. P., Nunes, I., and Zalila-Wenkstern, R., editors, *Engineering Multi-Agent Systems: 4th International Workshop, EMAS 2016*, pages 136–155. Springer.

Rao, A. S. (1996). AgentSpeak(L): BDI agents speak out in a logical computable language. In de Velde, W. V. and Perram, J. W., editors, *Proceedings of the 7th European workshop on Modelling autonomous agents in a multi-agent world (MAAMAW'96)*, volume 1038 of *Lecture Notes in Artificial Intelligence*, pages 42–55, USA. Springer-Verlag.

Stabile Jr., M. F. and Sichman, J. S. (2015). Evaluating perception filters in BDI Jason agents. In 4^{th} *Brazilian Conference on Intelligent Systems (BRACIS)*.

Villarrubia, G., De Paz, J. F., Bajo, J., and Corchado, J. M. (2014). Ambient agents: embedded agents for remote control and monitoring using the pangea platform. *Sensors*, 14(8):13955–13979.

Wooldridge, M. (2009). *An Introduction to MultiAgent Systems*. Wiley.