A Middleware for Using PIC Microcontrollers and Jason Framework for Programming Multi-Agent Systems

João Victor Guinelli
Centro Federal de Educação Tecnológica Celso Suckow da Fonseca - Campus Nova Friburgo
Av. Gov. Roberto Silveira 1900
Nova Friburgo-RJ, Brasil
CEP 28.635-000
joao.silva@cefet-rj.br

Carlos Eduardo Pantoja
Centro Federal de Educação Tecnológica Celso Suckow da Fonseca - Campus Maria da Graça
Rua Miguel Ângelo 96 -
Maria da Graça-RJ, Brasil
CEP 20.785-220
pantoja@cefet-rj.br

ABSTRACT

This paper presents a middleware using PIC microcontrollers for programming robotic agents controlled by a Multi-Agent Systems alongside with Jason Framework. The middleware is a hardware side library developed in C for PIC and is based on the Javino protocol. The proposed middlewarel allows a communication between hardware and software and aims to be used together with Javino for software side. A simple example is presented to show the basic functioning of a robot architecture using the middleware.

Keywords

Middleware; Multi-Agent Systems; Robotics

1. INTRODUCTION

Nowadays, a robot can be understood as an intelligent agent, which is able to perceive, reason and act into a real environment [7]. Similarly, agents can be defined as virtual or physical entities that are capable of perceiving or acting into an environment, where they can be situated. So, in robotics it is reasonable to use Multi-agent Systems (MAS) approach and agent-oriented programming languages. Furthermore, MAS approach can be used for complex and distributed problems.

However, embedding BDI agents in robotics is not a trivial task since their reasoning cycle can generate undesirable delays into the robot execution in real environments [8]. Some works tries to embed MAS in platforms using BDI frameworks such as [1] and [4]. In [1] it is proposed a robotic platform that moves from one point to another based on the GPS position. The prototype uses the JASON framework [3] but the reasoning agent is not embedded in the robot vehicle. On the other hand, [4] extends the Jason in order to control Lego robots that are able to follow lines on the floor and avoid obstacles. However, the extension does not allow embedding MAS because Lego platform does not have sufficient space for a MAS system.

In [5] it is presented a middleware that is responsible for helping the communication between low-level hardware and high-level programming languages and it is used alongside with Jason for programming embedded MAS. However, the Javino middleware is dedicated only to ATMEGA microcontrollers. Therefore, the objective of this paper is to present the Javic middleware for communication between PIC microcontrollers and the Jason framework. The Javic uses a library developed for the PIC side and it the JAVA side library of Javino. This middleware aims to provide the usage of PIC microcontrollers in embedded MAS since PIC is one of the most used microcontrollers in industry and automation applications.

This paper is structured as follows: section 6 analyzes some related works; section 3 presents the robotic-agent architecture; in section 4 the Javic middleware is explained; section 5 shows how to embed Jason and use Javic middleware to communicate with PIC microcontrollers; in section 6 a case study is presented; and the section 7 presents the conclusions and future works.

2. RELATED WORK

In this section it is discussed some related works, which also make use of multi-agent programming languages to control robotic platforms. In [1], the authors propose an integration between Jason and Arduino in order to control grounded-vehicles. To implement the communication between the Arduino board and Jason's, the authors used the RxTx library. Using that approach, it is possible to use Jason to operate any kind of vehicle, however, Jason is not truly embed since it is running in a external computer that communicates itself with the vehicle through transmitters and receivers. Besides that, no protocol is used to perform the communication between the software and hardware-side, what can cause failures on the system since data loss and interferences are not handled. Differently, this work can truly embed Jason in a Raspberry board, and we use Javic, that implements the same protocol implemented by Javino, to perform the communication between the software and the hardware-side.

In [4] the author extend Jason framework, through the implementation of internal actions, to make possible the communication between physical agents and the Lego Mindstorm NXT toolkit. That communication is performed using Bluetooth and LeJOS as the middleware between Jason and Lego. However, according to the author, this communication is kind of slow, and and the robotic platform is tied to Lego Mindstorm NXT. On the other hand, the agent-robotic architecture presented in this paper can be used in any robotic platform that uses Java-based languages in the software-side, and ATMEGA or PIC microcontrolers in the hardware-side.

3. ROBOTIC-AGENT ARCHITECTURE

The development of the Javic middleware was motivated mainly by its possible application in robotic-agents using embedded MAS, which can be able to control different devices connected to different microcontrollers. In fact, Javic allows the design of robotic-agents using PIC microcontrollers while Javino uses the ATMEGA microcontrollers. Despite of the possibility of a selection between these microcontrollers, both hardware-side libraries communicate with the same software-side library, allowing the usage of both microcontrollers into the same robot. In order to clarify this approach, we propose a robotic-agent architecture for using along with any microcontroller or agent-oriented programming language with components adherent to the protocol implemented by Javino and Javic.

A robotic-agent designed using the proposed architecture contains microcontrollers of different types, each of them controlling sensors and actuators, and a central core responsible for the reasoning, based on information perceived by sensors, and it is able of controlling actuators. For interconnecting the hardware and the software of the robot, it is necessary to use middleware.

In Figure 1, it is possible to see the architecture of a robot using different microcontrollers, such as PIC, ATMEGA, INTEL or any other, as long as this hardware uses a middleware compliant to the same protocol used in Javino and Javic when communicating. For each microcontroller there are several actuators and sensors connected. The hardware-side libraries used are Javic for PIC and Javino for ATMEGA. However the architecture is extensible, so it is possible to implement a hardware-side library for INTEL8051 for example (named Javintel) or any other library for other microcontroller.

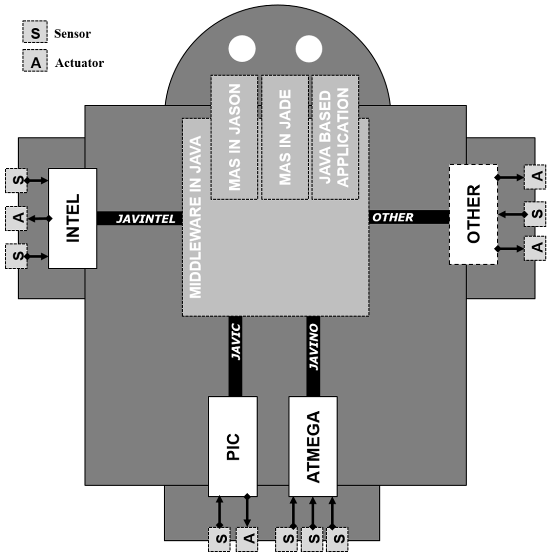

Figure 1: A robotic agent architecture using Javino and Javic middleware.

In the context of software-agents, the core of the robot needs to execute an application programed in Java or any other framework based on that language, such as MAS in Jason or JADE [2]. However, if another programming language or framework were employed, it is necessary to develop the software-side of the protocol of Javino. The Javino software-side library is responsible for gathering all the perceptions from the microcontrollers and, after that, these perceptions has to be processed as beliefs depending on the MAS framework chosen. Based on the beliefs processing, the MAS can control the actuators through Javino library, which communicates with the microcontroller sending the actions to be performed in real world.

Therefore, depending on the strategy chosen for the implementation of MAS, the agent may exchange information with only one microcontroller at time or be programmed to gathers all perceptions and process them all together. This means that, for example, importing Javino into the Jason's reasoning cycle, when the agent is perceiving the real world environment, it will catch only the perceptions that came from the microcontroller (in the serial port set on Javino) that is being used in that moment. The same thing occurs when it is necessary to manipulate an actuator, the agent only has access to the microcontroller connected to the selected in Javino in that moment, but the agent can use Javino at real time to change the port where it is communicating.

4. JAVIC MIDDLEWARE

The Javic is a middleware that implements the same protocol as Javino in order to allow a communication using the serial port between a software in Java and PIC microcontrollers. The Javic middleware is composed of a library for PIC (hardware-side) and another one for Java (software-side).

For this, it was developed a library for PIC devices, which uses the C language. Depending on the type of the PIC, the amount of available memory can interfere on the functioning of this version of the library. The library was developed to work in PIC with at least 256 bytes of RAM memory because the size of the message is up to 256 bytes. It is possible to use the PIC 18F, 24F, 30F and 33F without restrictions. The PIC from family 16F should have at least 256 bytes of RAM and for family 12F it is not possible to use this version of Javic (however a version with a short version of Javic could be implemented if it is desirable). Table 1 shows a summary of PIC specifications.

Table 1: Amount of available memory of PIC families

Devices	RAM (bytes)	ROM (bytes)
12FXXX	64 – 128	256
16FXXX	128 – 256	256
18FXXX	256 – 512	256
24FXXX	24K – 96K	–
30FXXX	512 – 8K	1024 – 4092
33FXXX	8K – 30K	–

As the Javic is an implementation of the protocol for hardware-side, in order to maintain a pattern between the previous existent implementation and possible next ones, the methods presents in the Javic are the same present in the Javino for ATMEGA microcontrollers.

In the software-side of the middleware, it is used the Javino library for Java since it is not bound to the type of the microcontroller. It uses serial communication for exchanging messages. Then, when the hardware-side library sends a message (using PIC or ATMEGA), the software-side library receives and verifies the correctness of the content. If there is no data loss during the transmission, the message is delivered. 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 (it is not possible to send the same message for all the devices at the same time).

With the development of the Javic, nowadays there is tree libraries that implement the Javino protocol: one for the software-side that uses the Java language, and others two for the hardware-side, one for ATMEGA, and another one for PIC microcontrollers.

5. EMBEDDING JASON + JAVIC

For embedding any MAS in robots, it is necessary to program and interfere in different abstraction layers using several hardware (sensors, actuators, microcontrollers, boards and operational systems) and software (middleware and programming languages). These layers should be independent depending of the robot architecture and project. The modularity of the layers provides a maintainable and scalable architecture with high cohesion and low coupling. Besides, it should provide a fault tolerance mechanism for avoiding a total block of the MAS in case of malfunctioning of a component in real time constraints.

For using Jason and Javino, or Javic, for embedding MAS into any robotic platform (Figure 3), it is necessary to interfere in four layers accordingly to [5]:

1. **Hardware**. Firstly, this layer represents the physical robot, where all the components are interconnected. The sensors and actuators are plugged on the microcontrollers that are connected on serial ports of the selected board. The board should be able to host an operational system where the MAS will run. The selected board was the Raspberry Pi. Therefore, the operational system of the selected board was adapted to automatically run the MAS as soon as it starts;

2. **Microcontroller.** The microcontroller is responsible for controlling the sensing and acting into the real world. In this layer, all desirable actions should be programmed as procedures that are called in return of received serial messages. Besides, the perceptions has to be sent through serial port. The microcontroller used was the PIC 18F4520;

3. **Middleware.** The middleware is responsible for exchanging the messages between the microcontroller layer and the agent layer. For this, the middleware should be imported in both layers (microcontroller and agent). Javino and Javic were used as the middleware, Javino in the software-side and Javic in the hardware-side.

4. **Agent.** In this layer, the MAS is programmed. The agent-oriented programming language used is Jason, where in the simulated environment the Javino library is responsible for exchanging messages with the microcontroller.

6. PRELIMINARY TEST

In order to test the proposed architecture, we present a basic test using Javic. We will use Jason Framework to develop the MAS because it is based on BDI, which is a cognitive model is responsible for the reasoning of the system. Our purpose with this section is to prove that the architecture can work properly using the middleware for communicating with a MAS. So, we live for further work a performance analysis and tests in a complex scenario.

Here, we show an example using PIC 18F4520 connected to an ultrasonic sensor and a led as actuator. We choose this PIC version because of the amount of memory available for mounting Javic's messages. In this case, the agent should turn on the light when it realizes that an obstacle is approaching. In this example, the PIC microcontroller was programmed based on methods that can be activated when a serial message arrive from an agent. The same occurs for the perceptions, once the agent requests them based on its needs. However, it is possible to program the microcontrollers to send the perceptions from time to time. The program code of the microcontrollers can be seen as follows (we use C to represent the PIC program code):

```
while(true){
  receivedMessage = javic.getMsg();
  if(receivedMessage="getPercepts")
    getPercepts();

  if(receivedMessage="turnOn")
    turnOn();

  if(receivedMessage="turnOff")
    turnOff();
}

void getPercepts() {
  //code for getting the distance from
    ultrasonic sensors
}

void turnOn() {
  //code for turning on the led
}

void turnOff() {
  //code for turning off the led
}
```

In Jason, we optioned to use the simulated environment in Java to provide the external actions to be performed by the agent in real world (using actuators) and the perceptions processing from the microcontrollers. The environment code can be seen as follows:

```
if(action.toString().equals("refresh")){
  this.javino.port = "COM8";
  this.javino.sendMsg("getPercepts");
  if(this.jBridge.availablemsg()){
    this.msg = "dist(pic," + this.javino.
   getmsg() + ")";
    addPercept(Literal.parseLiteral(this.
   msg);
  }
}

if(action.toString().equals("lightOnPic")){
  this.javino.port = "COM8";
  this.javino.sendmsg("turnOn");
}

if(action.toString().equals("lightOffPic"))
    {
  this.javino.port = "COM8";
  this.javino.sendmsg("turnOff");
}
```

The agent uses Jason's external actions to get perceptions and to activate the actuators. The external action activates the software side of the middleware that sends a serial message to the PIC microcontroller which get the perceptions or execute an action. The agent code can be seen as follows:

```
+!moving: dist(pic, X) & value(20) & X>J <-
    refresh;
    lightOffPic;
    !moving.

+!moving: dist(pic, X) & value(20) & X<=J
      <-
    refresh;
    lightOnPic;
    !moving.
```

7. CONCLUSIONS

This paper presented the Javic middleware, which implements the same protocol as Javino and allow the communication between Java-based languages and PIC microcontrolers. For future works, we will develop a tiny version of Javic for PIC with RAM memory below 256 bytes, reducing the length of the message in the protocol to 128 bytes. Besides, we will also provide a library for Javintel middleware, in order to provide a communication between INTEL8051 microcontrollers and the Java language. Then, the architecture will provide a selection among several microcontrollers used at industry.

The use of embedded MAS in real-time scenarios arises several performance issues as the time of processing perceptions and the response time that the robotic should proper act. So, the architecture should be analyzed in a complex scenario case study with real-time constraints. We also intend to use ARGO [5] for testing the Javic middleware in embedded scenarios.

8. REFERENCES

[1] R. S. Barros, V. H. Heringer, C. E. Pantoja, N. M. Lazarin, and L. M. de Moraes. An agent-oriented ground vehicle's automation using jason framework. In *ICAART (2)*, pages 261–266, 2014.

[2] F. Bellifemine, G. Caire, and D. Greenwood. *Developing multi-agent systems with JADE*. Wiley series in agent technology. John Wiley, 2007.

[3] R. H. Bordini, J. F. Hübner, and M. Wooldridge. *Programming multi-agent systems in AgentSpeak using Jason*, volume 8. John Wiley & Sons, 2007.

[4] A. S. Jensen. Implementing lego agents using jason. *arXiv preprint arXiv:1010.0150*, 2010.

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

[6] C. E. Pantoja, M. F. Stabile Jr, N. M. Lazarin, and J. S. Sichman. Argo: A customized jason architecture for programming embedded robotic agents. *Third International Workshop on Engineering Multi-Agent Systems (EMAS 2016)*, 2016.

[7] S. Russel and P. Norvig. Inteligência artificial. *Editora Campus*, page 26, 2004.

[8] F. R. Santos and J. Hubner. Avaliação do uso de agentes no desenvolvimento de aplicações com veículos aéreos não-tripulados. 2015.