An IDE to Support the Development of Embedded Multi-Agent Systems

Vinicius Souza de Jesus[1 [0000−0002−4534−6078]],
Nilson Mori Lazarin[1,2 [0000−0002−4240−3997]],
Carlos Eduardo Pantoja[1,2 [0000−0002−7099−4974]],
Gleifer Vaz Alves[3 [0000−0002−5937−8193]],
Gabriel Ramos Alves de Lima[2 [0009−0009−3233−9408]], and
Jose Viterbo[1 [0000−0002−0339−6624]]

[1] Fluminense Federal University, Brazil
[2] Federal Center for Technological Education Celso Suckow da Fonseca, Brazil
[3] Federal University of Technology - Paraná, Brazil
{vsjesus, viterbo}@id.uff.br, {nilson.lazarin, pantoja}@cefet-rj.br,
gabriel.ramos@aluno.cefet-rj.br, gleifer@utfpr.edu.br

Abstract. Embedded MAS development requires knowledge in different areas, such as agent-oriented programming, object-oriented programming, low-level programming, and basic electronics concepts. The literature has a consolidated Embedded MAS development architecture divided into four layers: Reasoning, Serial, Firmware, and Hardware. However, one of the main difficulties that MAS designers face is the need to use and configure different *Integrated Development Environments* (IDE) and make several integrations to embed the MAS. Even using all these technologies, embedding and monitoring the Embedded MAS is done using physical wired connections, making them limited and impracticable depending on the application. Therefore, this work aims to present an IDE to develop Embedded MAS that centralizes the entire development in a single IDE with all required integrations and configurations done. Moreover, the embedding and monitoring MAS of the IDE are done remotely without physical wired connections. Finally, aiming to show the IDE's applicability and functionalities, this work presents a case study set on a road junction with different Embedded MAS.

Keywords: Embedded MAS · Development · IDE.

1 Introduction

The Multi-Agent System (MAS) [19] comprises autonomous, proactive agents with decision-making capacity, social abilities to interact with other agents and cooperate to achieve a common goal. A MAS can be applied in virtual or physical environments. The virtual environment is a controlled environment that allows the application of MAS for specific tests. However, the physical environment is unpredictable and can change at any time, forcing the agents to update the

information captured from the environment frequently to avoid acting based on outdated information. For applying a MAS in a physical environment, the agents use several sensors and actuators grouped in a physical device to capture information and act/modify this environment. When a MAS is applied to a physical device can be classified as an Embedded MAS [5].

Several architectures for developing an agent, such as SOAR [11] and Belief-Desire-Intentions (BDI) [6], represent the cognitive reasoning processes that enable decision-making. The SOAR focuses more on the information storage process, and the BDI on decision-making. This work uses the BDI regarding its agents' decision-making process. In this process, Beliefs are the information that an agent accepts as truth, including information acquired through interactions with other agents or perceptions of the environment. Desires represent an agent's motivation to achieve a specific goal. Finally, Intentions represent the agent's commitment to performing some actions to achieve those goals. [6].

As the BDI allows agents to make decisions based on their beliefs, which can be perceptions of the environment, this architecture is interesting to be applied in an unpredictable environment such as the physical one. For embedding a MAS in a physical environment, there is a consolidated architecture that is divided into four layers [13]: the Reasoning layer is responsible for cognition, and it usually is composed of MAS with BDI agents. the Serial layer for the interface between the MAS and the microcontrollers [12, 8]; the Firmware layer for the programs of microcontrollers; Finally, the Hardware layer is composed of sensors and actuators. Therefore, to develop an Embedded MAS, the MAS designer needs to deal with different areas of knowledge, such as agent-oriented programming to develop the MAS (Reasoning Layer), object-oriented programming to develop the middleware that allows the exchange of information between the MAS and the microcontroller (Serial layer), low-level programming to develop the firmware of the microcontroller (Firmware Layer), and electronics for operating the sensors and actuators (Hardware layer). Moreover, the MAS designer must use one Integrated Development Environment (IDE) per programming language involved in this development procedure and provide integration between all of them.

Some frameworks in the literature aim to assist in developing MAS supported by IDEs, such as the *Jason framework* [4] that has an *AgentSpeak* [14] language interpreter for programming BDI agents and can be incorporated into the *Eclipse IDE*. In addition, the *Visual IDE's* [7] work allows the programming of MAS in code blocks to facilitate the codification for MAS designers who do not dominate the agent-oriented language to develop MAS. Finally, the *ARGO agents* [13] is a customized agent architecture capable of controlling microcontrollers.

However, these works do not cover the entire Embedded MAS development architecture presented, providing support only for the Reasoning layer. Consequently, to obtain support for the other layers, it is necessary to use different IDEs, sometimes one IDE per layer. In addition, the embedding and monitoring process requires physical wired connections, making the embedded process limited for instance in devices like Unmanned Aerial Vehicles (UAV) and Remotely Operated Vehicles (ROVs). Besides, the monitoring process with these devices

in operation is impracticable. For example, the *UAV* is an aerial and unmanned vehicle, so its operation is controlled and monitored remotely since maintaining a physical wired connection limits the mobility and the maximum area of exploration to the length of the wire.

Therefore, the objective of this work is to present an IDE to support the development of Embedded MAS considering all architecture's layers that need programming (Reasoning, Serial, and Firmware) with all necessary integration and configuration between these elements. So, the MAS designer can develop and program in three different layers of the architecture using a single IDE. In addition, the IDE allows to embed MAS and monitor their agents' minds (all Beliefs, Desires, and Intentions of each agent) remotely and wireless.

For this, the *Jason framework* was used to develop MAS (Reasoning Layer), as it is consolidated in the literature. For the Serial layer, the *Javino* [12] is the serial interface used, allowing messages to be exchanged with guaranteed content integrity. And in the Firmware layer, the *Arduino Command Line Interface (CLI)* was used to compile, upload and deploy the microcontroller program, allowing it to capture and send commands to the sensors and actuators. Finally, the integration and configuration of all these technologies were made and incorporated into a single IDE named *Cognitive Hardware on Network - Integrated Development Environment (ChonIDE)*.

To show the applicability of *ChonIDE*'s functionalities, a case study was elaborated and set on a road junction traffic scenario. The scenario is composed of an autonomous vehicle and a traffic light where the autonomous vehicle must be able to perceive the colors of the traffic light signal and, using its cognitive process, decide to move or stop the vehicle based on the colors of the traffic light. And the traffic light must change the colors according to the current traffic laws.

The main contributions of this paper are: **i.** allows the development and monitoring of the Embedded MAS using a single IDE, with all the required integrations of the different technologies made; **ii.** enables the development and monitoring of the Embedded MAS remotely without using a physical wired connection, which allows applying MAS in devices such as *ROVs* and *UAVs* that were previously done in a limited way; **iii.** assists in facilitating the Embedded MAS development by allowing MAS designers to embed and monitoring MAS even if they do not know how to deal with all layers of the architecture since it already has the technologies integrated and configured into the *ChonIDE*.

This work is organized as follows: Section 2, presents the related work; in Section 3, *ChonIDE* is presented, its functionalities and interaction with the designer; Section 4 discusses the case study where Embedded MAS is developed on a road junction scenario; finally, in Section 5, the final considerations and future works are presented.

2 Related Works

Considering the Embedded MAS development with agents capable of proactively and autonomously making decisions, some works in the agents' literature

implement the BDI architecture and have mechanisms to enable embedding MAS. Two frameworks stand out with several solutions to embed MAS and are part of many works in the literature, such as *ARGO agents* [13], *Physical Artifacts* [5], *Bioinspired protocols* [10], integration with the Robotic Operating System (ROS) [17]. The first framework is *Jason* (previously presented), and the second is *JaCaMo* [3] which combines three other frameworks: *Jason* is responsible for supporting the agents' development, *CArtAgO (Common ARTifact Infrastructure for AGents Open Environments)* [15] for the agents' interaction with the environment, and *Moise+* [9] for coordinating the organizational part of the agents in the MAS.

The *Jason framework* has a specific text editor for programming languages incorporated in its distribution called *jEdit*[4] that allows the programming of cognitive agents *(Jason + jEdit)*. In addition, the *Jason framework* has a plug-in for the *Eclipse IDE*[5] that provides tools for MAS development as a particular perspective that enables the creation, deletion, and import of MAS development projects and agents through the MAS interface in *Eclipse IDE (Jason + Eclipse)*.

However, these solutions only consider the Reasoning layer. As both use the *Jason framework*, it is possible to integrate the customized architecture of agents, named *ARGO agents*, into the presented solutions *(Jason + JEdit/Eclipse + ARGO)*. These *ARGO agents* can communicate and control microcontrollers, allowing them to capture information and act in the physical environment. For this, these agents have a serial interface called *Javino* in their constitution. *Javino* [12] has a process for verifying the integrity of messages and guaranteeing lossless delivery of information between the *ARGO agents* and microcontrollers.

As *Javino* is a generic serial interface that can communicate with different microcontrollers, the ARGO Agents (which has Javino in its architecture) can control different microcontrollers. Although several microcontrollers are available on the market, in this work, the Arduino microcontroller was used due to its applicability, versatility, and low monetary cost. For the *Arduino* microcontroller, several text editors and IDE support developing, compiling, uploading, and deploying its firmware. However, the *Arduino IDE* is a well-used tool because the producers built it themselves, it is free, and it has a web[6] and a desktop[7] version to install on the MAS device *(Arduino IDE)*. Moreover, the *Arduino IDE* allows the import of external libraries, which facilitates the integration with the *Javino* serial interface *(Arduino IDE + Javino)*.

Considering the *JaCaMo framework* integrates with several IDE, which stands out: the integration with the *Eclipse IDE*[8] that provides a whole perspective that allows MAS programming using all three frameworks (*Jason*, *CArtAgO*, and *Moise+*) jointly and simultaneous *(JaCaMo + Eclipse)*. In addition, there is an approach for agent-oriented visual programming that aims to enable MAS

[4] https://jason.sourceforge.net/doc/tutorials/getting-started/readme.html
[5] https://jason.sourceforge.net/mini-tutorial/eclipse-plugin/
[6] https://docs.arduino.cc/learn/starting-guide/the-arduino-web-editor
[7] https://www.arduino.cc/en/software
[8] https://jacamo.sourceforge.net/eclipseplugin/tutorial/

designers without programming experience but with specific knowledge of BDI agents can develop a MAS. Considering this approach, an IDE integrated with the *JaCaMo framework* named *Visual IDE* [7] was implemented, which has a visual programming system for agents that simplify the agent-oriented programming concepts using a blocks-based visual development environment *(JaCaMo + Visual IDE)*. Another IDE highlighted in the agents' area is a WEB IDE which uses the *JaCaMo* framework to allow interactive programming of MAS named *JaCaMo WEB* [2]. The interactive programming allows the MAS designer to modify the agents' source code at runtime without stopping and compiling the MAS. So, the MAS keeps running while the MAS designer changes the source code, maintaining MAS availability *(JaCaMo WEB)*.

Finally, *JaCaMo framework* has a plugin for *Visual Studio Code (VS Code)*[9]. Considering that the *JaCaMo framework* has *Jason* in its constitution, it is possible to integrate *ARGO agents* with *JaCaMo*. Since *VS Code* has a plugin for programming, uploading, and deploying firmware for *Arduino* microcontrollers, it is theoretically possible to centralize the development of an Embedded MAS in *VS Code* by performing all of the described integrations *(VS Code + ARGO + Arduino)*. However, to program the microcontroller firmware, it is necessary to have a physical wired connection between the MAS designer's computer (where the firmware is being developed) and the microcontroller, limiting the embedding and making it impracticable to monitor in some devices.

Aiming at the Embedded MAS development *ChonIDE* is a IDE for supporting the development of Embedded MAS with functionalities that include some layers (Reasoning, Serial, and Firmware) of the Embedded MAS development architecture, such as coding, compilation, uploading, and deployment of microcontroller firmware and agent source code. Furthermore, *ChonIDE* has a monitoring module that allows the designer to monitor the MAS log and the minds of all agents using *MindInspector* from *Jason framework*. All these functionalities can be done in remote solutions that do not require a physical wired connection., allowing embedding in devices such as *UAVs* and *ROVs* that require unrestricted mobility for locomotion. In Table 1, there are comparisons of the functionalities to develop Embedded MAS provided by *ChonIDE* and other IDE. These comparisons show which layers of the Embedded MAS architecture each IDE supports the development and whether the embedding and monitoring procedure is done remotely or not.

3 ChonIDE: The IDE For Embedded MAS

As seen previously, one of the main difficulties in the Embedded MAS development is dealing with concepts from different areas of knowledge (e.g., agent-oriented programming, object-oriented programming, and low-level programming with structured languages). In addition, the Embedded MAS designer needs to use one different IDE per programming language, making integrations and configurations between all of them.

[9] https://code.visualstudio.com/

Table 1: Comparison between *ChonIDE* and other IDEs.

	Reasoning Layer	Serial Layer	Firmware Layer	Wireless Embedding	Wireless Monitoring
Jason + *JEdit*	√	X	X	X	X
Jason + *Eclipse*	√	X	X	X	X
Jason + *JEdit* + *ARGO*	√	√	X	X	X
Jason + *Eclipse* + *ARGO*	√	√	X	X	X
Arduino IDE	X	X	√	X	X
Arduino IDE + *Javino*	X	√	√	X	X
JaCaMo + *Eclipse*	√	X	X	X	X
JaCaMo + *Visual IDE*	√	X	X	X	X
JaCaMo WEB	√	X	X	X	X
VS Code + *ARGO* + *Arduino*	√	√	√	X	X
ChonIDE	√	√	√	√	√

Thereat, aiming to facilitate and assist in the Embedded MAS development, this work presents a IDE named *Cognitive Hardware on Network - Integrated Development Environment (ChonIDE). ChonIDE* is a development environment that integrates technologies and tools to support the development of Embedded MAS, considering the three layers that require programming (e.g., Reasoning, Serial, and Firmware) in a single *WEB* platform. The *ChonIDE* works by establishing a network connection (wireless or wired) between the designer's computer and the physical device, so all functionalities can be performed remotely. Moreover, *ChonIDE's* main functionalities, such as embedding, starting, stopping, and monitoring the agents' minds, are activated by simply pressing buttons on the graphical interface. As the *ChonIDE* is a *WEB* platform, the MAS designer can develop an Embedded MAS using any device, such as a laptop, computer, or smartphone, without requiring configuration and integrations.

For the development of an Embedded MAS using *ChonIDE*, the MAS designer needs a physical device composed of sensors and actuators to interact with the environment, one or more microcontrollers to manage the operation of the sensors and actuators, and, finally, a single board computer (e.g., Raspberry) with a network adapter (can be wired or wireless) for connecting the physical device on the LAN to able *ChonIDE* to embed and monitor the MAS. With the physical device prototypes connected on the same LAN of *ChonIDE*, the MAS designer can embed and monitor the MAS operation of any prototype and send *Instructions* addressed to each one separately using the prototype identification on the LAN. The *Instructions* are to import a new MAS, start and stop them. Considering the microcontroller's firmware of the prototypes, the *Instructions* are to compile, upload, and deploy new firmware. The return of the execution of each of the Instructions is presented in a Log interface in *ChonIDE*.

Therefore, the information flow of the architecture of *ChonIDE* works as follows: the graphical interface of *ChonIDE* (*WEB* platform) is responsible for interactions with the MAS designer. These interactions are converted into the

Instructions (previously presented) to be executed on the physical device. The *Instructions* are sent via network communication (wireless or wired) to the physical device, and *ChonOS* (installed on the physical device) is responsible for interpreting and executing them. Figure 1 shows the interaction of *ChonIDE* with the physical prototype.

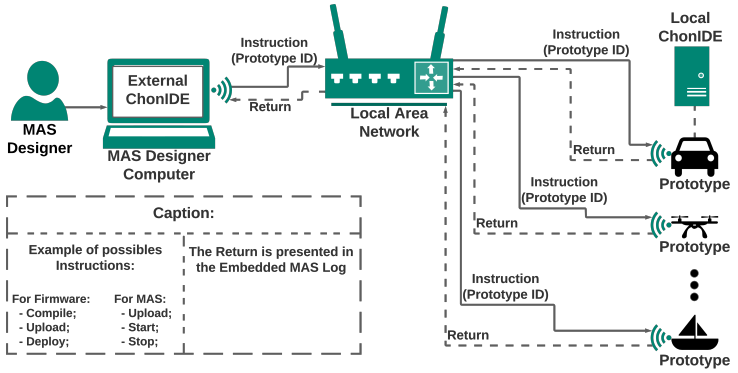


Fig. 1: Illustration of ChonIDE interactions.

To support the layers (Reasoning, Serial, and Firmware) that require programming, *ChonIDE* has a generic coder that allows programming in C (Firmware layer) and *AgentSpeak* (Reasoning layer). For the Serial layer, the integrations and configurations for the microcontroller of the physical device to communicate with the MAS agents are already done by the *Javino* serial communication interface. So, to control the microcontroller, the MAS designer only needs to use the *ARGO* agents with their internal actions (e.g., port, act, and percepts).

To facilitate the embedding of MAS using *ChonIDE*, this work uses a specialized GNU/Linux distribution, which has a set of services dedicated to embedding MAS. This distribution, named *Cognitive Hardware on Network - Operational System (ChonOS)*[10], has services such as network management, serial interface (using *Javino*), and a specific *Jason's* version for Embedded MAS. This *Jason's* version used by *ChonIDE*, in addition to the *ARGO agent*, has another customized agent architecture named Communicator Agents. The Communicator agents can interact with agents of other MAS using a network infrastructure. These agents allow the MAS designer using *ChonIDE* to create MAS capable of communicating and moving agents between different MAS. Moreover, the network manager service can check that no known networks are available; in this case, it changes the Wireless adapter configuration from client to access point mode to allow the MAS designer to connect with the physical prototype.

Like other IDEs, *ChonIDE* allows programming the source code of BDI agents, adding new agents, modifying their names, choosing their architecture

[10] http://chonos.sf.net

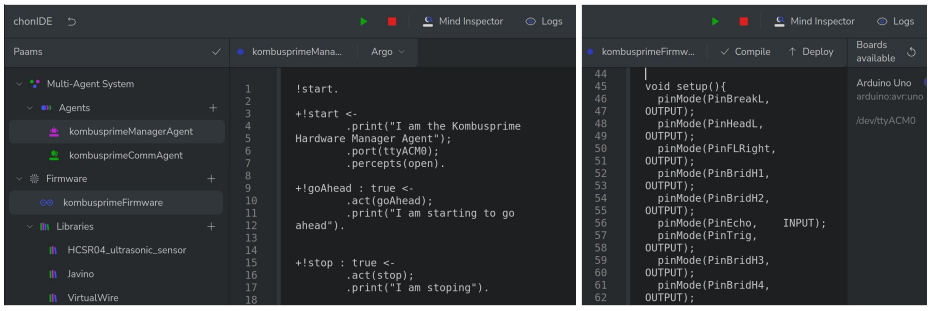
(e.g., Communicator, ARGO, or Standard Jason Architecture), and removing any MAS agents. Before starting the MAS execution, *ChonIDE* verifies if the developed source code is already in the physical device. If it is not found, *ChonIDE* uploads the MAS source code to the physical device and starts the MAS execution automatically. However, if the MAS source code is already found on the physical device, *ChonIDE* just starts the MAS execution. Moreover, the stop functionality stops the MAS execution, keeping the MAS source code on the physical device for another future execution. With all these functionalities, the MAS designer can develop a MAS remotely and embed it in a physical device without any wire connection.

To allow the MAS designer to monitor the execution of the MAS, *ChonIDE* has two functionalities: The first allows seeing the agents' minds at runtime with their beliefs, desires, and intentions provided by the *MindInspector* [4] of the *Jason* framework used by *ChonIDE*. The second is a *MAS log* interface that centralizes the logs and error messages generated by the MAS, allowing the MAS designer to identify and correct some errors or possible unwanted behavior of some agents. In addition, the MAS log also presents the information written on the console by the agents using internal actions such as *.print*. Like the embedding functionalities, these monitoring functionalities are enabled remotely without wired connections. Therefore, *ChonIDE* allows the application of Embedded MAS in physical devices such as the *UAV* that its operation is controlled and monitored remotely, which with previous technologies was done in a limited way since they use wire connections.

Finally, to support the development in the *Firmware* layer, the generic coder of the *ChonIDE* also allows the programming of the microcontroller's firmware. To assist in developing the firmware, *ChonIDE* has three functionalities: The first allows compiling the source code and seeing the return message from the compilation process for the MAS designer to see and correct any error that occurs with the developed source code. The second allows importing external libraries, which is essential when using some sensors or actuators that require a particular library to control them. The third allows uploading and deploying the firmware on the microcontroller and can also be done remotely without a physical wired connection. In Figure 2, ChonIDE's graphical interface is highlighted.

4 Case Study

To show the heterogeneity and applicability of *ChonIDE*, the case study is composed of MAS applied to physical devices that require mobility and remote monitoring (e.g., autonomous vehicle) and to a stationary device (e.g., traffic light). The proposed test scenario considers a road junction with an autonomous vehicle and semaphore prototypes. The scenario selection is mainly motivated by the following: i. Road junctions represent high risks of traffic accidents, as mentioned in [18]. ii. Previous work developed towards formalising and implementing autonomous vehicles endowed with road traffic rules, specifically road junction rules, as seen in [1] and [16].



(a) MAS' functionalities. (b) Firmware' functionalities.

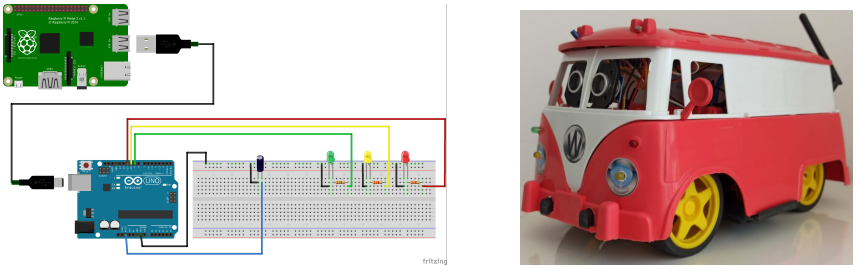
Fig. 2: ChonIDE's graphical interface.

This case study presents the feasibility of deploying what has been proposed in previous works (e.g., [1] and [16]) The agents (autonomous vehicles) with the corresponding knowledge of road traffic rules are embedded in a MAS. The traffic light must execute its change logic of the traffic signal color agreed, beginning with red, then changing to green, and finishing with yellow to restart, forming a cycle. Moreover, the traffic light has a mechanism to inform the autonomous vehicle of the current signal color. So, the autonomous vehicle should be able to receive the current traffic light and have knowledge about the traffic rules; it is agreed that in the red and yellow signal colors of the traffic lights, the vehicle must stop, and only at the green signal color should it go on. Based on this, the autonomous vehicle uses its decision ability process to respect the traffic rules and decide whether it should proceed through the junction or stop.

4.1 Deployment

The road junction scenario consists of two Embedded MAS applied in two physical prototypes (one for each). For implementing these Embedded MAS, it was used the Embedded MAS architecture presented, and the development started from the Hardware layer, passing through the Firmware layer, and going up until reaching the Reasoning layer. Therefore, the semaphore prototype was built using a single-board computer with a Wi-Fi network card (e.g., Raspberry PI) responsible for storing the MAS, a microcontroller (e.g., Arduino UNO) for interaction with the hardware components, and three LEDs on the green, red and yellow colors to represent traffic signals. Figure 3a has a schematic that shows how the traffic light prototype circuit was built.

The other physical prototype built was the autonomous vehicle prototype that is composed of a single-board computer with a Wi-Fi adapter (e.i. Banana PI), a light sensor called *Light Dependent Resistor* LDR; an HC-SR04 Ultrasonic Sensor for identifying obstacles; a set of five LEDs to light the vehicle; a buzzer; a set of *Direct Current* (DC) motors with an H bridge L298N for locomotion; a *Arduino UNO* microcontroller for controlling the hardware components. In Figure 3b, it is possible to see the autonomous vehicle prototype built.



(a) Schematic of semaphore prototype. (b) Autonomous vehicle prototype.

Fig. 3: Prototypes.

With the prototype built, *ChonIDE* was first used to develop the firmware of the prototype's microcontroller. After that, the MAS designer only needs to focus on developing the MAS. Considering this, the *ChonIDE* is used to develop and embed the MAS in the autonomous vehicle prototype. The MAS is composed of two agents: an *ARGO agent* controls the physical hardware of the vehicle, receiving perceptions of the environment through sensor measurements and acting with the activation and deactivation of the actuator such as motors, *Light-Emitting Diode* (LED), and buzzers. Finally, a *Communicator agent* requests and receives information about the current traffic signal color to the traffic light Embedded MAS.

Like the implementation of the autonomous vehicle Embedded MAS, the traffic light Embedded MAS also uses the *ChonIDE* to develop the microcontroller's firmware and the Embedded MAS. So the traffic light Embedded MAS is also implemented using two agents: an *ARGO agent* responsible for respecting the traffic laws by activating the traffic signals in the correct colors, knowing the current color of the signal, and informing the *Communicator agent*. The *Communicator agent* is responsible for receiving requests from the autonomous vehicle Embedded MAS and responding to the current signal color of the traffic light based on its *ARGO agent's* knowledge.

4.2 Reproducibility

The documentation and instructions for the reproduction of the presented traffic intersection test scenario can be found in the experiment repository of this work[11]. The procedure for reproducing the test scenario can be summarized into four main activities: construction of prototypes, installing *ChonOS*, firmware programming of the microcontrollers inserted in the prototypes, and finally, embedding and monitoring the MAS in the prototypes.

For the prototypes' construction, there is a step-by-step with the necessary electronic components and the diagrams for orientation in the prototypes' building in the experiment repository[11]. With the prototype built, the MAS

[11] https://chonide.chon.group/

designer must install the GNU/Linux *ChonOS* distribution on the prototypes' single-board computers following the installation manual available on the official *ChonOS* website[10]. With *ChonOS* installed on the prototypes, it is possible to access *ChonIDE* with all its functionalities. Given this, the MAS designer can use the *ChonIDE* resources to deploy the firmware of the prototypes' microcontrollers where the source codes are available in the experiment repository[11]. Finally, for the entire experiment's reproduction, the MAS designer must access the experiment repository to retrieve the source code of all MAS agents of the test scenario and use ChonIDE to embed and monitor the developed MAS[11].

5 Final Remarks

In this work, an Embedded MAS development architecture divided into four layers was used: The *Reasoning* layer; the *Serial* interface layer between the MAS and the physical device; the *Firmware* layer to program the microcontroller behavior, the *Hardware* layer composed of sensors and actuators. With this, it is possible to observe that one of the difficulties in embedding MAS is that it requires knowledge in different areas (Electronics, Low-Level Programming, Object-Oriented, and Agent-Oriented Programming).

Given this difficulty and others, such as the need to use different IDEs (usually one for each layer), this work presents a *WEB IDE* to help the development of an Embedded MAS named *ChonIDE*. *ChonIDE* has graphical interfaces and functionalities to support the three layers that require programming (Reasoning, Serial, and Firmware) of the architecture used, allowing the MAS designer to develop the Embedded MAS using only a single IDE, the *ChonIDE*. In addition, *ChonIDE's* embedding and monitoring procedure does not need to use physical wired connections with the physical device, which allows applying MAS to physical devices with remotely controlled and monitored operations, such as *ROVs* and *UAVs*, which with the previous technologies this sort of connection was somehow restricted.

As future work, it is intended to add to the coding interface a real-time syntax checker, the auto-completion function, keywords identification to produce a visual differentiation in writing these words, and functionalities to debug the Embedded MAS. Moreover, it is expected to provide integration with the *JaCaMo* framework to allow the programming of agents' interaction with the environment's Artifacts and the organizational part of the agents in the MAS. With the integration with JaCaMo done, it is expected that the integration with ROS will be facilitated and can be incorporated into *ChonIDE*. Finally, create an interface for Embedded MAS designers to send suggestions and comments to produce qualitative and quantitative feedback from *ChonIDE* to evaluate how much *ChonIDE* simplifies the development of Embedded MAS.

References

1. Alves, G.V., Dennis, L., Fisher, M.: Formalisation and implementation of road junction rules on an autonomous vehicle modelled as an agent. In: Formal Methods.

FM 2019 International Workshops. pp. 217–232. Springer (2020)
2. Amaral, C.J., Hübner, J.F.: Jacamo-web is on the fly: An interactive multi-agent system ide. In: Dennis, L.A., Bordini, R.H., Lespérance, Y. (eds.) Engineering Multi-Agent Systems. pp. 246–255. Springer International Publishing (2020)
3. Boissier, O., Bordini, R.H., Hübner, J.F., Ricci, A., Santi, A.: Multi-agent oriented programming with jacamo. Science of Computer Programming **78**(6), 747–761 (2013)
4. Bordini, R.H., Hübner, J.F., Wooldridge, M.: Programming Multi-Agent Systems in AgentSpeak using Jason. John Wiley & Sons Ltd (2007)
5. Brandão, F.C., Lima, M.A.T., Pantoja, C.E., Zahn, J., Filho, J.V.: Engineering approaches for programming agent-based iot objects using the resource management architecture. Sensors (Basel, Switzerland) **21** (2021)
6. Bratman, M.E.: Intention, Plans and Practical Reasoning. Cambridge Press (1987)
7. Burattini, S., Ricci, A., Mayer, S., Vachtsevanou, D., Lemee, J., Ciortea, A., Croatti, A.: Agent-oriented visual programming for the web of things (2022)
8. Guinelli, J.V., Pantoja, C.: A Middleware for Using PIC Microcontrollers and Jason Framework for Programming Multi-Agent Systems. In: Anais do Workshop de Pesquisa em Computação dos Campos Gerais WPCCG. vol. 1. Ponta Grossa (2016), http://www.wpccg.pro.br/volume001.html
9. Hübner, J.F., Boissier, O., Kitio, R., Ricci, A.: Instrumenting multi-agent organisations with organisational artifacts and agents. Autonomous agents and multi-agent systems **20**(3), 369–400 (2010)
10. Jesus, V.S., Pantoja, C.E., Manoel, F.C.P.B., Alves, G.V., Viterbo, J., Bezerra, E.: Bio-inspired protocols for embodied multi-agentsystems. In: 13th International Conference on Agents and Artificial Intelligence (ICAART 2021) (02 2021)
11. Laird, J.E., Newell, A., Rosenbloom, P.S.: Soar: An architecture for general intelligence. Artificial intelligence **33**(1), 1–64 (1987)
12. Lazarin, N.M., Pantoja, C.E.: A Robotic-agent Platform for Embedding Software Agents Using Raspberry Pi and Arduino Boards. In: Proceedings of the 9th Software Agents, Environments and Applications School (WESAAC). pp. 13–20. Niterói (2015)
13. Pantoja, C., Junior, M., Lazarin, N.M., Sichman, J.: ARGO: A Customized Jason Architecture for Programming Embedded Robotic Agents. In: Fourth International Workshop on Engineering Multi Agent Systems (EMAS 2016). Singapore (2016)
14. Rao, A.S.: AgentSpeak(L): BDI agents speak out in a logical computable language. In: de Velde, W.V., Perram, J.W. (eds.) Proceedings of the 7th European workshop on Modelling autonomous agents in a multi-agent world (MAAMAW'96). Lecture Notes in Artificial Intelligence, vol. 1038, pp. 42–55. Springer-Verlag, USA (1996)
15. Ricci, A., Viroli, M., Omicini, A.: Cartago: A framework for prototyping artifact-based environments in mas. In: International Workshop on Environments for Multi-Agent Systems. pp. 67–86. Springer (2006)
16. Schwammberger, M., Alves, G.V.: Extending Urban Multi-Lane Spatial Logic to Formalise Road Junction Rules. Electronic Proceedings in Theoretical Computer Science **348**, 1–19 (Oct 2021). https://doi.org/10.4204/EPTCS.348.1
17. Silva, G.R., Becker, L.B., Hübner, J.F.: Embedded architecture composed of cognitive agents and ros for programming intelligent robots. IFAC-PapersOnLine **53**(2), 10000–10005 (2020). https://doi.org/https://doi.org/10.1016/j.ifacol.2020.12.2718
18. Wada, Y., Asami, Y., Hino, K., Nishi, H., Shiode, S., Shiode, N.: Road Junction Configurations and the Severity of Traffic Accidents in Japan. Sustainability **15**(3), 2722 (Jan 2023). https://doi.org/10.3390/su15032722
19. Wooldridge, M.J.: Reasoning about rational agents. MIT press (2000)