A Heterogeneous Architecture for Integrating Multi-Agent Systems in AmI Systems

Vinicius Souza de Jesus and
Fabian Cesar Pereira Brando Manoel
CEFET/RJ
e-mail: souza.vdj,fabiancpbm@gmail.com

Carlos Eduardo Pantoja and José Viterbo
Universidade Federal Fluminense
e-mail: pantoja@cefet-rj.br,viterbo@ic.uff.br

***Abstract*—Several challenges arise when applying Multi-Agent System (MAS) in Ambient Intelligence scenarios such as the heterogeneity of the hardware and the domain where it is applied. There are several applications that use Agent-Oriented approaches but they provide solutions that tie the hardware to the software, and they do not provide generic architectures. So, in this paper, we propose a heterogeneous architecture for applying different microcontrollers in the design of embedded MAS for such kind of systems. An architecture and a small-scale prototype of a smart home assembled with several hardware devices connected to different ATMEGA and PIC microcontrollers are presented as proof-of-concept. Our architecture shows to be effective in several tests performed using different implementation strategies.**

I. Introduction

Ambient Intelligence (AmI) comprises electronic and intelligent environments characterized by the interconnection of different technologies with the purpose of helping users in their daily tasks in an autonomous, proactive and pervasively way [1]. Multi-Agent Systems (MAS) are composed of autonomous agents situated in an environment and have the capacity of making decisions based on perceived stimuli and interactions between others agents to realize common or conflicting goals [2]. The agent-oriented paradigm is appropriate for implementing AmI systems because agents can be proactive, have social abilities and autonomy, and it is capable of learning from its past experiences [3].

In a common architectural approach to implement AmI systems and MAS, many sensors and actuators are managed by different microcontrollers, while in a more external layer, generic services for accessing sensors and actuators are implemented and provided for cognitive applications running on a top layer. Thus, is important to have an architecture able of controlling microcontrollers supported by an Agent-Oriented Program Language (AOPL) responsible for reasoning. Particularly, Jason [4] is a framework to develop MAS widely used in the agent community for programming cognitive systems interfacing hardware of the same type [5].

An extension of Jason named ARGO aims to facilitate the use of hardware devices by intelligent agents independently of the domain of the solution [6]. It means that it is possible to develop MAS where the software layer is not tied to the hardware layer and it can be used in any domain. Since the hardware choice in the design of the solution is limited since it is only possible to use ATMEGA microcontrollers, ARGO employs a generic communication interface between the AOPL and the microcontroller [7], and nothing prevents the development of a communication interface for other microcontrollers.

Therefore, the objective of this paper is to present a layered architecture for designing MAS capable of adopting different microcontrollers where all layers are independent from each other. For this, we adopt Javino as communication interface between microcontrollers and the software layer, and we propose Javic for interfacing the software layer with PIC or other C based microcontroller. The PIC was chosen since it is often used in industrial applications because of its reliability. Besides, the Jason framework and ARGO agents were used as the cognitive reasoning in the software layer.

Then, we present as proof-of-concept a small-scale prototype of a smart home for temperature control and a doorbell system for helping the hearing impaired to identify if there is someone in front of the door. In order to evaluate the MAS and the prototype, some performance tests were executed taking into account parameters such as the number of agents and controllers, the agent's reasoning and the amount of environmental perception, to explore different implementation strategies of AmI System development supported by MAS.

Our contributions are: (i) the use of different kind of microcontrollers in the same MAS ; and (ii) a communication interface for working along with ARGO to program MAS for controlling PIC microcontrollers. This paper is structured as follows. In Section 2, we present some related work; In Section 3, the architecture is presented; In Section 4, the Smart Home architecture and its prototype are discussed; and Conclusion and future works are presented in Section 5.

II. Related Work

Some works exploit existent architectures and middleware to facilitate the connection between hardware and software. Some works try to embed the MAS into hardware platforms to provide real autonomy and other use a central processing unit for controlling the hardware from a distance.

In [8], it is presented an unmanned ground vehicle controlled by a MAS hosted in a computer and programmed in Jason. The agents can control the vehicle using commands that are sent to the hardware using radio transmitters on both sides

DOI reference number: 10.18293/SEKE2018-211

(computer and microcontroller), but the MAS only communicates with a single type of microcontroller. In addition, in all cited works, the MAS is not embedded with the hardware.

In [7], a platform for embedding MAS programmed in Jason using a Raspberry Pi board is presented. The MAS controls the functions of a vehicle using external actions. The agents are not able of controlling devices directly, becoming dependent on the simulated environment programmed in Java. Besides, it only communicates with a single type of microcontroller.

Applying MAS in AmI is not a new topic and several proposals integrating devices have already been presented in simulated smart homes such as [9]. However, it does not use AOPL, the agents do not use a cognitive model, and the implementation is tied to the solution. In [10], it is proposed a model for supporting residential accidents using embedded agents and Arduino. For each Arduino, there is a specific agent and a high-level agent replicated in case the former is not capable of processing. In this work, an ARGO agent controls many devices. An embedded approach in real time using Jade for programming MAS is proposed by [11]. The architecture maps each sensor and actuator in the high-level language, and there are six types of generic agents. In this paper, the sensors and actuators are connected to controllers and only ARGO agents can manage such devices.

III. The Architecture

In this section, we propose a layered architecture for the development of MAS using an AOPL (responsible for the MAS) and a separated and heterogeneous hardware layer, with several microcontrollers, actuators and sensors (a heterogeneous architecture is able of employing different types of microcontrollers in the same solution). To establish communication between both independent layers, a third layer is used as middleware that is responsible for the exchanging data, through serial communication, between the software and the hardware. Using this middleware, the hardware is programmed to send all the perception and execute requests arriving from the MAS. Already in the software layer, agents are designed to interact with the infrastructure available sending actions to the hardware layer and receiving perceptions coming from sensor. In our architecture, the MAS is independent from the hardware and can be modified or changed if it is desirable.

So, the proposed architecture could employ microcontrollers of different types, each of them controlling sensors and actuators in the hardware layer, and a central core responsible for the deliberation based on information perceived by sensors in the software layer. It is used the Jason and ARGO agents for controlling hardware devices. The use of Javino and Javic middleware along with Jason and ARGO agents provides a platform, which supports the developer to deploy AmI systems without concerns about integration issues between hardware and software because of the independence between layers. All perceptions are directly processed by the MAS without intervention of the designer. Besides, the capability of using different microcontrollers in the same project controlled by a MAS is the main contribution of using the architecture.

A. The Hardware Layer Implementation

In the hardware layer of the architecture, it can be employed different types of microcontrollers. For this, libraries for capturing data from sensors and sending them to the software layer must be adopted. In this section, we discuss Javino and present the Javic library for interfacing PIC microcontrollers.

1) The Javino [7]: Javino is a communication interface used to exchange messages between microcontrollers and programming languages using serial communication. Its main benefit is to ensure that the receiver will not accept messages with errors. The Javino consists of two libraries: one on the hardware-side (microcontroller), and another one on the software-side (programming language).

When using Javino, the message follows a structure with three fields: **Preamble**, composed by 2 bytes of a fixed value to identify the message; **Size** that has 1 byte used to inform the size of the message sent and; the **Message** (up to 256 characters) to be sent. The two firsts fields are both used to identify errors that can occur because of information loss during the message transmission. In this work, Javino is used on the hardware layer for interconnecting ATMEGA microcontrollers or any other platform that uses the Arduino IDE such as Galileo, Galileo Gen 2, and NodeMCU. In this case, Javino is responsible for gathering perceptions from all sensors connected to the microcontroller, and it sends them for the software layer. The designer of the system must be responsible for programming the sensors and actuators stimuli based on messages received from the communication layer.

2) The Javic: Javic implements the same protocol as Javino. The Javic is a C-based library for PIC microcontrollers. Depending on the type of the PIC, the amount of available memory can interfere in the functioning of the library, because it was developed to work in PIC with at least 256 bytes of RAM because the size of the message is up to 256 bytes. The methods implemented for Javic and Javino are:

- **sendMsg(String msg)**: Sends a message to the software layer using serial communication;
- **availableMsg()**: Checks if exists messages coming from the software layer, returning a boolean value informing whether there is a message available or not;
- **getMsg()**: If there is a message available, it is used to get the request information sent by the software to perform some action using actuators or to gather perceptions.

When the software side library needs to send a message to the other side, it is necessary to inform the port where the target device is connected. Including Javino and Javic, there are 3 libraries: one for the software side that uses the Java language, and two for the hardware side, one for ATMEGA, and another one for PIC microcontrollers.

B. The Software Layer Implementation

In this section, we discuss the technologies used in this work for the development of the software side of the architecture. Jason [4] is a framework used to build cognitive MAS, and when used together with the customized architecture of agents

named ARGO [5], it is possible to develop solutions using actuators and sensors that can interact with the real world.

Jason is a framework that has an interpreter in Java of AgentSpeak for the development of cognitive agents using the BDI. The BDI contains three basic constructions: "Beliefs" (information considered to be truth by the agent acquired internally, with other agents or with the environment), "desires" (agent's motivation to perform determined goal), and "intentions" (actions that the agent is compromised to execute) [4].

ARGO is a customized architecture of Jason agents for enabling the programming of agents capable of interacting with platforms of prototyping. The ARGO allows the intermediation between the cognitive agents and a real environment (using microcontrollers) through the Javino. A MAS can be composed of traditional Jason agents and ARGO agents working simultaneously. The Jason agents can perform plans and actions only in software level and communicate with other agents in the system (including ARGO agents). An ARGO agent is a traditional agent with additional features, such as the ability to communicate with the physical environment, perceive it, act upon it, and filter information perceived from sensors connected to microcontrollers. For this, ARGO has five internal actions to be used at runtime: (i) the action `.port(Port)`, where the agent chooses which device to control selecting the serial port where the device is connected (e.g. .port (com8)); (ii) the action `.percepts (open or block)`, where it is defined if the agent blocks or releases the flow of perceptions from the controller; (iii) the action `.limit (milliseconds)`, which defines for how long the environment should be perceived; (iv) the action `.act (message)`, which sends a message through the serial port to execute an action using an actuator and; (v) the action `.filter (XML)`, which selects the XML file responsible for filtering perceptions.

IV. The Smart Home Propotype

In this section, we present a Smart Home architecture based on the proposed architecture using the Jason and the ARGO and containing several controllers with sensors (temperature) and actuators (LED lights). In Figure 1, we propose one possible architecture for a Smart Home prototype developed in wood, which has six rooms each one controlled by a microcontroller (three ATMEGA328 and three PIC). Four rooms have light sensors (LDR) and LEDs; one room has a temperature sensor (LM35), an air-conditioner(Peltier), a heater(Peltier) and LEDs and; the last room has the bell door button, the bell's sound emitter (buzzer), the door motor and a LED. For each room, an ARGO agent is responsible for the cognitive management of sensors and actuators. The ARGO agents have the same communication skills as a traditional Jason agent, where an ARGO agent can communicate with another ARGO agent or with a traditional agent.

The methodology employed in the development of the smart home takes into account three layers where interventions are required: the interconnection of hardware devices, where sensors and actuators should be connected to the controllers in the desired room; the microcontroller programming, where all the activation functions of the actuators must be programmed in the controllers in response to serial port stimuli, and; the MAS's creation, where the perceptions coming from the sensors must be prepared to take into consideration the format expected by the MAS. The perceptions are sent to the agent every time an ARGO agent performs its reasoning cycle. Finally, the MAS must be independently programmed to the hardware, taking into account only the actions that must be performed in hardware. If it is necessary to change the device, there is no need to recode the MAS, but the agent should point to the proper serial port it desires to control, and the actions messages to be executed must be available on the new device.

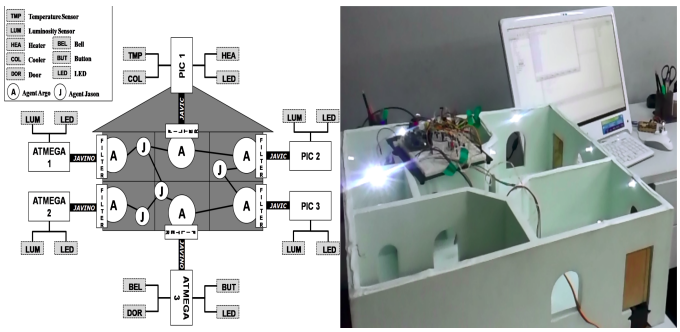

Fig. 1. The Smart home architecture and prototype.

A. Performance Tests

In order to test the applicability and to identify strategies for using ARGO agents in the AmI domain, we executed performance tests by performing a circuit of activating and deactivating of LEDs (the agent turns on all the LEDs from room to room and then turn them off). In the Jason, an event is generated for each perception received from the environment. This feature can lead to delays in the reasoning and execution of actions in situations that require fast responses. To deal with this situation, an agent can vary the use of the internal action that blocks and release the perceptual flow from sensors. Thus, tests were performed and the execution time of the activation circuit was measured employing from 1 to 6 microcontrollers being controlled by 1 to 6 agents in the MAS (36 possibilities) and using three different strategies:

1) **opened:** the agents open the flow of perceptions at the beginning of the execution to continuously perceive the environment until the end of the MAS execution without blocks its perceptions.
2) **just once:** the agent opens the flow of perceptions according to the need to execute each plain to search sensorial information and after that, it closes it again, acquiring just a few perceptions at the moment.
3) **lazy:** the agent open and close the flow of perceptions at the beginning of the first plan, repeat the operation at the last plan, and eventually when is necessary to modify controllers in the middle of the execution.

For each strategy, the tests were replicated 3 times totalizing 324 tests, of which 135 resulted in conflict because of the amount of ARGO agents was greater than microcontrollers employed (two or more agents can not use the same serial port at the same time). When there are conflicts by serial port competition, one solution is to implement a negotiation strategy to avoid that they do not use a device at the same time. For the remaining tests, when considering only the number of controllers managed by the MAS, it is noticed that **opened** strategy is the slowest of all because the number of perceptions processed is larger than the other strategies (this strategy never blocks its perceptions). The **lazy** strategy is slightly faster than **just once** strategy, but depending on the agent's programming or the domain, keeping the perception out of date for a period could cause some mistaken decisions.

Analyzing the results and the number of agents, we noticed that when one agent is responsible for controlling all devices, the execution time increases. The **opened** strategy is again the slowest due to the number of perceptions captured from sensors (they are updated in every reasoning cycle) while the **just once** and **lazy** strategy were faster because of the way perceptions were blocked. It is noteworthy that for all tests, the agent's codes were the same, differentiating only where the flow of perceptions was opened or closed. Figure 2 shows the scatter plot of the tests. For the next examples we decided that **just once** strategy should be used, it is slightly slower than **lazy** strategy, but it guarantees that the agents will have up-to-date perceptions of the sensors when compared with the former one. The **opened** strategy was not chosen because it was slower than the others. However, its characteristics always keep up-to-date information from sensors.

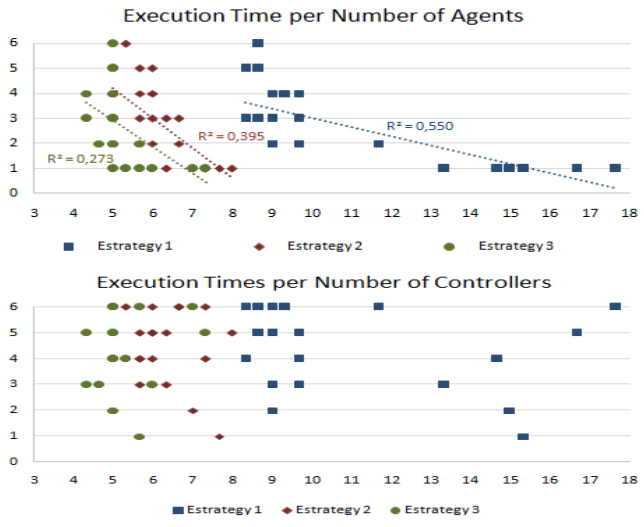

Fig. 2. The scatter plot of the number of agents employed (left) and the scatter plot of the number of controllers employed (right).

V. Conclusions

This work presented a layered architecture for programming MAS which uses hardware devices, it uses Jason framework and ARGO in the software layer, and it also uses libraries for communication to different microcontrollers. Then, we developed a specific library for integrating PIC microcontrollers with Jason framework allowing the design and construction of MAS and prototypes using both PIC and ATMEGA. This heterogeneous characteristic is an important issue for developing AmI systems because it is possible to use devices accordingly to the requirements of the designed MAS. Most of the platforms in the literature are directed to one kind of technology, or it ties to a particular domain. Our proposed architecture does not tie the design of the system to a particular domain, and it allows the use of several types of microcontrollers.

In AmI, we aim to use this heterogeneous characteristic of the proposed architecture to develop smart and proactive environments. So, we presented as proof-of-concept, a smart home example showing that is possible to use Jason to develop such kind of solutions. Besides, an analysis of three strategies for MAS implementation were presented to identify the best strategy to obtain quicker and efficient responses from agents. The results show that the delays generated are acceptable depending on the domain and strategy applied. As future work, it is necessary to test the approach in a real environment using complex scenarios with a high number of sensors.

References

[1] W. Weber, J. Rabaey, and E. Aarts, *Ambient Intelligence*. Springer, 2005.

[2] M. Wooldridge, *An Introduction to MultiAgent Systems*. Wiley, 2009.

[3] C. Maciel, P. C. de Souza, J. Viterbo, F. F. Mendes, and A. El Fallah Seghrouchni, *A Multi-agent Architecture to Support Ubiquitous Applications in Smart Environments*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2015, pp. 106–116.

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

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

[6] C. E. Pantoja and J. Viterbo, "Prototyping ubiquitous multi-agent systems: A generic domain approach with jason," in *Advances in Practical Applications of Cyber-Physical Multi-Agent Systems: The PAAMS Collection: 15th International Conference, PAAMS 2017, Porto, Portugal, June 21-23, 2017, Proceedings*, Y. Demazeau, P. Davidsson, J. Bajo, and Z. Vale, Eds. Springer International Publishing, 2017, pp. 342–345.

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

[8] R. S. Barros, V. H. Heringer, N. M. Lazarin, C. E. Pantoja, and L. M. Moraes, "An agent-oriented ground vehicle's automation using Jason framework," in 6^{th} *International Conference on Agents and Artificial Intelligence*, 2014, pp. 261–266.

[9] K.-I. Benta, A. Hoszu, L. Văcariu, and O. Creţ, "Agent based smart house platform with affective control," in *Proceedings of the 2009 Euro American Conference on Telematics and Information Systems: New Opportunities to increase Digital Citizenship*. ACM, 2009, p. 18.

[10] G. Villarrubia, J. F. De Paz, J. Bajo, and J. M. Corchado, "Ambient agents: embedded agents for remote control and monitoring using the pangea platform," *Sensors*, vol. 14, no. 8, pp. 13 955–13 979, 2014.

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