Revista de Informática Teórica e Aplicada - RITA - ISSN 2175-2745
Vol. 29, Num. 01 (2022) 81-90
RESEARCH ARTICLE

Integrating Embedded Multiagent Systems with Urban Simulation Tools and IoT Applications

Integrando Sistemas Embarcados Multiagentes com Ferramentas de Simulação Urbana e Aplicações IoT

Lucas Fernando Souza de Castro[1]*, Fabian Cesar P. B. Manoel[2], Vinícius Souza de Jesus[2], Carlos Eduardo Pantoja[2], André Pinz Borges[3], Gleifer Vaz Alves[3]

Abstract: The smart city systems development connected to the Internet of Things (IoT) has been the goal of several works in the multi-agent system field. Nevertheless, just a few projects demonstrate how to deploy and make the connection among the employed systems. This paper proposes an approach towards the integration of a MAS through the JaCaMo framework plus an Urban Simulation Tool (SUMO), IoT applications (Node-RED, InfluxDB, and Grafana), and an IoT platform (Konker). The integration presented in this paper applies in a Smart Parking scenario with real features, where is shown the integration and the connection through all layers, from agent level to artifacts, including real environment and simulation, as well as IoT applications. In future works, we intend to establish a methodology that shows how to properly integrate these different applications regardless of the scenario and the used tools.
Keywords: Smart City — Multi-Agents — Urban Simulation — IoT

Resumo: O desenvolvimento de sistemas para cidades inteligentes conectados com Internet of Things (IoT) tem sido o foco de muitas pesquisas no âmbito de Sistemas Multi-Agentes (MAS). Contudo, poucos trabalhos mostram como realizar a implantação e conexão entre os softwares utilizados nestas diferentes áreas. Assim, este trabalho propõe uma abordagem de integração de um SMA, usando o framework JaCaMo juntamente com a ferramenta de Simulação Urbana (SUMO), aplicações IoT (Node-RED, InfluxDB e Grafana) e uma plataforma IoT (Konker). A integração apresentada aqui é aplicada em um cenário de Estacionamento Inteligente com característica realista, onde a integração e conexão de todos componentes envolvidos é descrita desde o nível dos agentes e artefatos, passando pelo nível do ambiente real (físico), pelo nível da simulação urbana, chegando as aplicações de IoT. Em trabalhos futuros, pretende-se elaborar uma metodologia que mostre como integrar essas diferentes aplicações independentemente dos cenários e das ferramentas utilizadas.
Palavras-Chave: Smart City — Multi-Agentes — Simulação Urbana — IoT

[1] *Institute of Computing, Universidade Estadual de Campinas (UNICAMP), Campinas - São Paulo, Brazil*
[2] *Centro Federal de Educação Tecnológica Celso Suckow da Fonseca (CEFET/RJ), Rio de Janeiro - Rio de Janeiro, Brazil*
[3] *Computer Science Graduate Program, Universidade Tecnológica Federal do Paraná (UTFPR), Curitiba - Paraná, Brazil*
*__Corresponding author__: lucas.castro@ic.unicamp.br, {souza.vdj, fabiancpbm}@gmail.com, pantoja@cefet-rj.br, {apborges, gleifer}@utfpr.edu.br
DOI: http://dx.doi.org/10.22456/2175-2745.110837 • **Received:** 20/01/2021 • **Accepted:** 29/08/2021

1. Introduction

In one of its reports, Cisco points to an estimated 50 billion devices for the *Internet of Things* (IoT) in operation as of 2022 [1]. Besides, there is a growing demand for computer systems that provides functionality and connectivity for these devices. The development of these systems must fulfil the dynamics of the scenarios where the devices for IoT are located, such as the smart city scenario.

A smart city could be defined as a city that provides integrated technologies to its citizens to have a quality of life [2]. This quality provided to the population can be diversified into different categories: governance, security, economy, environment, and mobility. This last one has been an area of attention for research and industrial investment in smart cities [3]. Thus, in face of such demands there must be a development of systems capable to support citizens' mobility.

Multi-Agent Systems (MAS) are candidates to provide the necessary support for citizens' mobility. It is achievable due to some features provided by these systems, for instance, autonomy, pro-activeness with cognitive reasoning, decision-making, and finally the capability to communicate with each other in order to achieve a common or conflicting

objective [4].

Through these features, the problems related to mobility in smart cities are managed proactively without interference or even knowledge of citizens, because the intelligent agents can perform this role for the user's benefit and negotiate with their peers to take relevant decisions at a specific time to any service. Whereas in a smart city there could be numerous devices capable of exchanging information, the IoT emerges as an essential technology in the implementation of such solutions since, by definition, IoT [5] is a set of devices interconnected by the Internet. Since MAS can be employed in a simulated (virtual) and physical way, it is possible to forecast devices embedded with MAS applied in IoT applications.

Observing the technological demand required by urban mobility, there is a requirement for a suitable system able to face intelligent and organizational situations. Moreover, it has to rely on the IoT to scale the problem resolution. Therefore, the objective of this work is to describe an approach for integrating different applications, involving MAS, Urban Simulation, Embedded Systems, and the IoT applications with the possibility to create solutions that are built under these technologies in order to solve issues of mobility, heterogeneity, and low coupling among the technologies. Hence, this integration aims to provide both a solution capable of being applied in different scenarios and applications (heterogeneity), and also a solution that provides the possibility of using only a part - subset - of the integration as a solution (low coupling among technologies).

Here, we use the JaCaMo framework [6] to implement our approach, which is a composition of three programming frameworks: Jason, CArtAgO, and Moise. Jason [7] interprets an agent-oriented language called *AgentSpeak* in Java for programming BDI agents (*Belief-Desire-Intention*). CArtAgO is based on the *Agents & Artifacts (A&A)* model, which allows the development of the MAS environments layer and integrates the agents and artifacts of a MAS [8]. Finally, Moise implements an organizational model for MAS based on grouping, behavior, and objectives [6].

The remainder of this paper is organized as follows. Section 2 presents the general approach to the integration of systems. Section 3 describes the integration between MAS agents and artifacts with the urban simulation. Afterward, the Section 4 details the integration of agents and artifacts with the physical environment. While, in Section 5, the integration of MAS with the IoT applications is presented. At last, Section 6 describes the final remarks.

2. Integration of Systems

This section presents a heterogeneous approach to integrate MAS, Urban Simulation, Embedded Systems, and the IoT layer. This approach describes how to develop an embedded MAS for IoT applications using the JaCaMo framework and integrating several tools to facilitate the application and visualization of such systems, including in different application domains. The approach employed here allows the detachment of a system structure into levels where the responsibilities of software, hardware, and IoT are programmed separately to facilitate integration depending on the solution that has been created and which levels will be employed. The system integration proposal is composed of four well-defined levels: Agents, Environment, IoT, and Simulation level, as illustrated in Figure 1.

The agent-level layer is responsible for creating MAS agents using the Jason language. Agents have two specific extensions: the first one is for interfacing hardware using ARGO [9] and the second one is for communicating and transporting agents using IoT. The agent-level layer connects with different parts of the system: i) the environment level through the artifacts provided by Cartago [8]; ii) the simulation level to enable the proper simulation of agents using SUMO urban simulator [10]; iii) the physical environment level directly through the ARGO extended agent; iv) and the IoT level used to scale-up the agent applications.

The environment's level divides between the artifacts (provided by Cartago) and the physical environment, where hardware platforms with sensors and actuators can interact with the physical world. Thus, it is possible to create *Simulated Artifacts*, those that maintain information only at the software level, and Physical Artifacts, which perform the Hardware interface through a Serial Interface. Both artifacts can co-exist in a solution developed using the proposed approach since the technologies used at each level are independent. Physical Artifacts can also relate to the IoT level via MQTT (*Message Queuing Telemetry Transport*) connections with *KonkerLabs* and the *Node-Red Application*.

At the IoT level, *ContextNet* is employed as a middleware for IoT that can deal with several characteristics of distributed systems such as connectivity, scalability, and communicability [11], for communication among agents. The proposed approach with communicator agents can connect to *ContextNet* to exchange messages or transport agents from a MAS embedded in one device to another MAS from a different device.

In this way, it is possible to create devices using the agent approach to act physically in a Cyber-Physical System and IoT applications. In this paper, a Device could be defined as a component composed of an embedded MAS using Jason and CArtAgO, capable of interfacing sensors and actuators connected to a microcontroller (ATMEGA, PIC, Arduino) and of connecting to an IoT network to exchange information.

3. Integration: Urban Simulation and MAS

The JaCaMo framework enables the use of three layers of development for MAS: agents, artifacts, and social. Besides, it can interconnect with different tools, for example, simulation, embedded systems, and IoT. Such capacity is due to the framework's flexibility by providing CArtAgO's environment artifacts. This section describes how JaCaMo and SUMO are connected.

In order to illustrate this integration, a MAS developed through JaCaMo Framework is used to assign and negotiate

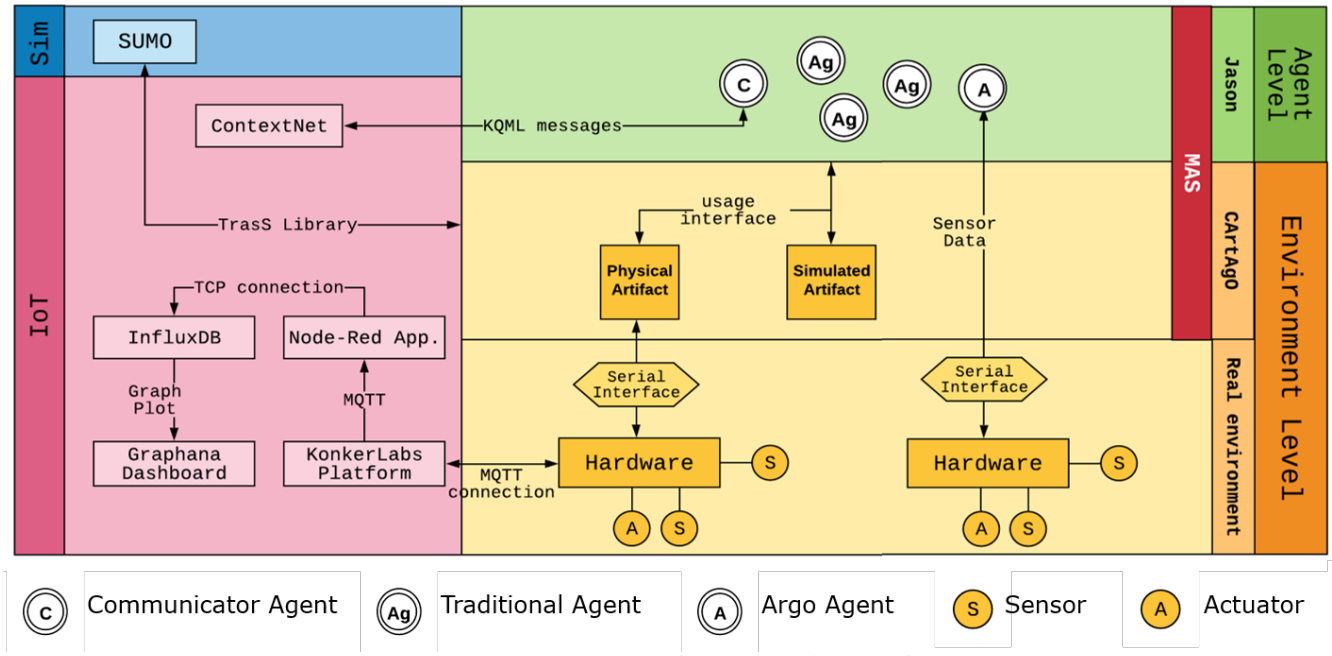

Figure 1. Systems Integration Approach: Overview

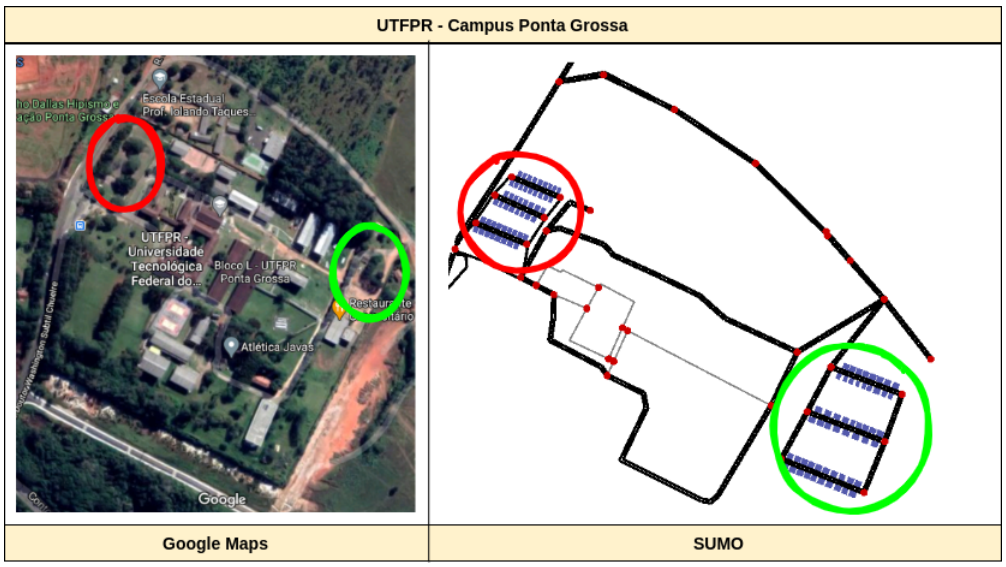

Figure 2. UTFPR (Ponta Grossa) - Map and Representation at SUMO

parking spaces in a Smart Parking. The developed MAS covers one smart parking with two parking areas (see Figure 2): red circle (left) for students and visitors, green circle (right) only for students. Regarding the simulation with SUMO, a network was designed with the roads and the parking areas[1] to simulate this scenario as showed in Figure 2.

The integration of MAS with SUMO was divided into two layers: agents and artifacts. The agent layer comprises the programming of agents in the Jason language and their interactions. Meanwhile, the second layer presents the artifacts developed in CArtAgO and their agents' interactions. Both layers are described in detail in the following sections.

3.1 Agent Layer

Altogether, three groups of agents compose the MAS: *builder, pspace, driver*. Figure 3 illustrates the agents, beliefs, and interactions (messages) of the system.

The *agents* group owns the following members:

- *builder*: it is responsible for instantiating all CArtAgO artifacts, workspaces, and *pspace* agents.
- *pspace*: it is able to representing the parking space, both physically and virtual (in SUMO, as shown in Figure 2).

[1]The term network in SUMO is used to denote a map used in the simulation.

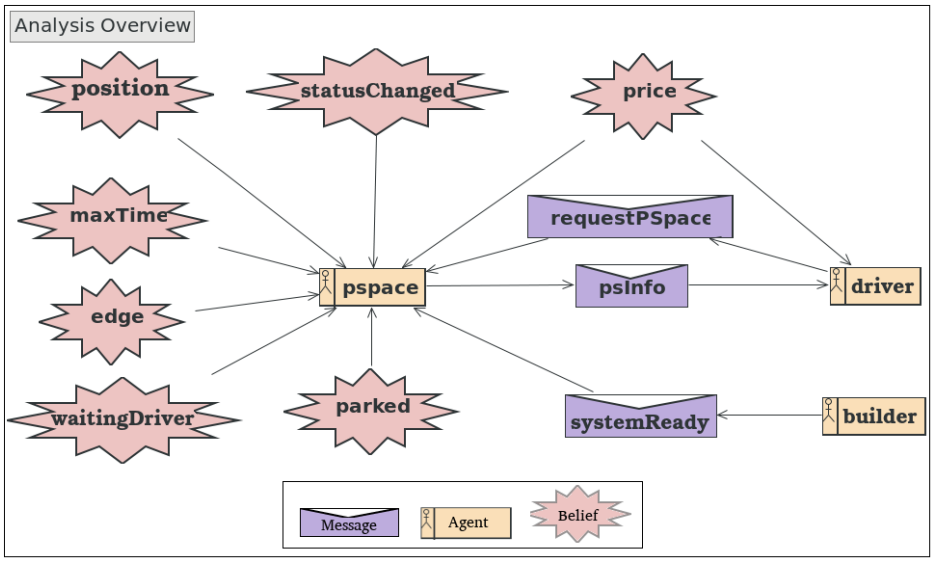

Figure 3. Prometheus Diagram - Agents: beliefs and messages

- *driver*: it represents the driver who wants to get a parking space.

The *pspace* agent has the following beliefs:

- *price*: the price for each hour of using the parking space;
- *statusChanged*: the value corresponding to the variation of the distance (ultrasonic) sensor (in use or free);
- *position*: the network path position in SUMO (0 - one way / 1 - regress);
- *maxTime*: the maximum period of time that a driver can use the parking space. If the time is over, there will be an extra charge in the price;
- *edge*: the value that identifies the edge (street) of the SUMO network in which the parking space is positioned. By default, SUMO provides pre-established editable values such as (edge_ID);
- *waitingDriver*: it informs if the parking space is in the status *waiting for a driver*. This belief is used if the parking space is occupied without being previously assigned to a given driver. Thus, if this occurs, the parking space was improperly occupied by an unwanted object or driver;
- *parked*: it indicates the presence of a driver parked in the parking space that was assigned to the driver.

Among the group of agents, the following messages can be exchanged:

- *requestPSpace*: the *driver* agent sends the message to the *pspace* agent informing the parking space request for a price predetermined by the *price* belief;
- *systemReady*: message sent when instantiating *pspace* agents, and Cartago *workspaces*, and artifacts. In this process, the *builder* agent sends a message to the *pspace* agents informing them that the system is working. Thus, *pspace* agents are open to requests from *driver* agents;
- *psInfo*: when the *pspace* agent accepts a parking space request, that agent must inform the respective *driver* agent of the information about the assigned parking space (e.g., location).

For the sake of clarity, we shall not present the details of how parking space is requested and negotiated. However, the reader may find specific MAS solutions for the negotiation of parking spaces in the following works [12], [13], and [14].

3.2 Artifact Layer - Connection with SUMO

The artifacts described in this scenario comprise the interconnection of MAS with SUMO, illustrated in Figure 4. Based on Figure 4, the artifacts used to connect the MAS to SUMO are `SUMOConnect`, `StructureInfo`, and `PSControl`. The artifact `MQTTConnection` is used to integrate the MAS to IoT Tools, which is explained on the Section 5.

The following artifacts are used in the system:

- ***SUMOConnect:*** The artifact is used to connect to SUMO. The connection to SUMO is made through the TraaS library [2] [15]. There are two operations that *pspace* agents can perform on the *SUMOConnect* artifact.
  - `addVehicle`: operation that adds vehicles to the SUMO network. MAS *driver* agents in JaCaMo are considered vehicles in SUMO, whereas *pspace* agents are SUMO's

[2] Python-based library that communicates with the SUMO TraCI (connection with *Socket*).

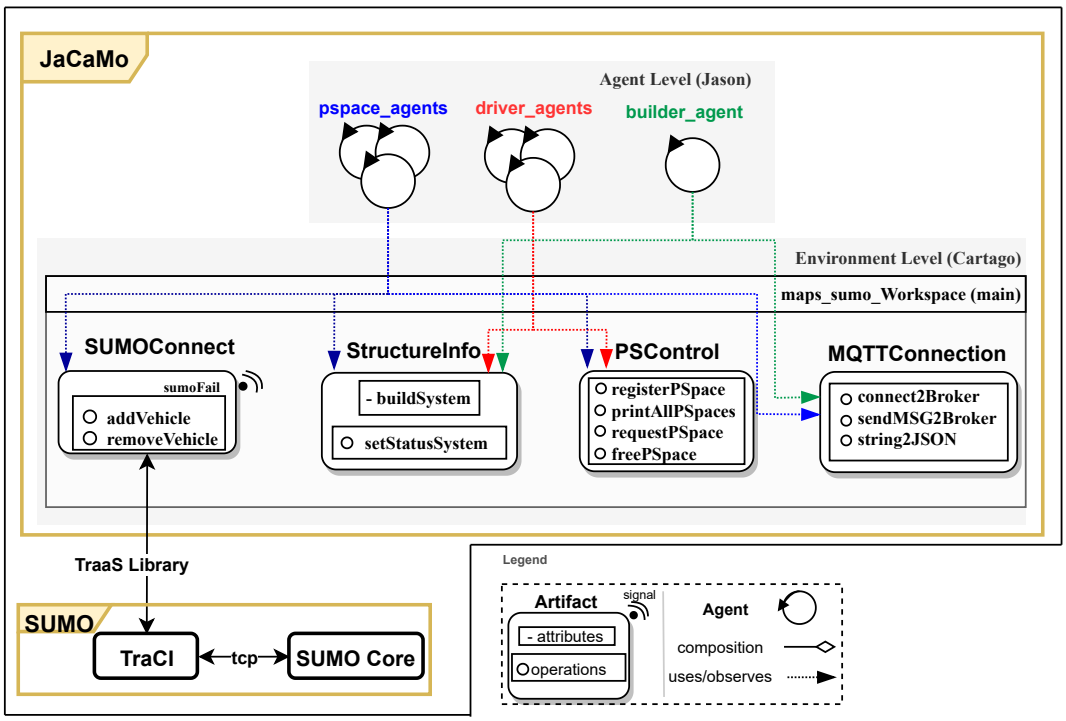

Figure 4. Interaction between JaCaMo and SUMO

parkingSpace resources. Listing 1 features the *addVehicle* function. The function parameters are the agent name (SUMO vehicle id), the SUMO road id (*edge*), and the road position (0 - one way, 1 - regress). The function initially (line 5) adds the vehicle to the SUMO network. In this line, there is also the *routeToParking*, which is previously defined as a sequence of roads used by the vehicle to find the parking space for it. Finally, line 6 defines which parking space the vehicle will occupy, which is previously negotiated in the agent layer.

Listing 1. Function: `addVehicle` - CArtAgO

```
1 public class A_SUMOConnect extends Artifact {
  SumoTraciConnection conn;
3 @OPERATION
  public void addVehicle(String agentName,
       String edge, double position) {
5     conn.do_job_set(Vehicle.add(agentName,"
          DEFAULT_VEHTYPE", routeToParking, lane
          , position, speed,Byte.valueOf("0")));
      conn.do_job_set(Vehicle.setParkingAreaStop
          (agentName, parkingAreaName, duration,
           timeOut, flag);}
```

– `removeVehicle`: Listing Code 2 presents the function of removing the vehicle from the network in SUMO. After a *driver* agent finishes using a *pspace*, the corresponding vehicle in SUMO is asked to leave the *parkingArea*[3] (Line 3) and start the route to the network exit (routeToLeave) (Line 4).

[3]The place where the driver parks the car we call parking space in this paper. However, the SUMO calls it as parkingArea in its tool.

Listing 2. Function: `removeVehicle` - CArtAgO

```
  @OPERATION
2 public void removeVehicle(String agentName) {
          conn.do_job_set(Vehicle.resume(
              agentName));
4         conn.do_job_set(Vehicle.setRoute(
              agentName, routeToLeave));}
```

Finally, in *SUMOConnect* there is the use of the *sumoFail* signal in case of a connection failure with *TraCI* and problems with the SUMO network.

- ***StructureInfo e PSControl:*** The *PSControl* artifact is used for registration, allocation, release, and control of *pspaces*. In turn, the *StructureInfo* artifact is used by the *builder* agent to inform the other agents about the MAS status.

4. Integration: Physical Environment and MAS

This section presents JaCaMo as an embedded system that employs a MAS that interfaces physical environments. There are two ways of interfacing the real world: **i.** using agents that interfaces the physical environment directly; and **ii.** using artifacts as an intermediary between agents and the hardware.

The first one allows agents to access raw data from sensors and process them as perceptions directly in their reasoning cycle in each cycle performed, and they can activate actuators by sending commands to the hardware using a serial interface. Since the hardware is managed by an autonomous and cognitive agent, it allows improving the decision-making process

at the edge of the system once the information was processed and reasoned. However, the more data the environment can produce, the more information the agent will deal with and its reasoning cycle could be overloaded in converting them into perceptions. Some solutions were implemented to tackle this issue, such as perception filters [16], which incorporates a filtering perception engine inside Jason's interpreter to eliminate the irrelevant perceptions and reduce processing time. Even with the perception filter, it is still possible to overload the reasoning cycle of an agent depending on how the filter is designed or even if new information becomes available in the environment considering its unpredictability.

The second way of interfacing with the real world is using artifacts to communicate with microcontrollers in the physical environment. Using these artifacts, the agents do not need to access the physical environment in each reasoning cycle performed and access the artifacts only when it is really necessary. In addition to alleviating the responsibility of the reasoning cycle, the artifact can also be used by other MAS agents, allowing sensors and actuators to be accessed by any MAS agents.

Both agent and environment layers use a serial interface for accessing sensors and actuators. The Javino [17] is a two-way communication protocol, which provides reliability since it has a process to verify the message's integrity to guarantee that there is no information loss during the communication process between the low and high levels.

4.1 ARGO: Jason Agent and Hardware Integration

The integration of Jason agents with hardware allows developing MAS in real physical environments. The notion of autonomy, pro-activity, and collaboration on the part of a MAS agent justifies the need to integrate this system into a real environment to deal with this physical environment's unpredictability.

ARGO [9] is a customized agent's architecture for communicating with microcontrollers in the AgentSpeak programming language. The agents using this architecture are named ARGO agents and allow developing a MAS interacting in a physical environment through Arduino, ESP, or PIC microcontrollers. ARGO uses the Javino serial communication interface as a middleware to establish the communication between the hardware and the MAS.

Therefore, with the Javino serial interface, an ARGO agent can send and receive information from microcontrollers. The information received is treated as perceptions that the ARGO agent receives from the environment via sensors of the hardware that the agent is controlling and, therefore, are automatically added as beliefs in this agent's belief base. The information sent to the microcontroller is treated as actions performed in the physical environment via hardware actuators.

In order to use these features of ARGO agents, four new internal actions (*i.e.*, pre-programmed actions or behaviors inherent to the agent [7]) were developed exclusively to control microcontrollers, namely:

- ***.port("Serial port")***: defines which serial port the agent will control and, consequently, which microcontroller the agent will control. This internal action has an argument that represents which serial port the agent will control;
- ***.perceive(open/block)***: defines whether the agent will perceive the environment or not. This internal action argues and has two options: *open*, used to make the agent open perceptions and receive information from the hardware; and *block* used to close the agent's perceptions and no longer receive information;
- ***.limit(Time in milliseconds)***: defines a time interval for the agent to perceive the environment. This internal action has an argument that represents the time in milliseconds that the agent will switch perception from open to closed;
- ***.act("Action")***: defines an action that the microcontroller must perform. This internal action has an argument that represents the action that must be sent to the microcontroller to perform.

In addition to ARGO agents, there are also Physical Artifacts that enable you to integrate conventional artifacts from the CArTAgO framework with physical devices and their microcontrollers [18].

4.2 Physical Artifacts: CArtAgO and Hardware Integration

The JaCaMo architecture [6] presents the difference between two environments: the internal environment, where the artifacts are logically organized, and the external environment, which can be represented by simulated or real scenarios. When adopting a physical environment composed of hardware technologies represented as artifacts, it is possible to develop Cyber-Physical Systems (CPS) employing agents using our proposed approach.

Artifacts can be used to interface the hardware in the real world since any agent from the MAS can access them if they are available. Besides, agents do not need to gather information from sensors and process it as perception in their reasoning cycle, which overloads agents depending on the application domain. Moreover, when agents interface such artifacts they do not need to process the gathered information in every cycle and they request it whenever it is necessary and available.

One advantage of these artifacts is the abstraction of the technical details of the hardware employed since agents access their operations provided by the usage interface. An artifact responsible for interfacing hardware in a physical environment is named Physical Artifact in this paper. It exchanges commands and data between hardware and agents, controls the actuators, and monitors the sensors of this device.

The Physical Artifact is an extension of traditional artifacts from CArtAgO and it also has (1) a set of operations that can be performed by agents, (2) instructions that describe how these artifacts should have their functionality accessed, (3)

the purpose of existence, and (4) the internal structures of its functionalities [8]. In CArtAgO, methods call Operations, which determine the behavior of the artifact, and Observable Properties allow the artifact to notify the agent of any event of the environment. Thus, agents can obtain data from a Physical Artifact by using Observable Properties, and they can perform commands in actuators using Operations. The Physical Artifacts also use Javino for interfacing the device's microcontroller. Thus, it is necessary to implement three abstract methods in the project:

- ***String definePort()***: returns the value of the communication port to be used to exchange data with the microcontroller;
- ***int defineAttemptsAfterFailure()***: returns the number of times the artifact will attempt to send the same message when an error has occurred during the message exchange;
- ***int defineWaitTimeout()***: returns the waiting time in milliseconds between two requests made to the device.

When it is necessary to implement Operations and Observable Properties of Physical Artifacts, two methods can be employed:

- ***String read()***: returns a message sent by the device's microcontroller, usually showing the measurements collected by its resources;
- ***void send(String message)***: sends the message passed by parameter to the device, usually presenting a command to operate this device's resources.

5. Integration: IoT Applications and MAS

The proposed approach so far showed how to integrate MAS with the external environment, whether physical or simulated. Although the approach offers these mechanisms, there is still a gap when MAS needs to communicate with peers or with other entities. If this communication became possible, MAS and other entities inserted in similar or different environments could share their knowledge and optimize processes.

In this context, the Internet of Things (IoT) promises that devices and clients can establish communication as a network node on a large scale. However, one of the challenges of IoT is to ensure that devices of different types communicate in the same language and in a secure and scalable way.

In order to develop such a feature, communicating the MAS with other IoT entities through the network, we design different modules that together provide interaction between MAS and IoT platforms. Figure 5 presents these modules as well as their connections and sequential interactions, which are described below.

- **IoT Prototype:** The prototype consists of three items: Two Arduino (One Arduino MEGA and one Arduino UNO), and a Raspberry Pi 3. Initially, the Arduino performs the reading and sends the sensor data (ultrasonic distance) to the pspace agents on MAS. The Raspberry Pi 3 is responsible to host the MAS, NODE-RED Application, Influx Database, and the Grafana Server and its Dashboards;
- **MAS:** The MAS is composed of the following agents and artifact to connect the MAS to the IoT Tools;
  - *Artifact: MQTT Connection* It is able to estabilish a connection with the IoT Platform (MQTT Broker) in order to notify it about the pspace sensors and drivers. The communication between the MQTTConnection and the IoT Platform is provided by the Eclipse Paho[4]. This artifact is composed by three operations, which are:
    - *connect2Broker:* This operation is able to connect the MAS to the MQTT Broker (see Listing 3). Initially on lines 4-8, there is setup to assign the device id, url and port of the broker, and the user and password. After, on line 10, the MAS tries to connect.

Listing 3. Operation:connect2Broker - CArtAgO

```
  @OPERATION
2 void connect2MQTTBroker() {
  mqttClient = new MqttClient(MQTT_Config.BROKER
       .getValue(), MQTT_Config.DEVICE_ID.
       getValue(), persistence);
4 cOptions = new MqttConnectOptions();
  cOptions.setCleanSession(true);
6 cOptions.setPassword(MQTT_Config.PASS.getValue
       ().toCharArray());
  cOptions.setUserName(MQTT_Config.USER.getValue
       ());
8 try {
          mqttClient.connect(cOptions);
10     System.out.println("Connecting to broker:
           " + MQTT_Config.BROKER.getValue());
```

    - *sendMSG2Broker:* After a successful connection to the Broker, the MAS sends a message to the Broker with an updated sensor value if some sensor has its value changed. The message is based on JSON format due to the required payload from the Broker. The JSON message is composed of two fields: value and unit. Since we use ultrasonic sensors, the unit is centimeters. This operation is presented on Listing 4. First, on line 1 is the sensor value is being cast as a JSON message. After on line 2 is the QoS setup and finally on line 3 is the publishing of the JSON message on the MQTT Broker.

Listing 4. Operation: `sendMSG2Broker` - CArtAgO

```
  MqttMessage message = new MqttMessage(this.
       string2JSON(msg, "cm").getBytes());
2 message.setQos(Integer.valueOf(MQTT_Config.QOS
       .getValue()));
  mqttClient.publish(MQTT_Config.PUBLISH_TOPIC.
       getValue(), message);
```

  - *ARGO Agent*: it establishes the connection with the IoT prototype to perceive the physical environment through

[4]This is library by the Eclipse Foundation that provides MQTT connections (publish/subscribe) between the client and broker MQTT

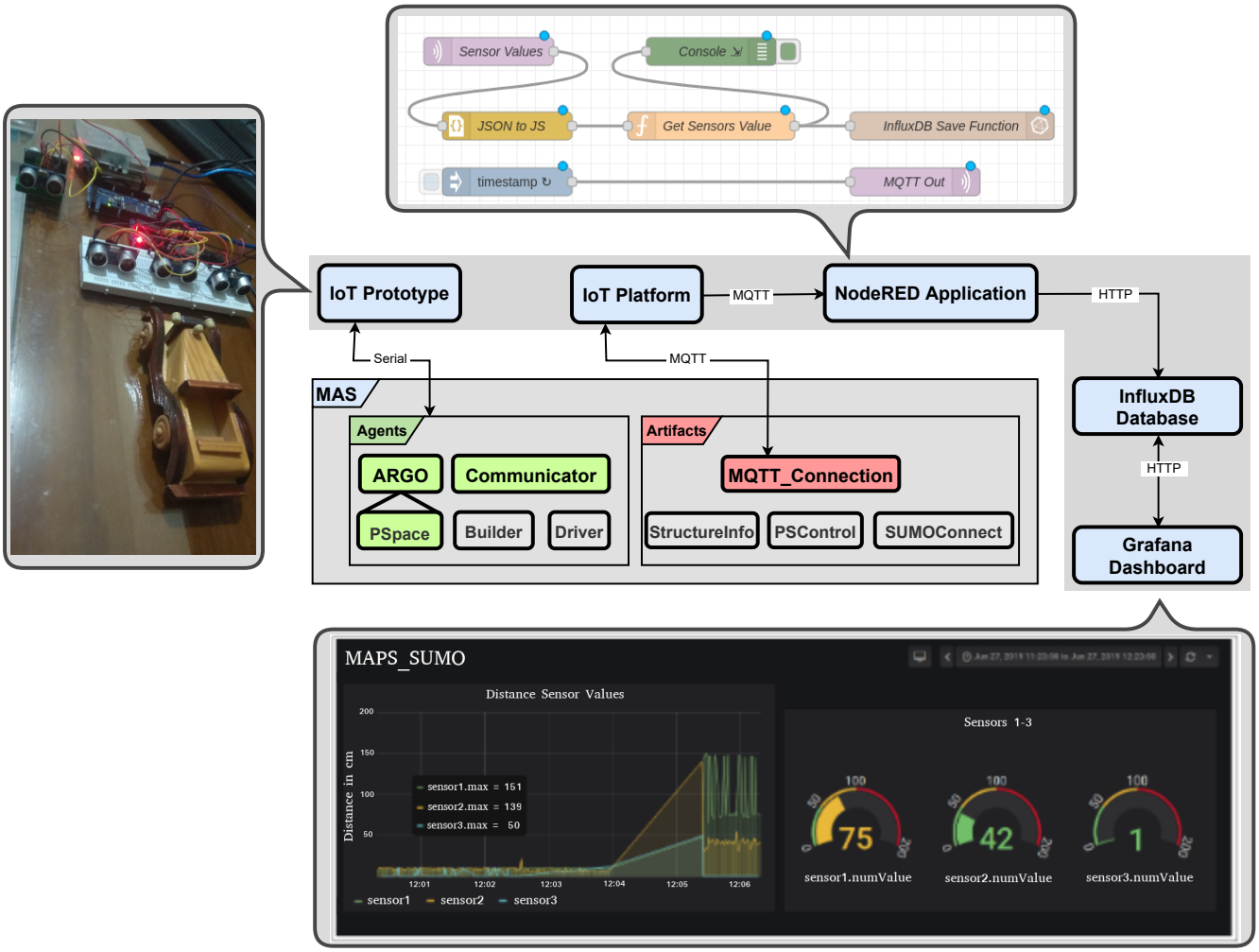

Figure 5. Integration: Prototype, MAS, platform and IoT Applications

ultrasonic distance sensors. Upon obtaining the perceptions, the ARGO agent notifies the artifact MQTTConnection and the Communicator agent if there is environment changing (significant change in the sensor values). In our work, the pspace agents are considered the ARGO agents, since the pspace agents have to perceive the physical environment to check if something has ocurred;

– *Communicator Agent (ContextNET)*: the agent is responsible for communicating with other MAS via the ContextNET network. Using this agent, the MAS can communicate with other MAS for multiple cooperation between the systems. For example, a network of smart parking lots available at various locations in the same city. Thus, a single parking lot can provide parking spaces.

- **IoT Platform:** the IoT platform is a MQTT Broker which aims to integrate different IoT devices through MQTT routes (publish/subscribe). Upon receiving the messages via the prototype's MQTT, the platform redirects the messages via MQTT to NodeRED. The platform used is the one provided by KonkerLabs. Moreover, there are other solutions provided by Eclipse, Google, and others;

- **NodeRED:** used to decomposition and extract JSON messages from the IoT platform and storage in InfluxDB via HTTP. Decomposition and extraction is used to obtain the value of the prototype sensors;

- **InfluxDB:** Time series databases (TSDB). The use of this type of database is due to its characteristic of having a timestamp as its primary key. Thus, the storage of the sensor values is carried out according to the measurement time;

- ***Dashboard* Grafana:** The Grafana dashboard provides different components for visualizing data in real-time. The dashboard accesses the sensor data via HTTP in the InfluxDB database.

5.1 The ContextNet

The ContextNet [11] is a middleware created based on the Internet of Things (IoT) and context service that focuses on collaborative applications, coordination of activities of mobile entities, and information sharing. Mobile entities can be mobile devices, such as vehicles, smartphones, tablets, laptops,

or even autonomous robots connected to the IoT network. Among the technologies used for the development of ContextNet, the Scalable Data Distribution Layer (SDDL) is a middleware embedded in ContextNet responsible for the ability to communicate and distribute context. The other features of ContextNet were developed in software modules at the top of the distribution layer. Besides, ContextNet allows devices to enter the network and treat them appropriately.

Besides the MAS integration with the physical environment and urban simulation, our approach also allows MAS to be integrated with the IoT. Thus, each embedded system can communicate with another embedded system or even with other nodes in the ContextNet network.

5.2 The Communicator Agent

Communicator Agent [19] is an agent extension that implements an IoT node using ContextNet. With this, Communicator Agent can communicate with entities distributed over the network, no matter they are sensors, actuators, web clients, or even other Communicator Agents. Each Communicator Agent has a Universally Unique Identifier (UUID), which must be known by the entity that wishes to exchange messages. When sending messages, Communicator Agents can define the illocutionary force as *tell* or *achieve*. The first is to update the receptor agent's belief base, and the second is to update the receptor agent's plan library.

The integration between MAS and urban simulation using the UTFPR map (Ponta Grossa) (see previously in Figure 2) can also be done with Communicator Agents. In this case, *pspace* and *driver* become Communicator Agents who communicate through ContextNet to negotiate parking spaces. With this, these agents can even be in different MAS, which allows expanding the MAS reach that carries out the negotiation.

The Communicator Agent has an internal action called *sendOut* that takes as a parameter the recipient's UUID, the elocutionary force of the message (achieve or tell), and the message. The messages sent between Communicator Agents are given as follows:

- *requestPSpace*: sent from agent *driver* to agent *pspace* by calling *sendOut*. The parameters are the UUID of the agent *pspace*, the elocutionary force *tell* and the parking space request message. Example:

  .sendOut("788b2b22-baa6-4c61-b1bb-01cff1f5f879", achieve, requestPSpace(MT)),

  where *requestSpot(MT)* is the request message.

- *psInfo*: sent from agent *pspace* to agent *driver* by calling *sendOut*. The parameters are the UUID of the agent *driver*, the elocutionary force *tell* and the message with the parking space information.

6. Conclusion

This work presents an approach for integrating a MAS developed in JaCaMo with Urban Simulation tools, Embedded Systems, and IoT Applications. The main objective is to deploy such a variety of tools and systems in a way that a strong coupling between all the levels is avoided.

The advantage of this approach is its generalization. The proposed levels are heterogeneous, allowing different applications to use this approach as support for systems integration. Another advantage is the low coupling between the layers, which gives the flexibility to use both the fully integrated architecture and only parts. Through this work, problems that demand scalability and an intelligent system can be solved with IoT integrated into a MAS. It is still possible to choose which type of environment should be used: simulated or physical one.

As future work, we intend to develop an integration methodology as a framework for integrated urban mobility solutions MAS, IoT, Simulation, and Embedded Systems. It is also necessary to deploy an application (robust and complex) in the domain of Smart Cities, which uses all the proposed levels. With such an application, it will also be possible to run experiments to evaluate the integration methodology's characteristics, such as scalability, low coupling, reliability in the exchange of messages and information, among others. Finally, we also intend to develop the social organization layer by using Moise (JaCaMo). Thus, it will be possible to establish rules and behaviors in agent societies, promote a social organization, and create social groups of agents; these elements can properly represent smart cities and urban mobility features.

Acknowledgements

The work of the first author is supported by Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Finance Code 001. The author is grateful for the financial support.

Author contributions

- Conceptualization: All authors.
- Formal Analysis: All authors.
- Investigation: All authors.
- Practical experiments: Lucas Fernando Souza de Castro, Fabian Cesar P. B. Manoel, Vinícius Souza de Jesus.
- Supervision: Lucas Fernando Souza de Castro, Gleifer Vaz Alves.
- Writing-original draft: Lucas Fernando Souza de Castro, Fabian Cesar P. B. Manoel, Vinícius Souza de Jesus.
- Writing-review & editing: All authors.

References

[1] EVANS, D. How the Next Evolution of the Internet of Things Is Changing Everything. *Cisco Internet Business Solutions Group*, p. 11, 2011.

[2] ALBINO, V.; BERARDI, U.; DANGELICO, R. M. Smart cities: Definitions, dimensions, performance, and initiatives. *Journal of urban technology*, Taylor & Francis, v. 22, n. 1, p. 3–21, 2015.

[3] NEIROTTI, P. et al. Current trends in smart city initiatives: Some stylised facts. *Cities*, v. 38, p. 25 – 36, 2014. Disponível em: ⟨http://www.sciencedirect.com/science/article/pii/S0264275113001935⟩.

[4] WOOLDRIDGE, M. J. *Reasoning about rational agents*. [S.l.]: MIT press, 2000.

[5] ZHANG, D. et al. Internet of things. *J. UCS*, v. 18, p. 1069–1071, 2012.

[6] BOISSIER, O. et al. Multi-agent oriented programming with jacamo. *Science of Computer Programming*, Elsevier, v. 78, n. 6, p. 747–761, 2013.

[7] BORDINI, R. H.; HüBNER, J. F.; WOOLDRIDGE, M. *Programming Multi-Agent Systems in Agent Speak using Jason*. [S.l.]: John Wiley & Sons Ltd, 2007. 273 p.

[8] RICCI, A.; VIROLI, M.; OMICINI, A. Programming MAS with artifacts. v. 3862 LNAI, p. 206–221, 2006. Disponível em: ⟨https://www.scopus.com/inward/record.uri?eid=2-s2.0-33745661012&doi=10.1007%2f11678823_13&partnerID=40&md5=6c8f34aae1c78bccd47fabd2c3705e5e⟩.

[9] PANTOJA, C. E. et al. Argo: An extended jason architecture that facilitates embedded robotic agents programming. In: SPRINGER. *International Workshop on Engineering Multi-Agent Systems*. [S.l.], 2016. p. 136–155.

[10] KRAJZEWICZ, D. et al. Sumo (simulation of urban mobility)-an open-source traffic simulation. In: *Proceedings of the 4th middle East Symposium on Simulation and Modelling (MESM20002)*. [S.l.: s.n.], 2002. p. 183–187.

[11] ENDLER, M. et al. Contextnet: context reasoning and sharing middleware for large-scale pervasive collaboration and social networking. In: ACM. *Proceedings of the Workshop on Posters and Demos Track*. [S.l.], 2011. p. 2.

[12] CASTRO, L. F. S. D.; ALVES, G. V.; BORGES, A. P. Using trust degree for agents in order to assign spots in a Smart Parking. *ADCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal*, v. 6, n. 2, p. 45–55, jun. 2017. Disponível em: ⟨http://revistas.usal.es/index.php/2255-2863/article/view/ADCAIJ2017624555⟩.

[13] DUCHEIKO, F. F.; ANDRé, P. B.; GLEIFER, V. A. Implementação de Modelo de Raciocínio e Protocolo de Negociação para um Estacionamento Inteligente com Mecanismo de Negociação Descentralizado. *Revista Junior de Iniciação Científica em Ciências Exatas e Engenharia*, v. 1, n. 19, p. 25–32, 2018. Disponível em: ⟨http://www.icceeg.c3.furg.br/index.php?Itemid=837&option=bloco_texto&id_site_componente=1241⟩.

[14] ALVES, B. R. et al. Experimentation of Negotiation Protocols for Consensus Problems in Smart Parking Systems. In: MARIK, V. et al. (Ed.). *Industrial Applications of Holonic and Multi-Agent Systems*. Cham: Springer International Publishing, 2019. (Lecture Notes in Computer Science), p. 189–202.

[15] KRUMNOW, M. Sumo as a service–building up a web service to interact with sumo. In: SPRINGER. *Simulation of Urban MObility User Conference*. [S.l.], 2013. p. 62–70.

[16] STABILE Jr., M. F.; PANTOJA, C. E.; SICHMAN, J. S. Experimental analysis of the effect of filtering perceptions in bdi agents. *International Journal of Agent-Oriented Software Engineering*, Inderscience Publishers (IEL), v. 6, n. 3-4, p. 329–368, 2018.

[17] LAZARIN, N. M.; PANTOJA, C. E. A Robotic-Agent Platform for Embedding Software Agents using Raspberry Pi and Arduino Boards. In: 9^{th} *Software Agents, Environments and Applications School*. [S.l.: s.n.], 2015.

[18] MANOEL, F. C. P. B. et al. Physical artifacts for agents in a cyber-physical system: A case study in oil & gas scenario (EEAS). In: GARCÍA-CASTRO, R. (Ed.). *The 32nd International Conference on Software Engineering and Knowledge Engineering, SEKE 2020, KSIR Virtual Conference Center, USA, July 9-19, 2020*. KSI Research Inc., 2020. p. 55–60. Disponível em: ⟨https://doi.org/10.18293/SEKE2020-154⟩.

[19] PANTOJA, C. E. et al. An architecture for the development of ambient intelligence systems managed by embedded agents. In: *SEKE*. [S.l.: s.n.], 2018. p. 215–214.