Bruno Miguel Morais Dias

Licenciado em Ciências de Engenharia Eletrotécnica e
de Computadores

Integration of a mobile autonomous robot in a surveillance multi-agent system

Dissertação para obtenção do Grau de Mestre em
Engenharia Eletrotécnica e Computadores

Orientador: Pedro Alexandre Sousa, Professor Auxiliar, FCT-UNL

Co-orientadores: João Paulo Pimentão, Professor Auxiliar, FCT-UNL

Júri:

Presidente: Professor Doutor Ricardo Luís Rosa Jardim Gonçalves
Arguentes: Professor Doutor José António Barata de Oliveira
Vogais: Professor Doutor João Paulo Branquinho Pimentão

FACULDADE DE CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

Setembro, 2014

Integration of a mobile autonomous robot in a surveillance multi-agent system

There is no elevator to the success, you have to take the stairs.

Acknowledgments

Firstly, I would like to thank Professor Pedro Sousa and Professor João Paulo Pimentão for having accepted to be my supervisor and co-supervisor respectively and the opportunity to work on ambitious projects in my area of interest. Their orientation was decisive in the course of work done in this dissertation. I would also like to thank Sergio Onofre, David Rodrigues, Tiago Ferreira and staff of Holos for all the help given, especially in the implementation details of this dissertation.

Then I would like to thank my friends who followed my academic journey from the beginning, Paulo Rodrigues, Tiago Pereira, André Pardal, José Reis, Ricardo Pombeiro, Tiago Bento, António Bernardino, Fábio Nogueira, Fábio Lourenço, Celso Almeida, Ana Rita Salvado and Rui Cardoso. Without their cooperation and all the spirit of the group, would not be possible to reach this milestone in my academic journey. I would also like to thank the colleagues who worked with me on the same project during the development of this thesis, Jorge Claro, Bruno Rodrigues and Fabio Miranda.

Also want to thank my friends who have accompanied me before I entered college, Andreia Santos, Tânia Batista, Cláudia Pereira, Joel Pereira, Diogo Soares, Daniela Filipe, Tânia Nini and a special thanks to Mário Freire, for all the friendship and experiences we shared.

I also want to express special thanks to my parents and my sister for all the help, patience and care that have given me during my life. They were

always present and were a comforting support throughout the stages of this thesis.

Resumo

Esta dissertação tem como objetivo garantir a integração de um robô móvel e autónomo dotado de vários sensores num sistema distribuído, georreferenciado, e multi-agente de segurança.

Com a integração de um robô móvel e autónomo neste sistema abrem-se portas a diversas funcionalidades que os clientes do sistema de segurança poderão usufruir. Essas funcionalidades podem ser de dois tipos: utilizando o robô como um agente que irá atuar no ambiente envolvente ou utilizando o robô como um conjunto de sensores móvel. Como um agente no sistema, o robô poderá se deslocar a determinadas localizações quando são recebidos alertas, de forma a realizar reconhecimento dos eventos ou executar uma ação de forma a ajudar na solução deste evento. Como uma plataforma de sensores no sistema, poderá aceder-se à informação detetada a partir dos sensores presentes no robô e, desta forma conseguir complementar medições realizadas por outros sensores do sistema multi-agente.

Para integrar este robô móvel de uma forma eficaz é necessário estender a arquitetura atual do sistema multi-agente para que faça a ligação entre os dois sistemas e as funcionalidades prestadas pelo robô sejam integradas no sistema multi-agente.

Palavras-chave: sistema multi-agente, robô móvel, sistema de segurança

Abstract

This dissertation aims to guarantee the integration of a mobile autonomous robot equipped with many sensors in a multi-agent distributed and georeferenced surveillance system.

The integration of a mobile autonomous robot in this system leads to new features that will be available to clients of surveillance system may use. These features may be of two types: using the robot as an agent that will act in the environment or by using the robot as a mobile set of sensors. As an agent in the system, the robot can move to certain locations when alerts are received, in order to acknowledge the underlying events or take to action in order to assist in resolving this event. As a sensor platform in the system, it is possible to access information that is read from the sensors of the robot and access complementary measurements to the ones taken by other sensors in the multi-agent system.

To integrate this mobile robot in an effective way it is necessary to extend the current multi-agent system architecture to make the connection between the two systems and to integrate the functionalities provided by the robot into the multi-agent system.

Keywords: multi-agent system, mobile robot, surveillance system

Contents

List of Figures

List of Tables

Glossary

ACK - Acknowledge

ACL - Agent Communication Language

CCTV - Closed Circuit Television

CFP - Call For Proposals

CORBA - Common Object Request Broker Architecture

DVA - Advanced Surveillance System based on Agents

FIPA - Foundation for Intelligent Physical Agents

GPS - Global Positioning System

JADE - Java Agent DEvelopment

JAX-B - Java Architecture for XML Binding

LAN - Local Area Network

MAS - Multi-Agent System

OpenRAVE - Open Robotics Automation Virtual Environment

OpenRTM - Open RT Middleware

OROCOS - Open Robot Control Software

P2P - Peer-to-peer

PUB/SUB - Publisher/Subscriber

REQ/REP - Request/Reply

ROS - Robot Operating System

RTAI - RealTime Application Interface

SOA - Service-Oriented Architecture

SOAP - Simple Object Access Protocol

UDDI - Universal Description, Discovery and Integration

WSDL - Web Service Description Language

XML - Extended Markup Language

XSD - XML Schema Document

YARP - Yet Another Robot Platform

1

1. Introduction

1.1. Motivation

Since technology is evolving day-by-day, new technological paradigms are emerging in order to help the humanity to live longer, safer and more comfortable. Evolution at telecommunications is enabling better communication between different systems and devices. Telecommunications are getting available almost at every place and, with developments at wireless technology, it enables mobility, privacy and quality of service in the communication process, mainly using mobile networks [1]. Nowadays, it is possible to capture information about the environment (physical, chemical and biological properties) and to classify it according to the values obtained, gathered by different devices [2,3].

Evolution at robotics made it possible to use autonomous mobile robots to help the human activity, mainly at industry, military and surveillance applications. Machines are operating with more autonomy and, this way, some tasks can be delegated to robots, decreasing human dependency [4]. Besides, in some tasks where human supervision is indispensable, with the evolution of artificial intelligence, machines are getting smarter and able to process fast and efficiently large amounts of information replacing efficiently the humans, as it happens at intelligent surveillance.

Surveillance systems are part of the current mechanisms of the society for its protection against events that attempt against people's health and goods. The first systems developed for this purpose were centralized video surveillance systems, where the feed from surveillance cameras was shown at a control room and security

guards had to monitor the images collected, like CCTV – the first generation of surveillance systems [5]. These systems have evolved, using digital techniques to gather information and being less dependent on humans, automatizing the events' detection. Besides camera feed, the current generation of surveillance systems are using more sensorial information to detect events and are being implemented with distributed architectures. With the information gathered by their sensors, it is possible to process and trigger events to notify the responsible authorities to act in case of need [6,7].

1.2. Problem

A traditional surveillance system is composed by sensors and cameras at fixed positions. This "less-mobile" architecture represents a problem when it is required to monitor a vast area, due to information overload and to large number of devices operating in parallel [7,8]. As the system's sensors operate in fixed locations, they have no possibility of adapting to events, whose location is known, which need to gather more specific information or perform a special action.

With these systems it is possible to add more sensors to extend their coverage area and gather more information about the environment. This way, the system's coverage is static over the time, without the possibility to change the sensor network topology automatically, as needed and be reconfigured to optimize its performance. Also, the computational power required to process all that data increases significantly.

A system capable of relocating its sensors elsewhere is surely more flexible and able to adapt to the specific needs with no need for a reconfiguration of the system's topology and sensors. The mobility of sensors is a factor to improve the traditional surveillance systems, so the system's coverage is dynamic and able to handle delicate situations with more efficiency, without being necessary to add more resources to the system [9].

Most traditional surveillance systems are centralized, where all the information collected is transmitted to a processor node (usually located in a control room), which is to an interface conveyed where the human element is in charge of the detection of most of events. This type of centralized architecture is always dependent on this central node. If it breaks down, it can hinder the entire system's performance [7].

Besides adding more sensors to surveillance systems, the integration with other system's elements and platforms can improve the autonomy, flexibility and reliability of the surveillance systems. Elements capable of acting on the environment can be classified as actuators and increase the autonomy of the system, reducing the dependence of human action to perform an action on an event detected by the system. Typically in a surveillance system, the human element is responsible for the monitoring of the image (or images) captured by the camera or by analyzing the values read by the sensors, which results in an expensive, time consuming, task since it is allocating all the attention of a person in a monotonous routine, which may to induce it in error [10]. This shows the benefits that can be attained by in increasing the level of automation of the surveillance systems, making their automatic event detection methods more efficient, thus freeing manpower for other tasks, where they can take full advantage of their capabilities.

These actuators can be mobile robotic elements with surveillance and monitoring capabilities, that can be used within a surveillance system, making these systems more autonomous and less dependent on human interaction. Mobile and autonomous robots are elements that may supply their capabilities to a surveillance system so that it can make use of them. This integration will solve some of the problems described above and partially reduce the current surveillance system's limitations.

However, when the most surveillance systems have been developed, these didn't include robots (or mobile sets of sensors) as elements in their architecture, and many service robots are currently being developed without the concern of being integrated within a surveillance system.

This raises a problem of communication, since they are developed in different frameworks and development platforms, using different programming languages and operating systems.

The problem here, is that it is necessary to create a link that will bridge the gap between these two different components, and efficiently integrate the services of a mobile robot equipped with sensors in a surveillance system, so that it can make the surveillance system more autonomous. This thesis aims to accomplish this integration of two different systems. To validate the implementation of this thesis

we will consider the system of intelligent surveillance DVA[1] and the service robot ServRobot[2], both developed by Holos.

1.3. Structure of the dissertation

This chapter presents the introduction to the theme of the dissertation. The overall context is presented, the focus problem was identified. In chapter 2, is presented the state of art technologies involved in the subject of this dissertation. In chapter 3, an explanation of the proposed architecture to solve the problem presented is made. In chapter 4, are detailed the application scenario and steps of the implementation of the proposed architecture. In chapter 5, presents and discusses the results obtained by different tests, based on implementation done. And to conclude this thesis, in chapter 6, a conclusion of all the work and research undertaken pursuant this work is presented.

[1] http://dva.holos.pt

[2] http://servrobot.holos.pt

2

2. State of the art

This chapter presents the state of the art that concerning the development carried out by the scientific community regarding multi-agent systems and robotic systems. In Chapter 2.1 an introduction is made regarding distributed systems. In Chapter 2.2 a definition of multi-agent systems is presented, taking into account the characteristics of an agent. In Chapter 2.3 the agent's communication, negotiation and social behaviors are detailed. In Chapter 2.4 an analysis is made of how multi-agent systems can be applied to several areas. In Chapter 2.5 the multi-agent systems applications are focused at the applications to robotic systems, then in Chapter 2.6 different tools for developing multi-agent and robotic systems are presented and analyzed. Finally, in chapter 2.7 is done a brief explanation of the different architectures of surveillance systems, since this is the kind of systems where this dissertation is focused.

2.1. Distributed Systems

With the evolution of computing capabilities and network services, distributed architectures have been much in demand to solve certain problems in the technology world, since without a centralized point, it is possible to design a more robust and fault tolerant system [11].

It is possible to observe in nature certain societal behaviors distributed among living beings, who use communication as a fundamental pillar for the functioning of that society, and these are characteristics that can be applied it to a distributed

robotic system. One of the most popular examples is the case of ants; how they are organized, communicate with each other through the environment (relying on chemical communication) to be able to find food and to carry it to their nest successfully [12]. This case was a source of inspiration for many distributed systems, particularly for robotic swarms [13].

Different distributed systems can be implemented at software engineering level, by replacing processes and services that are usually executed by human resources. There are different software development paradigms for distributed systems [14], that model these processes and different types of data, using different techniques and perspectives. Each one has its pros and cons:

- Object-oriented, where an object has a set of attributes that contain information, which can be accessed through methods. An object encapsulates all these functionalities and its own data. An object-oriented system is usually composed by different instances of objects that can be extended to other objects. It is an approach that promotes modularity, extensibility and reusability. An efficient concurrency is only possible to achieve by implementing threads [15].
- Components, similar to objects, can encapsulate functionalities and data but they give importance to the context where the component is inserted. Concerning context dependencies, a component has well defined interfaces that allow the implementation of non-functional aspects like scalability and security by configuring these interfaces. Organizing components at containers can allow request-based concurrency transparent to the user. Distribution of data it is only possible with object-oriented remote invocation [16].
- Service-oriented architecture (SOA) is an approach that is based in distributed applications in different domains. It follows the concept of service in the real world, where services can be found to be decomposed at different business levels. So at SOA, a user invokes a service that is processed at an higher level. In a SOA paradigm, different entities and business levels are involved, so it is important to maintain interoperability of service invocations by following industry "de-facto" standards such as WSDL, UDDI and SOAP [17].

- Agent-oriented, it uses software agents that are running at a certain environment with an individual objective. An agent should be autonomous, with the capability of decision-making and should be reactive to environmental changes. Different agents can communicate with each other using social capabilities. To this purpose, multi-agent systems have been developed, improving the interaction and collaboration between multiple agents involved in a distributed system [18].

2.2. Multi-agent system definition

A Multi-Agent System is one of the approaches to achieve a robust distributed system. This system is constituted by several collaborating autonomous agents which results in a set of distributed processes that can be independent and with communication capabilities [19].

These agents can reside on the same computer or on several computers over a network. Some multi-agent platforms ever support agent integration over the platforms computers. An agent can be a processing node and usually plays a specific role in the system, e.g. representing a physical resource, a service, a manager, a broker, an interface to an user and other components external to the system.

Each agent can act autonomously by taking simple and particular objectives. But when facing a complex problem, a simple agent may not be able to solve it. It is necessary to make a task decomposition of the initial problem, in order to solve this problem, tackling smaller parts of it [20], and to have a team of agents performing those decomposed tasks towards the problem's solution.

Since it is necessary to operate with a team of agents, those agents must have certain social capabilities, otherwise their cooperation and collaboration isn't going to be optimal [19]:

- some autonomy so that the agents can make decisions about their individual operation;
- communication methods (like broadcast or unicast messages) to interact with other agents in the system;
- negotiation skills that will help to overcome some changes on the environment or on other agents;

These systems can be really dynamic with parallel units working together since each agent is autonomous, which makes this system really robust and adaptive to dynamic environments.

A variant of this traditional model for Multi-Agent Systems is Intelligent Multi-Agent System which incorporates a learning module that provides to this system the capability to learn over the time, and to be more efficient responding to the events on their environment. This feature can be achieve applying machine learning techniques like Neural Networks. [4].

2.3. Agent Communication

A fundamental aspect of a multi-agent system is the ability of supporting communication between the different agents involved. This communication can be categorized in two ways [12]:

- **Explicit Communication**: is the conventional type of communication, where an agent takes the initiative to send a signal or message to another specific agent.
- **Implicit Communication**: is when the way the agent interacts with its environment can be observed and interpreted as a message from that agent.

In the research developed by Balch and Arkin [12], the behavior of a team of mobile robots implemented with a multi-agent system was studied with different methods of communication. Each robot was an agent of the system and their task was to search and find an object in the laboratory. These robots used Explicit Communication when they were exchanging messages between them about their current state (searching, found an object) or about their goal (when they found the object, they sent their coordinates). Since the goal communication got more information about the goal than the state communication, the goal communication contributed more efficiently to the system's performance. This experience shows how information that is used for communication between agents can influence its performance, as shown on Figure 1 where the elements Robot 1, Robot 2 (robots used in the experience) and Attractor (the object to be found) are represented with a dot and the robots path is represented with a dashed line (when the robots are searching for the object) and a solid line (when the robots found the object). The robots initiated their search for the Attractor at the same time.

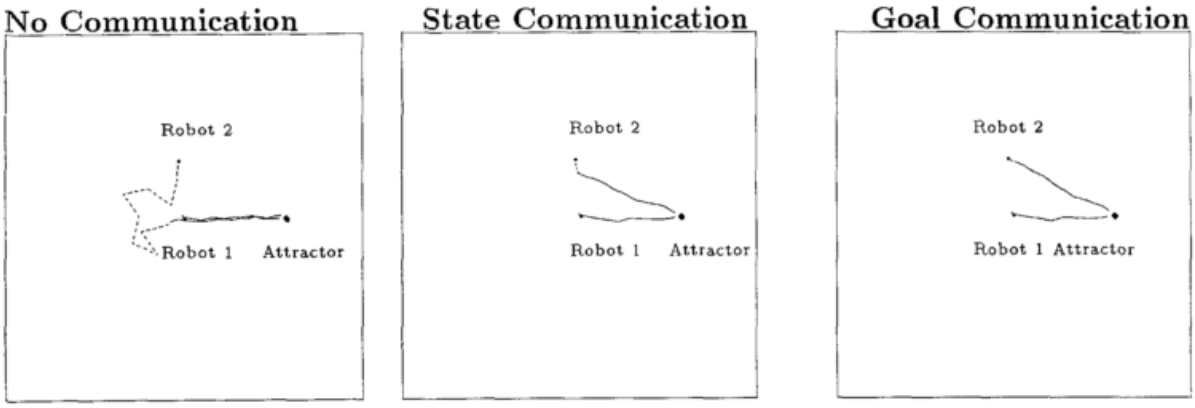

Figure 1 – Comparison of the different methods of communication [12]

Still in this research work, a simulation was made where robots exploring the area of laboratory and used the communication methods above. While the robots were exploring the area, they left a mark on the floor and that was visible to other robots. The robots didn't explore an area that was previously explored by other robots. Performing this task, the robots were using Implicit Communication by leaving a mark on the floor when they explored it. With this implicit communication, the experiments with State Communication or Goal Communication didn't have a relevant contribution to the team's performance, one can then conclude that in some situations, where Implicit Communication is used, the Explicit Communication may become irrelevant.

With the ability to exchange messages, the agents can also have the ability to negotiate with each other, which is very useful when you need to assign tasks to certain agents. To accomplish this, the agents must follow a message protocol like Contract Net Protocol [21], described below at Figure 2. This mechanism to negotiate tasks and resources is more efficient than a centralized approach to make this decision [22].

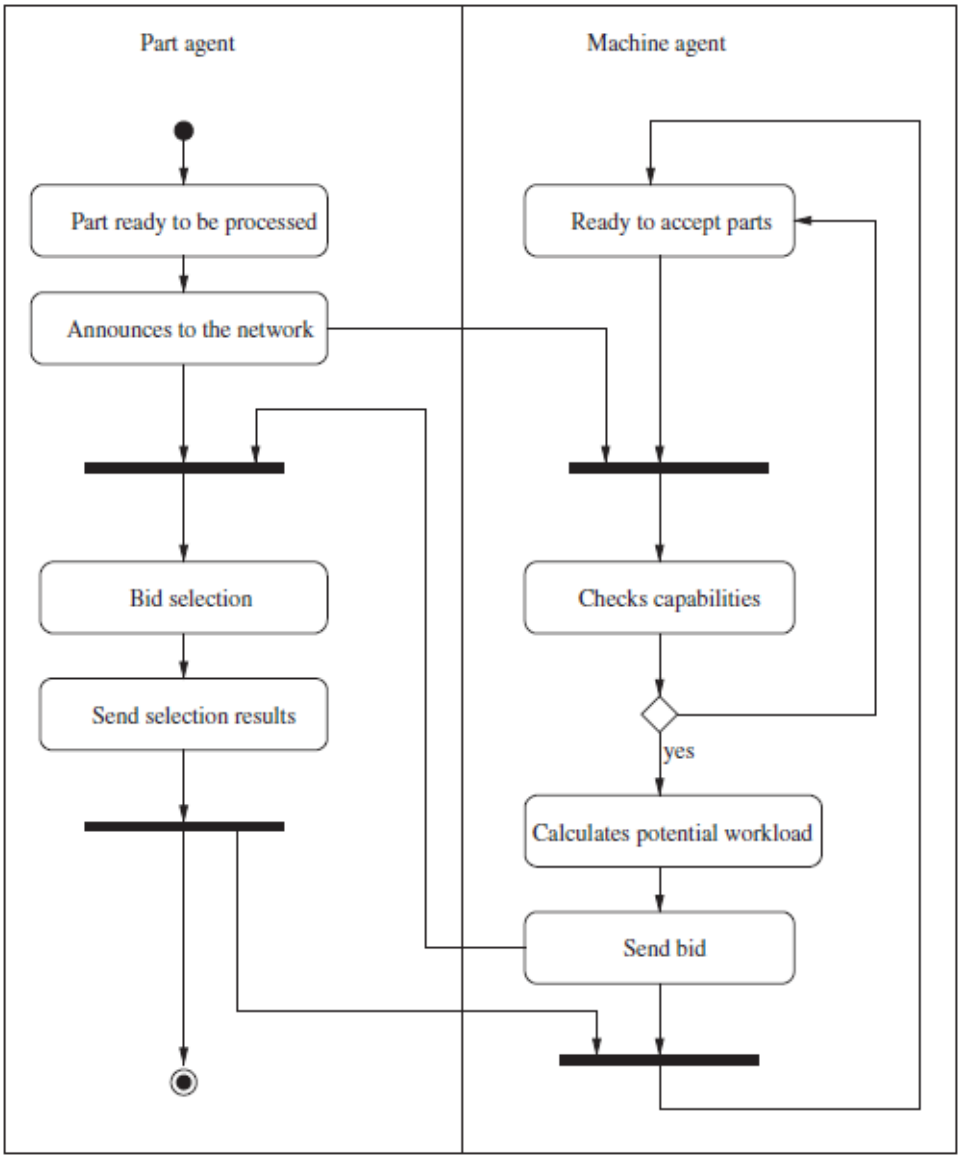

Figure 2 - UML diagram representation of an example of an application using Contract Net Protocol [21]

In this technique of negotiation, the Part Agent is the initiator of the negotiation and broadcasts to the other agents (in this example, Machine Agents) the requirements of the pending request, with a Call For Proposal (CFP) message. The machine agents check if they're able to do that task and reply with a bid message, which is a proposal from the Machine Agent to the Part Agent to take this task, taking in account parameters like estimated processing time. After receiving the bids, the Part Agent selects the Machine Agent that fits better in the requirements and send to him an award message [21].

This is the typical process of Contract Net Protocol, but some improvements can be made in order to avoid message flooding in the network:

- The Part Agent selecting only the appropriate agents to announce the negotiation, instead broadcast it to all agents (this will require, of course, a pre acquired knowledge of agents' capabilities);
- The agents who receive task announce only respond with a bid message if they're able to execute the task;
- The Part Agent only accepts bids during a fixed period of time. After that period it considers that the Machine Agents aren't able to execute his request and didn't send a bid message.

When developing a multi-agent system using Contract Net Protocol, the developers must be aware about the existence of deadlocks in their negotiation loops. If they exist, they could be result of wrong parameter estimation, mainly the fixed period of time that Part Agent waits for bid messages. This time must take into account how much delay exists on the network and how much time the agents take to respond and process messages [21]. To be able to do a simulation of negotiation protocols, towards resolving possible existing deadlocks, we can make a model outlined in Colored Petri Nets or other schemes-based mechanism [23].

Some multi-agent software frameworks [24] already implement ACL (Agent Communication Language) Messages to build messages and exchange them between agents, and the FIPA (Foundation for Intelligent Physical Agents) specifications for Contract Net Protocol and Interaction [25,26]. Following these standards it's easier to develop a scalable multi-agent system, able to communicate with external agents.

2.4. Multi-agent system applications

This paradigm allows the development of distributed, flexible, robust, scalable and reconfigurable systems. These features lead to the progress and optimization at various technological applications systems, including: Industrial/Manufacturing [18,19], Wireless sensor networks [27], Environment Monitoring [28], Surveillance [27], Robotics Systems (swarm robotics, mobile sensor networks, modular robots) [24,27], and services to the E-Commerce, Financial Sectors [28], Energy and Utilities sectors [29].

With a Multi-Agent System constituted by a team of multiple agents, each one with a specific skill or task, it is possible to build a network with shared information flowing between these agents, data filtering and analysis. Since this system is a distributed system, these agents can be positioned at different locations and make a wide-ranged monitoring. With these characteristics, it is possible to design systems with objectives like protection of an international border against trespassing, timely detection of a bushfire, accurate analysis of the traffic state of a city, remote monitoring of a vehicle carrying critical goods, and so forth [28].

Each scenario mentioned before requires different strategies and concerns to reach an optimal solution to the problem. In an Industrial/Manufacturing scenario, each agent can represent a machine, a robot, a processing unit or another physical resource [18]. In this system, managing agents that are capable of researching for the different agents and managing their availability taking into account the deadlines and the resources available, must also be present. Since the Manufacturing units have started producing more customized products, instead of mass production with all units being identical, it is a priority that scheduling plans for the production must be more flexible than an optimal pre-planned schedule. To achieve this, a multi-agent system is an approach that allows flexibility in production planning, due to the ability of agents to negotiate among themselves. With agents representing the resources of the industry, it is possible for them to adapt their production according to the desired requirement [19].

In an Energy and Utilities scenario, it is possible to design an intelligent system to manage energy resources like water and electricity. In this specific case there are designed agents with the purpose of controlling power generators, power stations and to monitor transmission lines and power consumers, as shown on Figure 3 [29]. With a distributed smart-grid, the energy grid becomes more robust and reliable. In both scenarios (Industrial and Energy) scalability and monitoring tools are really important to an efficient solution in these scenarios.

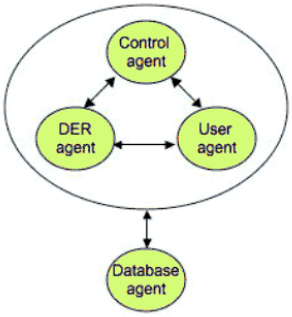

Figure 3 - Relation between agent classes [29]

In other scenarios like E-commerce and Financial services, multi-agent systems are associated with protecting critical resources and data, using tools to establish a secure communication only with reliable entities [28]. Since this kind of services is dealing with personal and confidential data of the entities involved, such as credit cards' details, personal identification, bank identification and others, it is crucial to protect this data. To achieve this, a multi-agent system can be useful to organize entities in a hierarchical way considering their competences and functionalities and creating agents to act as intermediate entities to confirm their trustworthiness. In this architecture it is also important to ensure the integrity and confidentiality of the messages exchanged by the multiple agents. To make this possible, encryption techniques are used in a way to minimize the effects on the system stability and messages delay. Taking into account this conditions, the method proposed to guarantee confidentiality and integrity of information exchanged by agents is symmetric key cryptography, that has a little impact on the scalability of the system, less than when public key cryptography is used [28]. There is a compromise between the level of encryption of the system and the speed of message exchange and the scalability of the system.

In Environment Monitoring, wireless sensor networks are used with many sensors distributed by different target areas and they are operating remotely, sending their readings to a centralized base station or sharing, with communication agents, that store and analyse the data. These sensor networks can be applied to bushfire monitoring, temperature monitoring, motion sensing for surveillance, and others [28]. In these networks, many sensors can operate simultaneously and it is necessary to make a fusion of all data read in the same area. This is useful to detect a malfunctioning sensor or glitches in some readings, improving the reliability of these

systems. This analysis can be processed in a centralized base-station or in distributed nodes responsible for each area. In case of an event occurs, for example: a fire, since all the sensors and nodes are connected, this event can be easily detected and located, in order to alert the local authorities.

2.5. Multi-agent Robotics System

Since Robotics Systems are evolving, from industrial robots that are only responsible for one task, performing it automatically and repetitively, to autonomous and mobile robots that can collaborate among themselves and use sensors to understand their operational context, it is necessary to develop new systems to take advantage of all these new capabilities. These robots are able to adapt in new environments and are ready to face changes in their goals and behaviors [4].

Within the robotic systems, mobile robots have evolved greatly in terms of intelligence and autonomy. Developments of mobile robots has focused mainly on the issue of location, sensory, visual and autonomous navigation, collaboration with other systems and development environment tools [30-32].

Taking this into account, it is possible to manage this new intelligent features and requirements, at robotics, with a Multi-Agent System. This paradigm can be applied to Robotics System in several forms:

- **Heterogeneous mobile robots**: where a team of mobile robots, where each member has complementary skills, has a constant communication with each other, for example, on a team of aerial fire-detection robots. When a member of this team detects a fire, it communicates to other members and decides through cooperative perception if it really is a fire or a false alarm. [24]
- **Robots working in ambient intelligence environments**: these robots take advantage of the information available in the environment where they belong. For example, a robotic assistant in an intelligent house in order to help solving housework problems or helping disabled people. [33]
- **Collective robot swarms:** are constituted by a homogenous collective of robots that are able to perform a simple and local task. Since they are a homogenous team, every robot has the same skill and makes this system very scalable and robust. They collaborate together to solve a complex problem, for example, exploration or surveillance. [13]
- **Mobile sensor robotics networks:** Unlike a fixed sensor network that is limited to retrieve information about a certain area, a network composed by

mobile robots equipped with sensors is a more flexible structure and can take an advantage of the coverage of monitoring. [24]

- **Multi-agent control systems:** In case of a very complex robotic structure with many different components, it is possible to implement a multi-agent control system to improve the collaboration and sociability between the different parts of the system. An example of this is a Humanoid or Biped robot [24].
- **Surveillance Systems:** From collaboration between software agents with autonomous mobile robots comes the possibility of developing intelligent surveillance systems, where robotic agents support the decision process of the system, making it more autonomous and smart [31,32]. This topic is discussed further in chapter 2.7.

The implementation of a Multi-Agent System in a Robotic System can bring many advantages to the performance of autonomous robotics systems, such as [24]:

- **Concurrence**: With a distributed system it is possible to have different processing units working at the same time, taking advantage of all the resources in the system. If the system has many concurrent processing units, it is necessary an introspection mechanism to make a monitoring of the system's performance.
- **Scalability:** In every distributed system, is important to guarantee its scalability, mainly in a mobile robotic team. In a well-designed multi-agent system, there must be no barriers to the coupling of more units to the robotic system. To achieve this, the software dependencies must be kept to the minimum, then it is possible to have a modular architecture where new units (similar to the existing units or different, with new skills) can be added to the system anytime, without an interrupting the process.
- **Robustness:** When a component of the system has an irregular behavior, this malfunction must not stop the whole system from working. Considering a distributed system and if this malfunction component is not an exclusive resource, the system must manage its resources and its new limitations in order to complete the respective mission. With more resources available, the more robust the system is, so there's more fault tolerance which provides a better guarantee to complete the mission.
- **Flexibility:** Whereas the above requirements are met, this approach presents a great flexibility which it is a fundamental requirement to a mobile autonomous robot that faces a dynamic environment.

However, a Multi-Agent System can act between a User and a Service provided by a robotic system in order to establish all the communication though the network, making all the process transparent to the user. To achieve this connection, a framework, proposed by Renato Vidoni [4], defines three classes of agents: Service Provider Agents, User Agents and Framework Agents. Each class of agents represents different entities of the system as show on Table 1.

Class	Service Provider Agents	User Agents	System Management Agents
Represented by:	Provider Agent Service Agent Robot Agent	Discovery Agent Costumer Agent	Framework Agent Broker Agent
Objective:	Manage the access to the services, establishing all the connections necessary and guaranteeing that the conditions agreed to the execution are met.	Interact with the client and identify which service best suits the request. Also returns to the client the result of the service. Costumer Agent represents the end-user in the system.	The Framework agent monitors all the activity in the system and messages exchanged and acts in case some unexpected event occurs. Also the framework agent provides tools that can help other agents.
Requirements:	Connection with Robot/Web services and other agents	Connection with the graphical interface that interacts with the client and other agents.	Connection with all agents in the system, introspection and monitoring tools

Table 1 - Description of each class of agents

All these agents get their individual objectives and their main objective is: make the client requests possible, while maintaining the integrity of the entire system. A representation of all these agents in the system is presented on Figure 4.

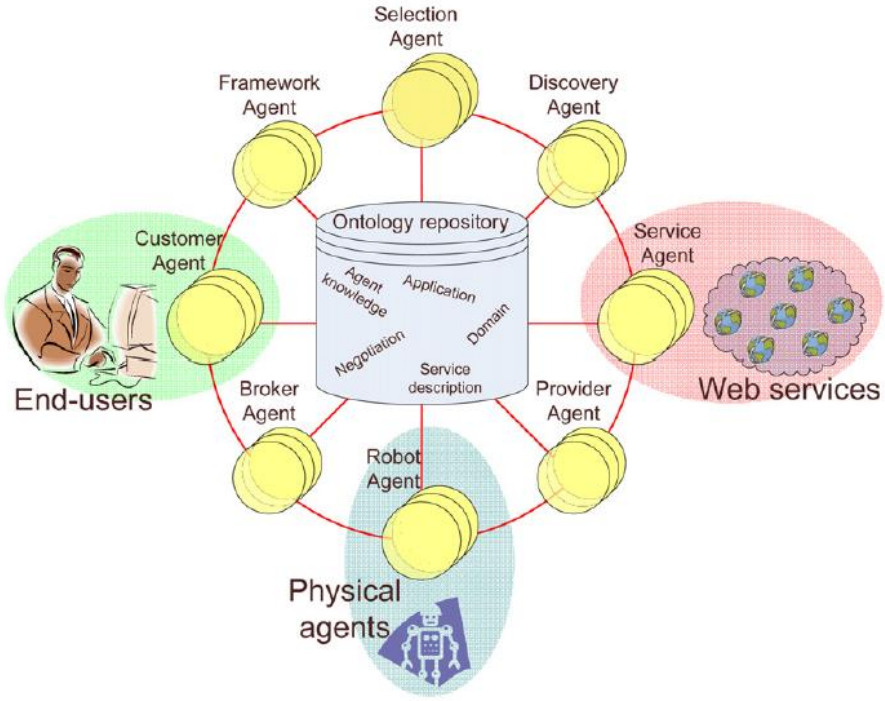

Figure 4 - Agent architecture proposed by this framework [4]

The Robot Agent is the agent that plays the role of the physical robot in the software system. With this agent, the services and missions performed by the robot are available to all the clients of this system. This agent is responsible to open/close the communication with the robot and manage their resources and state. Considering an autonomous mobile robot, this agent must contact the robot by a wireless network and must be able to translate the client requesting the robot's language. The transmission should end with a feedback from the robot informing on the result of the mission. A series of commands should be available to the Robot Agent, such as a command to check the status, location, resources and missions available.

2.6. Robotics and Multi-Agents Frameworks

In general, Robotic systems are controlled by Robotic Software Frameworks. These frameworks are focused on providing scalability, reusability, deployment and debugging of the software developed in the system. There are many Open-Source Frameworks available to the development of Robotic Systems such as: Player, OROCOS, ROS, YARP, OpenRAVE, OpenRTM, and others [24].

Despite the variety of frameworks that currently exist, these frameworks have been developed focused in the following areas:

- **Middleware:** for a distributed system; a tool that promotes the system's scalability, fault tolerance and makes development and integration easier.
- **Introspection and management tools**: these tools are very important since it is difficult to do debug in a distributed architecture, these tools also provide a way to monitor the status of the working system.
- **Development tools**: these frameworks aggregate different libraries and generic reusable robotic algorithms that facilitate the development of robotic systems. They also include compilers to the different languages in use and dependency managers.
- **Robot hardware interfaces and drivers**: that make possible the reusability of the high-level algorithms to different robotic devices, since the specific hardware code is saved in a configuration file.
- **Simulation and Modeling tools**: provide the possibility to do an early viability test before implementation, resulting in savings in time and money. An example of these tools are: OpenRAVE, Gazebo, Stage and Breve [24].

These frameworks also promote a layered architecture with many tools in each layer. It is possible to define three generic layers with regard to their level of abstraction: Application, Functionality and Hardware Abstraction, represented on Figure 5. The lower layers provide services to the upper layers and vice-versa through RPC mechanisms towards the system's flexibility.

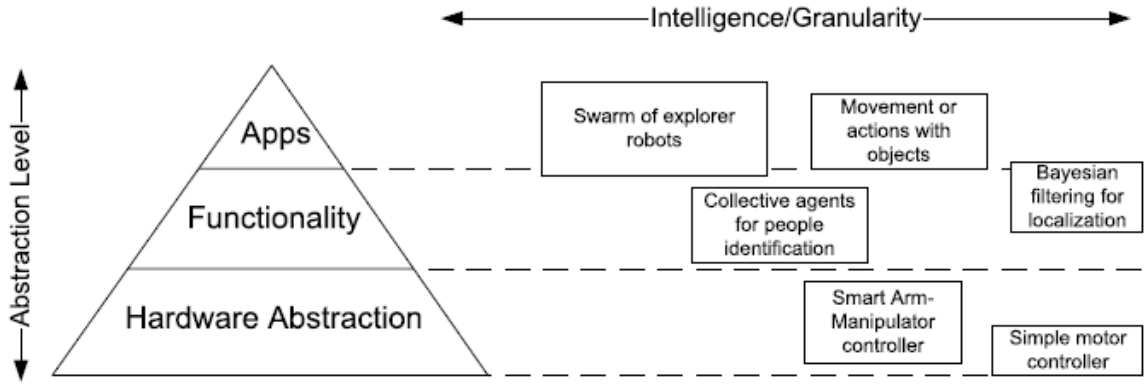

Figure 5 - Example of a layered architecture [24]

The multi-agent systems will be applied particularly in the application layer. With a Multi-Agent System implemented, the whole system can benefit from high-level development tools and a communication mechanism provided by these. Every Multi-Agent Software Framework got their own communication mechanisms that take care of all the communication concerns (sending messages, naming and lookup services). Other alternatives that Robotic Systems have to communication is implementing a custom communication mechanism or the implementation of a Middleware mechanism like CORBA or Web services [24]. Despite this, the Multi-Agent Software Frameworks can also intervene in Functionality and Hardware Abstraction Layers, in applications like Multi-Agent Control Systems, described before at Chapter 2.5.

Some Robotic Software Frameworks have enough capabilities to run a multi-agent system because they are able to deploy programs which can run autonomously and in parallel, each one with its own purposes and with communication infrastructure that can make each node connected to the others, just like the agent definition referred at Chapter 2.2. However, these frameworks don't provide services more oriented to the multi-agent systems because they are developed focused at Robotic and Hardware Systems [24].

Multi-Agent Software Frameworks, are frameworks that achieve the necessary conditions to support the operation of a multi-agent system, mainly in what concerns the communication mechanisms, like messages, ports, topics; naming and lookup services, agent mobility, development and introspection tools. Some of these frameworks are: JADE, Mobile-C or Zeus [24].

Table 2 and Table 3 show the main characteristics of several Robotics and Multi-Agent Frameworks where it is possible to make a comparison between these frameworks. This comparison was made concerning the following criteria:

- **Distributed Architecture**: this criteria is related to the network topology formed by the system's nodes.
    - In a **Purely P2P network**, all nodes are connected to each other, establishing a fairly robust and fault-tolerant communication, not being dependent on a coordinator node. However, this is not a very scalable solution, when we have systems composed of

many nodes, since each node needs to know all other nodes in the system.

  - In a **Centralized network**, the dependence on a central node in the network can be a point of failure throughout the network and requires all messages have to go through this element, overloading it and increasing the latency in communication.
  - In **Hybrid P2P**, this is a node that is known by all and first contacted for establishing the link between the communicating nodes. From that instant on, communication is done in P2P.

- **Node communication mechanism**: this criteria is related to the mechanism used to send and receive information.
  - **Simple Message**, it is the most basic method of communication where an incoming message by another node is sent.
  - **Ports**, this method allows nodes to read and write information into the ports, which can be interconnected across multiple nodes. The messages can be received by polling or callback.
  - **Topics**, is a method of asynchronous communication that follows the Publish/Subscribe pattern and allows multiple connections on the same channel. It is similar to the ports mechanism, however a port belongs to a node as a topic does not belong to a node necessarily a part of the system.
  - **Events**, this is an asynchronous communication in which a node communicates with subscriber nodes through a pattern observable/observer. The difference between this and "ports" is that in Events, node subscribers process the information received with callback.
  - **Services**, this method of asynchronous communication enables the remote procedure call where the client node sends a request message that specifies which procedure to run, and after the procedure has been completed, the server node returns a message with the result of this.

- **Properties**, in this mechanism, a set of properties is allocated on a node that might be consulted by other nodes. For each property, there are get/set operations. In hybrid P2P architectures, these properties can represent parameters of the system.

- **Naming, Lookup and Discovery Services**: are centralized elements on the network who know all the network elements and help these to communicate to each other.
  - **Naming Service**, also known as White Pages, allows finding a node in the network through its global name.
  - **Lookup Service**, also known as Yellow Pages, allows the network to find elements that are able to perform a particular service or have a particular skill.
  - **Discovery Service**, is an adaptation of the Lookup Service in a pure P2P architecture, where information about the services provided by network elements is found distributed over all network nodes. The Discovery Service is more appropriate than the Lookup Service when it is a scenario with connectivity issues and you cannot lose connectivity to a given node (the node that stores information in the Lookup Service).
- **Development and Deployment tools**: usually a set of tools are provided by the framework to help the developers to solve dependency problems and allow interconnection of system modules. Deployment tools consist of tools and configuration files that allow the deployment of the system, maintaining its scalability and complexity.
- **Simulation Capabilities**: "The simulators offer a visual representation of the problem and execution in virtual worlds of rigid solids."[24] Despite the fact that the frameworks presented are not focused on simulation systems, they have an easy integration with the best suited simulation tools, such as: Stage, Gazebo, Webots, and others.
- **Robotic Algorithms**: Some frameworks provide robotic algorithms for their use by developers in a generic and reusable manner.

- **Real-time capabilities**: In robotic systems, it is always important to take into account real-time constraints. Nevertheless, it is always a very sensitive issue given the operating system (OS's only like RTAI and RTLinux support real-time), communication protocol and libraries used. Nevertheless, in view of the development of an autonomous mobile robot, this restriction can be overcome by using libraries that allow the use of different communication protocols and different processing nodes.

Name	Distributed architecture	Node communication mechanisms	Naming Service	Lookup Service	Discovery Service
JADE	Hybrid P2P	Simple-messages, topics, complex interactions	Yes	Yes	No
Mobile-C	Hybrid P2P	Simple-messages	Yes	Yes	No
ROS	Hybrid P2P/Pure P2P (planned)	Topics, properties, services	Yes, pushing, remapping, rel./abs. addresses	Yes	No
YARP	Hybrid P2P/Pure P2P	Ports, topics	Yes, pushing, rel./abs. addresses	No	No
OpenRDK	Hybrid P2P	Ports, properties	Yes, remapping	Yes	No
OpenRTM	Hybrid P2P	Ports, services, properties	Yes	No	No
OROCOS	Hybrid P2P	Ports, services, events, properties	Yes	Yes	No
ORCA	Hybrid P2P	Services	Yes	No	No

Table 2 - Aspects of distributed systems of the compared frameworks [24]

Name	Development tools	Deployment tools	Simulation Capabilities	Robotics algorithms	Real-time capabilities
JADE	No	Deploy configuration	No	No support	No
Mobile-C	No	No	No	No support	Yes
ROS	Command-line	Command-line, deploy configuration	Stage, Gazebo	High support	no
YARP	No	Command-line, deploy configuration	Stage	Medium support	Yes
OpenRDK	No	Command-line, deploy configuration	Webots, USARIM, Stage, Gazebo	Low support	No
OpenRTM	Command-line and visual tools	Command-line, deploy configuration	Stage, Gazebo, OpenHPR	Medium support	No
OROCOS	No	Deploy configuration	No	Medium support	Yes
ORCA	Visual tools	Deploy configuration	Stage, Gazebo	Medium support	No

Table 3 - Other aspects of robotics frameworks [24]

It is possible to conclude the following regarding Multi-Agent and Robotic Software Frameworks [24]:

- **Mobile-C**: In this framework it is possible to implement agents with C++ with mobility.
- **JADE**: Is a framework with good acceptance by the scientific community, because in addition to their efficient naming and lookup services, it also implements the standards defined by FIPA.
- **YARP**: This framework has a strong development with regard to communication, presenting a robust distributed architecture since it supports flexibility on transport mechanisms and with the availability to implement a pure P2P architecture and an efficient message mechanism (topic-port).
- **ROS**: This framework has a large community of users, from which one can find very useful and well organized support documentation. It also

aggregates a wide set of robotic software and drivers. Since it is widely used in the scientific community, this framework allows the integration in different robotic systems.

- **OpenRDK**: This framework has implemented a concurrency model that distinguishes agents as processes and components as threads, a characteristic that is rarely implemented at the others frameworks.
- **OpenRTM**: This framework is a mature project developed by the Japanese National Institute of "Advanced Industrial Science and Technology", it has a large community in Japan with the inconvenience of most of its documentation not being translated into English and only available in Japanese. It offers various integrated development tools, like Eclipse, which facilitates the development with this framework.

2.7. Surveillance Systems

One of the main topics of technological development are security systems, with the primary purpose of ensuring the safety of people and their property through devices such as sensors and surveillance cameras for monitoring different areas (private areas, buildings, international borders, etc.) [36] .

These systems have evolved over time, and it is possible to distinguish three generations of surveillance systems:

- **First generation**: the first surveillance systems were composed of a closed-circuit video (CCTV), composed of several cameras scattered around the location to be monitored. With the use of a switch that received the signals sent by the cameras, it was possible to view one camera at a time on a display or several cameras in different displays. Such systems were totally dependent on humans because the whole process of detection of dangerous situations and events was done by a security guard who visualized the images captured by the cameras [5].
- **Second generation**: Due to high human dependence for the events detection by viewing the images captured by the cameras, the second generation of surveillance systems developed algorithms to make them more autonomous systems in detecting events. These algorithms are able to analyze the captured images, and through patterns recognition,

background subtraction and silhouette separation techniques, detect situations of disaster, accidents and intrusion [37,38].

- **Third generation**: Since there was a need to monitor increasingly larger areas, surveillance systems began to be used based on distributed platforms, unlike the first and second generation systems, which were based usually on centralized system platforms. Some of the platforms used are multi-agent platforms where it is possible to distribute different agents to monitor the area [35].

Systems of first and second generation run in a centralized manner, where all information captured is sent to a single processing unit, which triggers alarms depending on the analyzed information. This central point is critical in architecture and when subject to failure can it jeopardize the correct operation of the entire system.

The third generation systems, to improve this aspect, present a distributed architecture, more tolerant and less overloaded with information to analyze.

However, despite the improvement in automatic detection of events in the second and third generation systems, research continues to be made towards the detection of events become increasingly autonomous and effective, depending less and less on human intervention. To improve this shortcoming, robotic units have been introduced in the surveillance systems to give greater autonomy and support to the human elements that are responsible for these systems, as referenced in the projects [27,6].

3

3. Architecture

This chapter presents the system's architecture that was used to the implementation of this dissertation. The described work in MAS's (chapter 2) assumes that all the agents are developed in the same framework, using the problem scenario for developing a multi-agent system for the resolution of their requirements. Nevertheless, there are systems that could improve their features and performance by interacting with "agents" from other systems. None of the cases studied did an integration of different systems to address a new problem or improvement of the system.

If instead of the development of a new system, the new features were integrated an already developed system, it would save on cost of development of a new platform and this system could be reused in other scenarios. To achieve this, it is necessary to guarantee the communication between the different elements of the architecture, using a communication protocol that could be understood by both elements.

So, since the objective is to integrate a robotic device onto an existing system, it becomes necessary to develop software structures that represent the features of this device in the system and a communication protocol. These structures are software agents that will be added to system to collaborate and interact with other agents in the system.

3.1. Communication protocol

In this scope, is necessary a communication protocol so that the systems can effectively communicate and guarantee that all the functionalities are available. This communication protocol needs to be based on a middleware framework that guarantees the connectivity and a message protocol that both systems could interpret.

With a middleware framework, both systems are able to send and receive messages using different patterns: Request/Reply (REQ/REP) and Publish/Subscribe (PUB/SUB). The REQ/REP pattern is used when a request is made and the sender needs an answer from the receiver. A PUB/SUB pattern is suitable when a system is sending information (publisher) that could receive by another one or more systems (subscribers), without the need of an answer/acknowledge from the receiver.

3.2. Message Protocol

With the communication established thanks to the middleware framework, it is necessary that both systems use messages that both can interpret. In this context, it is necessary to develop a messaging protocol for this purpose, where both systems can make requests, get replies and obtain information about the system in general.

A communication protocol using a middleware framework and a messaging protocol was developed for this purpose and presented in [39].

3.3. Driver

For both systems to interpret and use efficiently all functionalities guaranteed by the protocol messages, a driver, who can provide these services at the level above (software agents), is required. This driver is the object responsible for creating, reading and validating of the messages defined in the communication protocol and will be able to send them in different patterns: REQ/REP and PUB/SUB.

3.4. Manager Agent

Since this architecture is intended to integrate external elements into a surveillance system, a software agent that manages these devices in the system is necessary. Therefore, this agent must receive requests, such as requests for registration, deregistration and list of devices available in the system and make their structures available in the system so that it works correctly.

3.5. Sensor Agent

A sensor agent will be responsible for handling all the sensory information in the system, i.e., is the agent that will capture the values read by the sensors of the robot and integrates them into the system along with the remaining values of other sensors. In order to gather the values read by the robots' sensors, this agent must use the respective driver.

A sensor agent has the following properties:

- **location coordinates** - With a geolocated agent, it is possible to know the location that the read values belong in this way it is possible to detect geolocated events;
- **agent subtype** - should specify the type of data that the agent will obtain (temperature, humidity, light, etc.);
- **frequency data** - the frequency with which this agent will get the data of the robot;
- **driver** - structure that will allow the agent to obtain sensor values read by the robot.

This agent will have the following behaviors for proper functioning: behavior for data generation and behavior for sending data to other agents.

In the behavior for generating data, the driver is used for, cyclically data reading, by the robot sensor, at a specified rate in the agent properties. In this behavior there is also a data processing towards the integration of these values read into the system, adding to these an unique ID and the agent's location coordinates, transforming the values read into parameters.

In the behavior for sending data to other agents, the parameters will be passed to the other agents who treat the remaining system integration and event detection.

3.6. Mobile Agent

To use a robot for events detected by the security system, another type of agent is needed. This agent will have a driver to be able to communicate with the robot, to know its status (it is available to respond to events), which sensors has available, for the purpose to know what kind of events you can use this mobile robot.

In this dissertation, the mobile robot will be considered to respond to requests for events that are to be confirm by the state. Upon receiving a request, the robot can send a "acknowledge mission" message to confirm, cancel or hold for confirming the respective event. A "acknowledge mission" involves two phases: a first phase where the robot has to move to the location of the event, and a second phase in which it sends the values read by the sensors, when in event location, for the agent and this may confirm whether or not the event.

A mobile agent has the following proprieties:

- **driver** - structure that will allow the agent to obtain the sensor values read by the robot, make requests and get replies.
- **location coordinates** - With a geolocated agent, it is possible to know the location where the robot is. In this way, when this agent receive a CFP message, it is possible calculate the distance between this agent and the event and send it in the event proposal message.
- **sensor list –** With a list of sensors available on the robot, it is possible for this agent to determine what kind of events this robot can respond.
- **Event configuration file** - A configuration file of events for easy configuration of the behavior of this agent face to requests for response to events is required, as shown in Figure 6. In this file is defined for each type of event, the type of sensor that validate this type of events, the criterion used with this type of sensor and its limit to confirm or not the event.

```
<?xml version="1.0" encoding="UTF-8"?>
<root>
  <event>
    <!-- EVENTO DE FOGO -->
    <type>1</type>
    <metaParameter>Temp</metaParameter>
    <criteria>valueAbove</criteria>
    <thresold>40</thresold>
  </event>
  <event>
    <!-- EVENTO DE INUNDAÇÃO -->
    <type>2</type>
    <metaParameter>H2O</metaParameter>
    <criteria>valueAbove</criteria>
    <threshold>90</threshold>
  </event>
</root>
```

Figure 6 - Event Configuration file example

In summary, all these elements (communication protocol/middleware framework, Driver, Agent Manager, Sensor Agent and Mobile Agent) contribute to the integration of an external device into a surveillance system, namely a mobile robot, in the sense that system power benefit all the advantages of this new element in your system, as shown in Figure 7.

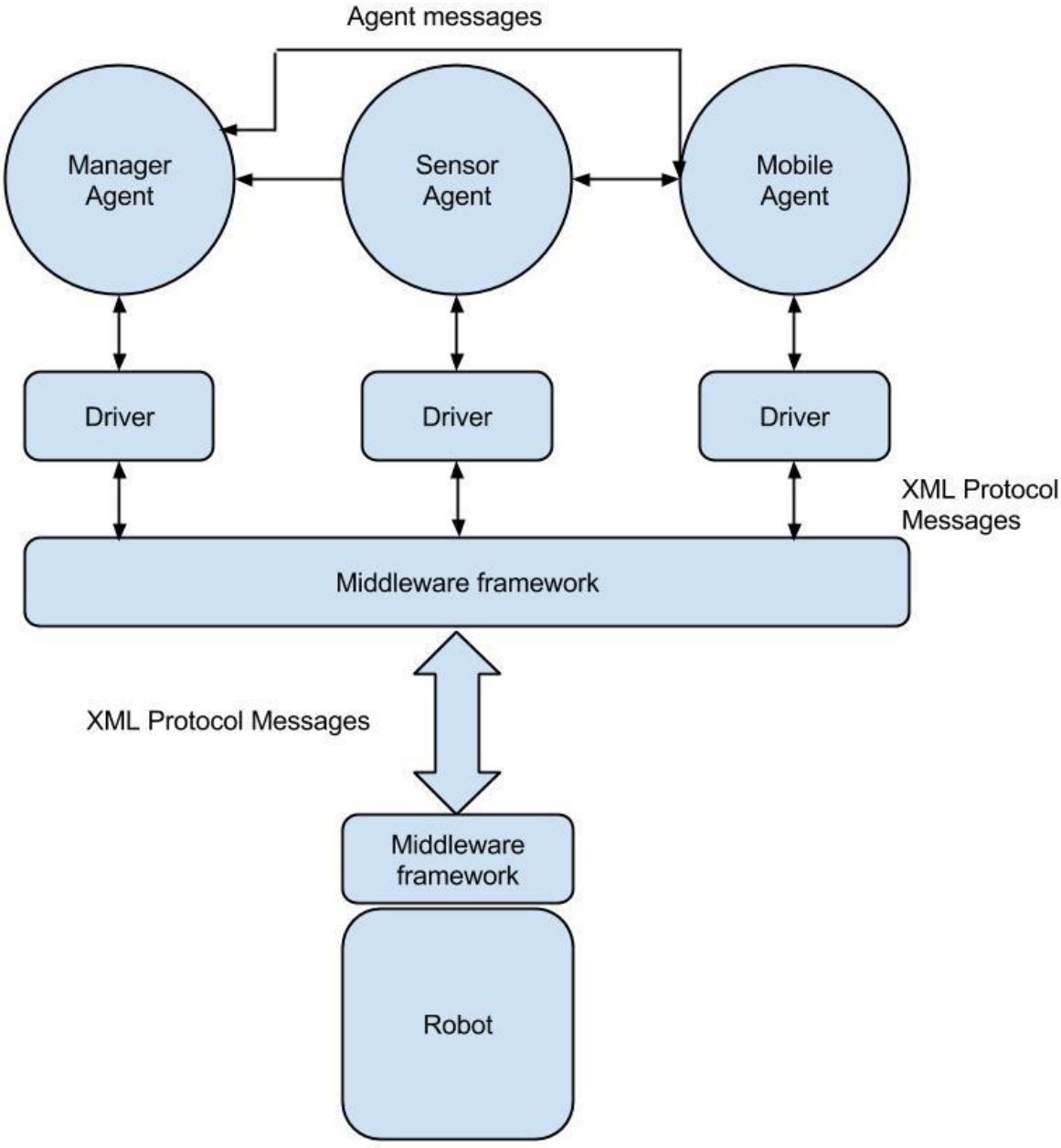

Figure 7 - Architecture elements

4

4. Implementation

To implement the system architecture presented at chapter 3, two different systems are used: DVA, an intelligent surveillance system and ServRobot, a mobile service robot.

DVA developed an intelligent surveillance system based on a multi-agent platform, using JADE framework [24]. DVA's system supports different sensors types and implements mechanisms of geo-referencing sensor's data and events. This distributed system implements different agent's types, such as:

- Sensor agent – provides sensor information;
- Processor agent – transforms sensor information into parameters;
- Inference agent – uses parameters in rules for event detection;
- Action agent – executes predefined actions for each event type;
- Backup agent - stores all the system information;
- Interface agent – shows (in maps) the values and locations of the sensors, events, actions and the system status;
- Mobile agent – Associated with a human, equipped with a mobile device who is responsible to perform events' actions, such as confirming the event or handling the event;
- Monitor agent – monitors all system's agents, ensuring correct system performance.

Figure 8 presents the elements of the DVA system and their major interactions.

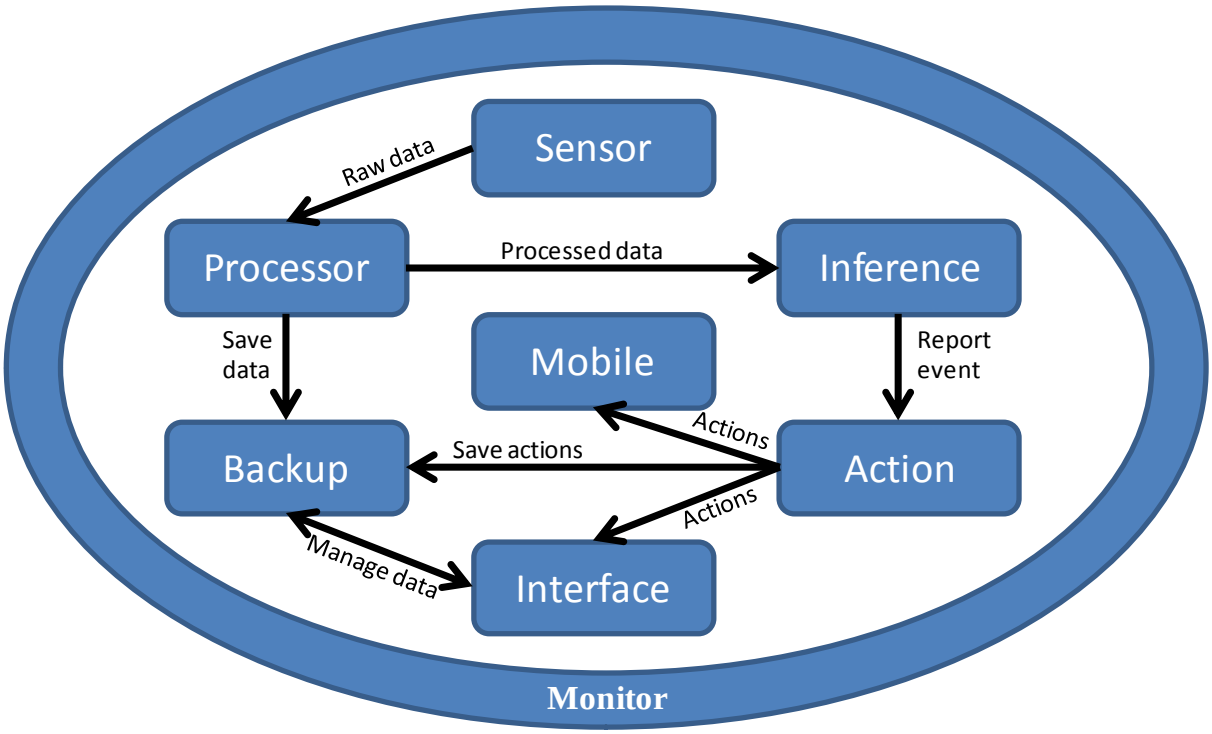

Figure 8 - DVA's agents topology

Furthermore, ServRobot is a server/client system based on ROS framework [24], where different software modules are implemented allowing inclusion of new features as an autonomous service robot. Is capable of: following people or lines, teleoperation, execute predefined missions (such as: go to a specific GPS position) and obtain sensor data.

Figure 9 - ServRobot service robot

Both projects are operating independently of each other and it is not possible to include ServRobot as agent of DVA's system or use its sensor data. In this dissertation, a new architecture is proposed that extends DVA's architecture to allow ServRobot integration without compromising the operation of these systems separately. This way, ServRobot and DVA can continue to function as it is currently implemented differing only in the ServRobot's role, this being an agent or a sensor, available at DVA.

The integration of ServRobot in the DVA's system, as a Mobile Agent and Sensor Agent, could significantly improve DVA's performance, using it in the event's detection, fusing its sensor data with the DVA's traditional sensors, improving their inference engine, event's validation and event's handling by using robot mobility and its environment perception; and it is expected that this proposed integration will improve intelligent surveillance system, reducing false positives, using robot's sensors to complement DVA's static sensors improving the reliability of events triggered and the need for human's interactions using the robot mobility to confirm and handling events.

In summary, through this extended architecture, the DVA system will be capable of: sending a robot agent to execute a mission; get robot's sensors data; ensure the mobile surveillance of different areas; send a robot to confirm events, getting feedback from the environment. This architecture also makes possible for new devices to enter in the system as client devices, capable to teleoperate the robot or to get feedback from their sensors.

4.1. Communication protocol

In this scope, it was developed in this dissertation a communication protocol for the DVA's system to effectively communicate with ServRobot and guarantee that all the functionalities reported before are available. This communication protocol needs to be implemented over a middleware framework that guarantees the connectivity and uses a message protocol that both systems could interpret.

4.2. Message protocol

The language proposed for the exchanged messages, is the Extended Markup Language (XML). XML was adopted taking into account its advantages to: modulate the concepts of the scenario in study (instead a byte codification); make changes in the message protocol by modeling new objects and data types [40]; developed in

different platforms, debug problems and validate the messages' composition [41], [42].

Also, with XML an important issue to this architecture is guaranteed: communication interface does not contain limitations, so in future, new functionalities can be added easily with scalability, for new sensor/modules in the autonomous robot or new devices in the system [24].

The message was defined using XML tags. Messages are initiated by the tag <msg>. This element must have a child elements *Header* (that defines the type of the message) and could have a *DataFrame* child element if necessary. *Header* contains receiver's identification (*DestinationID*), session used to send the message (*SessionID*), message identification (*MessageID*), sender's identification (*SourceID*) and when this message was sent (*Timestamp*). The *DataFrame* can be of different types (five *DataTypes* explained bellow) and each has its type of *Data* as a child element. [34]

With this message protocol, it is possible to use different message types:

- Emergency Command - Used to abort an activity that is being undertaken. Independently of the current status of the robot, when it receives an emergency command, the robot should stop immediately.
- Heartbeat Message - The heartbeat message is used as an “I’m alive” type of message. It is sent using a PUB/SUB messaging pattern and is present in all devices in the architecture.
- Registration Message - To join the system, the devices have to send a registration request to the "Robot Manager" agent at DVA.
- Simple Message - Simple Messages are regular messages defined in this protocol. These messages have a *DataType* that specifies the type of data contained on the message to send and can take values such as: *Robot Status, Mission, Sensor, Device Subscription* and *Reply*.
    - *Reply* type is used as an "answer" to every request made from DVA agent’s to the robot.
    - *Mission* messages are used to request execution of missions to the robot and get feedback from the result of an executed mission. Different mission types are available, taking into account the

robot's capabilities. The mission arguments can be already stored at the robot's memory or sent in the message as *Data*. It is also possible get a mission list available from the robot's memory or to get more detailed data about one specific mission. This way the DVA agents know the robot capabilities available. It is possible to upload and remove missions on the robot's memory.

- *Sensors,* using these messages it is possible to get information about the robot's sensors, for example: get the list of all the sensors or get the detailed information of a specific sensor, as represented at Figure 10. This way, the DVA agent knows what kind of data it can gather from the robot. With these messages it is also possible for the DVA agent to subscribe or unsubscribe to specific sensor values that are published by the device. Those values are published with a certain "Update Interval" requested by the device when it subscribes it. There are other methods to get values from sensors following a REQ/REP, obtaining the sensors information in terms of a specific parameter (for example a temperature of the left motor) or type of parameter (for example all the temperatures from the robot's sensors).
- *Robot Status*, with this message, the DVA agents will be able know the robot's availability and the operation mode (Idle, Mission Mode or Teleoperation Mode).
- *Device Subscription* messages are used to get a list of devices connected to DVA system. It is also possible to use these messages to initiate a new session on a device with a certain permission level.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<msg>
  <SimpleMsg>
    <DestinationID>ROBOT001</DestinationID>
    <MessageID>165326542313525DVA</MessageID>
    <SourceID>DVA</SourceID>
    <Timestamp>165326542313525</Timestamp>
    <SessionID>DVAsession</SessionID>
    <Sensor>
      <GetSensorInfo>
        <SensorID>1</SensorID>
      </GetSensorInfo>
    </Sensor>
  </SimpleMsg>
</msg>
```

Figure 10 – "Get Sensor Info" command message example

4.3. Proposed implementation

As stated above, with the integration of ServRobot at DVA, new functionalities become available in this intelligent surveillance system, including the integration of ServRobot's sensors in DVA, adding these readings with readings from traditional DVA's sensors. Another possibility is also the chance to use the ServRobot's autonomous navigation and send missions to accomplish this. Given that at the DVA system all events are geolocated, it is possible, when an event occurs, to send a acknowledge mission for the robot to check what happened in that location (to confirm if the event is real or if it is a sensor error).

This way, the DVA may use ServRobot in two roles: as a **mobile sensor** and as a **mobile agent**. To achieve this objective, it is necessary to implement on the DVA the appropriate structures to communicate with the robot and to do all the management other information being sent and received for this purpose. Therefore, in this proposal there is an information flow as represented the diagram in

Figure 11, where three new elements (*RobotSensor*, *RobotMobile* and ServRobot) are added in the DVA architecture.

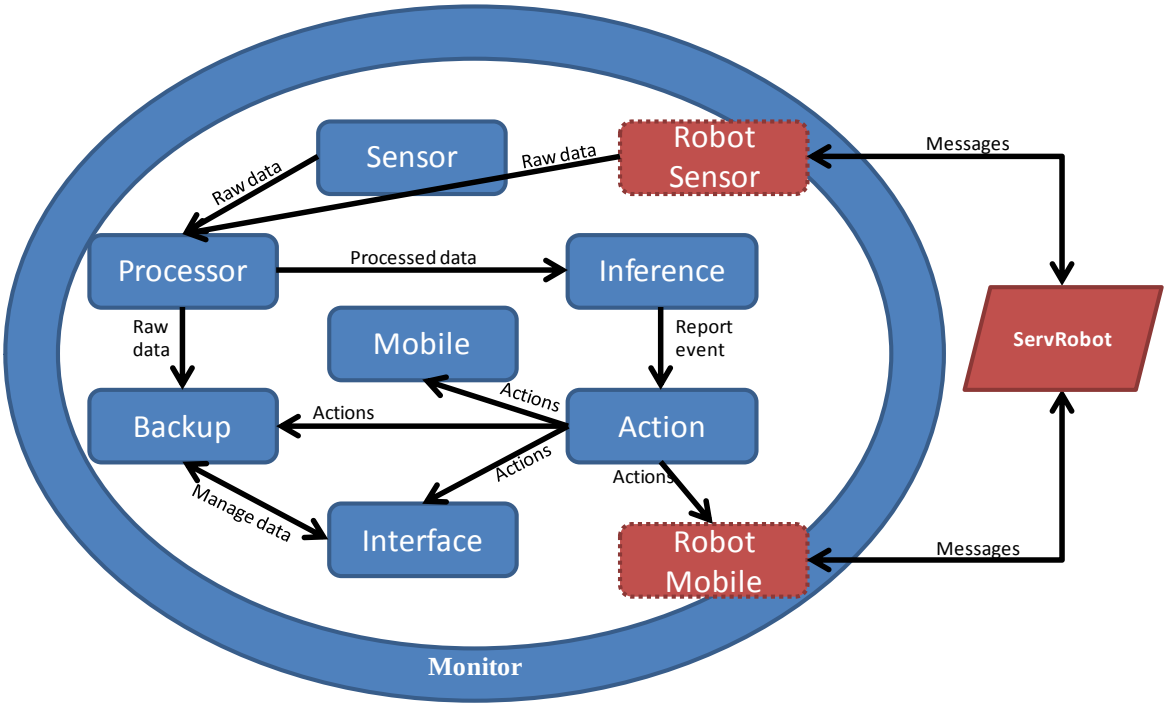

Figure 11 - Proposed architecture

To this purpose the following data structures were implemented at DVA: *ZMQDriver, RobotManagerAgent*, *RobotSensorAgent*, and *RobotMobileAgent*.

ZMQDriver

This is the module responsible for the communication process between the robot and DVA's agents.

In this module the following has been implemented:

- Classes for sending and receiving messages in different patterns (REQ/REP and PUB/SUB), using the methods implemented by ZeroMQ [43]. In a context of REQ/REP, the server and the client are not blocked waiting for a response. For this, a mechanism to timeout (the server) and polling (the client) to overcome this block were implemented. In a context of PUB/SUB, it is possible to determine the frequency at which information is being published and who will receive this information.
- Classes for creating, parsing and validation of protocol messages using classes generated by JAX-B [44], whose source is an XSD (XML Schema)

file which defined the structure of messages of this protocol. With this XSD file is possible to identify any irregularity in a message.

- Class that defines static and constant values that should be remain unchanged during the communication process (name tags, ports values and other elements).

This module is present both in the DVA system and the ROS system that operates in ServRobot, being essential to maintain the consistency and efficiency in the communication process.

RobotManagerAgent

This is the agent responsible to handle the registration process of external devices at DVA system. This agent runs in its setup the *RegistrationBehaviour*, which uses the *ZMQDriver* to create a server to respond to registration requests made by other devices using *ZMQDriver* with the message protocol described before.

According to the device you want to register in the system, the agent will take the initiative to integrate it with the rest of the system. If the new device is:

- **A mobile robot equipped with sensors**, this agent will create a new *RobotSensorAgent* to receive all of this robot sensory information and integrates it in the DVA system so that information gathered by this can be accessed by users and also to contribute for the events' detection. *RobotMangerAgent* will also create a *RobotMobileAgent* that provide this robot for the allocation of "acknowledge missions" or "operation missions" (taking into account their skills) if an event is detected nearby;
- **A teleoperation device** operating in a smartphone with DVA's mobile application, this agent will notify the device about which robots are available for teleoperation.

This agent also handles the device deregistration process with *DesregistrationBehaviour* added at its setup. A device must be deregistered when its operator shuts down the device or if the DVA operator wants to disconnect this device from DVA. When the device shuts down it must send a *Desregistration* message to this agent.

RobotSensorAgent

This is the agent responsible to handle all the sensory information received by the robot. For this, each agent has an associated *ZMQDriver*, connected to the correspondent robot, to receive sensory information gathered by the corresponding robot and a *ProducerRobotDriver* that transforms this information into parameters to be integrated into the DVA system. *ProducerRobotDriver* extends *ProducerDevice* DVA's class.

After being created, the agent initiates communication with the robot with the intention of knowing that it has sensors. For this, messages with *DataType Sensor* (*GetSensorList* and *GetSensorInfo*) are sent, and after that, the agent's *ZMQDriver* can access the values published by the robot.

Beyond traditional sensory information, this agent also collects the GPS position published by the robot. This data must be gathered separately from traditional sensory information because every DVA sensor parameter must have a location. This way, GPS position it is not considered a sensor parameter but an agent and a parameter's property. To gather GPS position separately from other sensors, it is created another instance of *ZMQDriver* to do it.

The interaction between these modules and agents is represented at Figure 12.

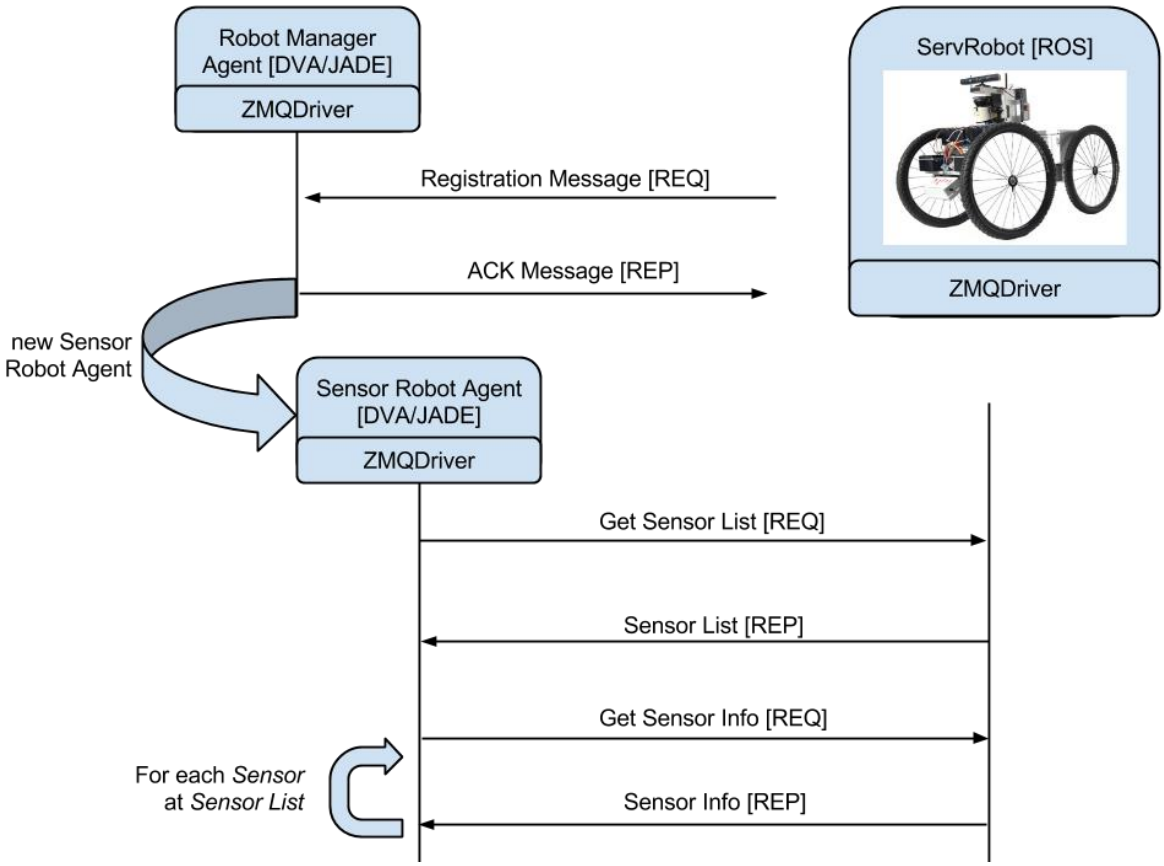

Figure 12 - Communication diagram between DVA agents and ServRobot - Registration (1)

RobotMobileAgent

This is the agent responsible to process the events attributed to the robot. This agent is created as soon as the *RobotManagerAgent* completes successfully its registration process, as showed at Figure 13. This agent can only be created after a login is made in the DVA web application. Otherwise, this agent could not perform any event related action at DVA system.

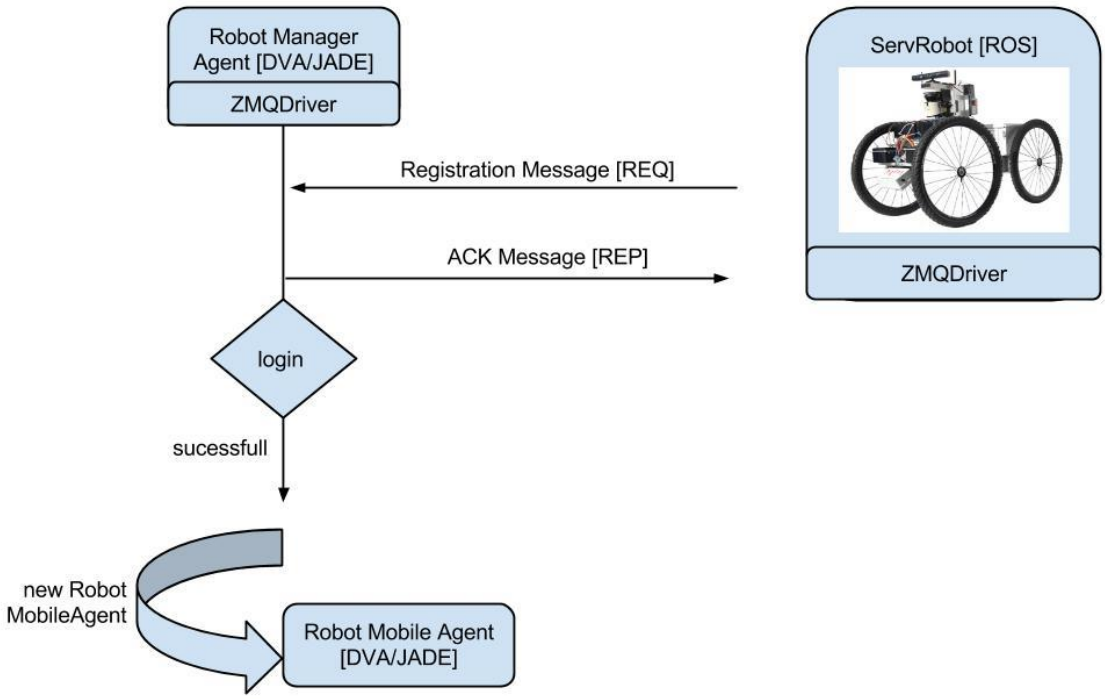

Figure 13 - Communication diagram between DVA agents and ServRobot - Registration (2)

This agent runs at its setup four main behaviours: *GetPositionBehaviour, ReceiveSensorListBehaviour*, *CheckRobotStatusBehaviour*; and the following methods: *setupEventProposalResponder* and *acknowledgeMission*.

With *setupEventProposalResponder* method, this agent is able to receive a *CFP* request message from an *ActionAgent*. This *CFP* message contains events' details that *RobotMobileAgent* will use to handle this event and reply with an event proposal, only if the robot is able to handle this type of event and available to perform a mission. This agent loads the configuration file *RobotEventConfigurationFile* at its setup and when it knows which sensors the robot has, it builds a list with the types of events that are able to respond.

This event proposal contains the distance from the agent to the event. After sending this proposal, this agent will wait for a inform message from *ActionAgent* that means the event it's attributed to this agent. This way, *RobotMobileAgent* will analyze the event status and decide which command can send to the robot. If the event status is *TO_CONFIRM*, means that the event has not been confirmed by any DVA agent (*Interface*, *Mobile* or *RobotMobile*), in this case, *RobotMobileAgent* will

execute an "acknowledge mission" with the respective robot, with *acknowledgeMission* method. An "acknowledge mission" is to move the robot to the event's location and when it arrives there, read the related sensors to confirm or not the event.

To perform this, *RobotMobileAgent* uses *GoToGPSPosition* and *GetMissionStatus* messages to interact with the robot. An event is confirmed if the values of the parameters read by a sensor show the behavior of the same event. The parameters of this verification are in a configuration file (RobotEventConfigFile.xml), where the criteria which the agent will follow to confirm whether or not an event is. For example, in case of a fire event, the *RobotMobileAgent* will use the temperature sensor of the robot to confirm whether or not the event. If the value read by this sensor exceeds 40ºC, confirms the event. Otherwise, the event will continue to be confirmed or will be cancelled. This threshold value can be changed at the configuration file.

4.4. Interaction between agents

Since *RobotMobileAgent* and *RobotSensorAgent* are integrated in the same device, they need to share some information. Thus, it is not necessary that the two agents are communicating simultaneously with the robot, exchanging the same type of information. So, the robot handles less requests, being that all the processing and distribution of such information is on the side of DVA agents, not congesting the connection between the robot and the DVA.

These agents need to communicate with each other through ACL messages. In order for them to distinguish these messages from other messages of the agents, the Ontology property of an ACL message is used. For this purpose, the GPS and SENSOR ontology labels were defined to identify messages used to exchange this kind of information.

At *RobotMobileAgent*'s *GetPositionBehaviour*, this agent receives an *ACLMessage Inform* message, using *GPS* ontology, from the *RobotSensorAgent* (that handles the sensor information from the respective robot) with the current GPS coordinates that the robot published, and stores this position internally, as shown in Figure 14. This data is needed when *RobotMobileAgent* receives a *CFP* message from *ActionAgent* and wants to make a proposal to attend a certain event. All the event proposals must include the distance from the agent to the event so that the *ActionAgent* can make a correct decision.

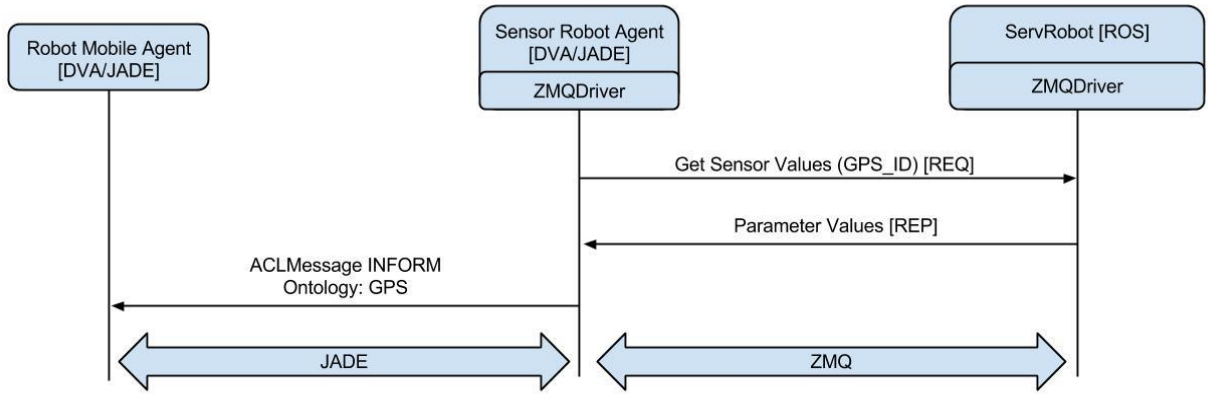

Figure 14 - Communication between DVA agents - GPS

At *RobotMobileAgent*'s *ReceiveSensorListBehaviour*, this agent receives a message from the *RobotSensorAgent* with the list of sensors (meta-parameter and parameter id for each sensor entry) that the device have, as shown at Figure 15. This agent will use this information to know what type of events this robot is able to perform an "acknowledge mission" and which parameter ids will use at *ZMQDriver* to get parameter values from the robot. For example, if a robot is equipped with a temperature sensor, *RobotSensorAgent* will send an *ACLMessage* Inform message, using *SENSOR* ontology, to the corresponding *RobotMobileAgent* with an *Hashtable* with an entry [Temperature; PARAMETER_ID]. This way, *RobotMobileAgent* knows that this robot is able to do a acknowledge mission to Fire Events, using its temperature sensor to confirm or not the event (according to the configuration file shown in Figure 6).

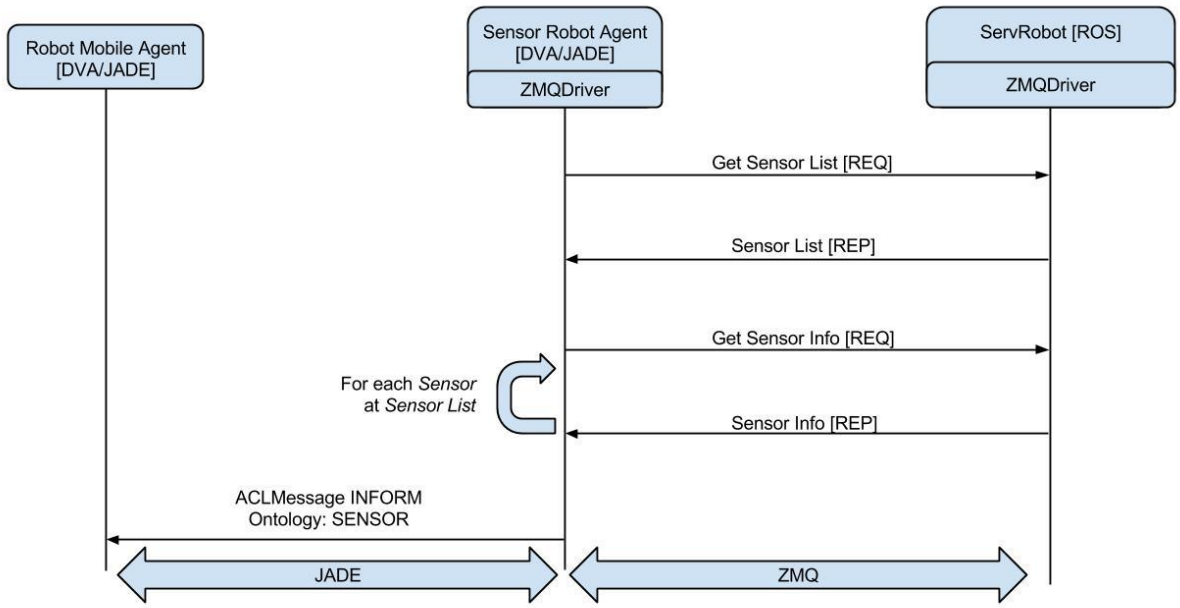

Figure 15 - Communication between DVA agents - Sensor

At *CheckRobotStatusBehaviour* this agent will cyclically ask to the robot what is its operation status, with a *GetRobotStatus* message, as it is represented at Figure 16. The robot replies with a *RobotStatus Reply* message with all the information about their current status. This agent will only use the *Operation Mode* and *Battery* data. With these data, the *RobotMobileAgent* can determine the availability of the robot to respond to events. If the robot *Operation Mode* is *Available*, it means that this robot is not performing any mission or not being teleoperated, so it could be available to perform a mission. Another criterion for assessing the availability of the robot is its time of operation that is available. If the robot's battery is running at low energy, the *RobotMobileAgent* will not consider this device available to perform a mission.

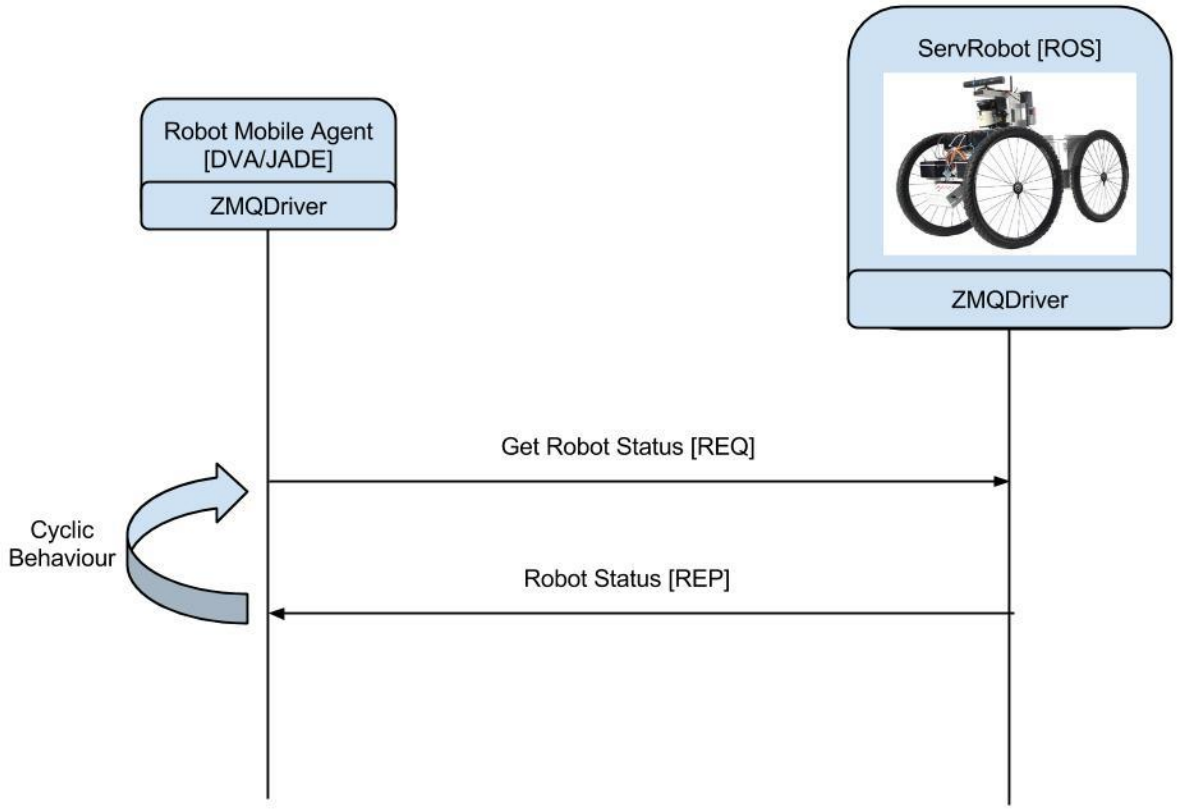

Figure 16 - Communication diagram between RobotMobileAgent and ServRobot

5

5. Results

After the development of the software structures referred in chapter 4.3, followed the testing phase to validate and analyze the results obtained in simulation and experimentally. To perform these experiments, a test laptop Acer Aspire 5737Z was used with an Intel Core 2 Duo 2GHz, 4GB of RAM, running Windows 7 Professional 64-bit.

Experimental tests have passed through different phases. Initially, the multi-agent platform was launched locally on the test laptop, with the robot simulator in the same laptop – Simulation 1. At this phase it was possible to test the robot integration as a sensor, this because in the test computer there was no access to all the remaining structure of the DVA detection of related events.

Then, those agents related to the integration of robot were executed on the test laptop but were launched in the distributed multi-agent platform of the DVA – Simulation 2. After this phase, the robot simulator was replaced for the robot itself, communicating directly with ServRobot system – experimental test.

5.1. Simulation 1

Before making an experimental test involving all the hardware of the robot, several simulations to test the robustness of the messaging protocol developed were performed. For this a Java Application that simulated different possible behaviors of the robot was developed, in the framework of detecting the existing weaknesses. This Java Application's main method sends a Registration message to the *RobotManagerAgent* and starts Java Threads to publish sensor values and to answer to

DVA agents requests. In this simulation, this Java application simulates the robot equipped with a temperature sensor and a light sensor, and it uses a *ZMQDriver* to communicate with the DVA system.

With this simulation it was possible to evaluate the behavior of *RobotManagerAgent* face to requests for registration of the robot and other devices. It was found that, after receiving a registration message, the *RobotManagerAgent* handles the request depending on the type of device registered, as expected. In the case of a robot, the *RobotManagerAgent* launches a *RobotSensorAgent* and *RobotMobileAgent* as stipulated in chapter 4.3. In the case of another device, a request from teleoperation device was tested, the *RobotManagerAgent* registers this same device and does not perform any further action.

It was also confirmed that the *RobotManagerAgent* effectively responds to requests for lists of registered devices.

At this stage of simulation it was also possible to evaluate whether the *RobotSensorAgent* is operational to receive the data sent by the robot simulator and integrates them in the DVA system. With the result presented at Figure 17, it is possible to conclude that the *RobotSensorAgent* is correctly processing sensory values of the robot, generating parameters with the value of 60ºC for the temperature parameter (*Param[1]*) and 5W for the parameter of light (*Param[9]*).

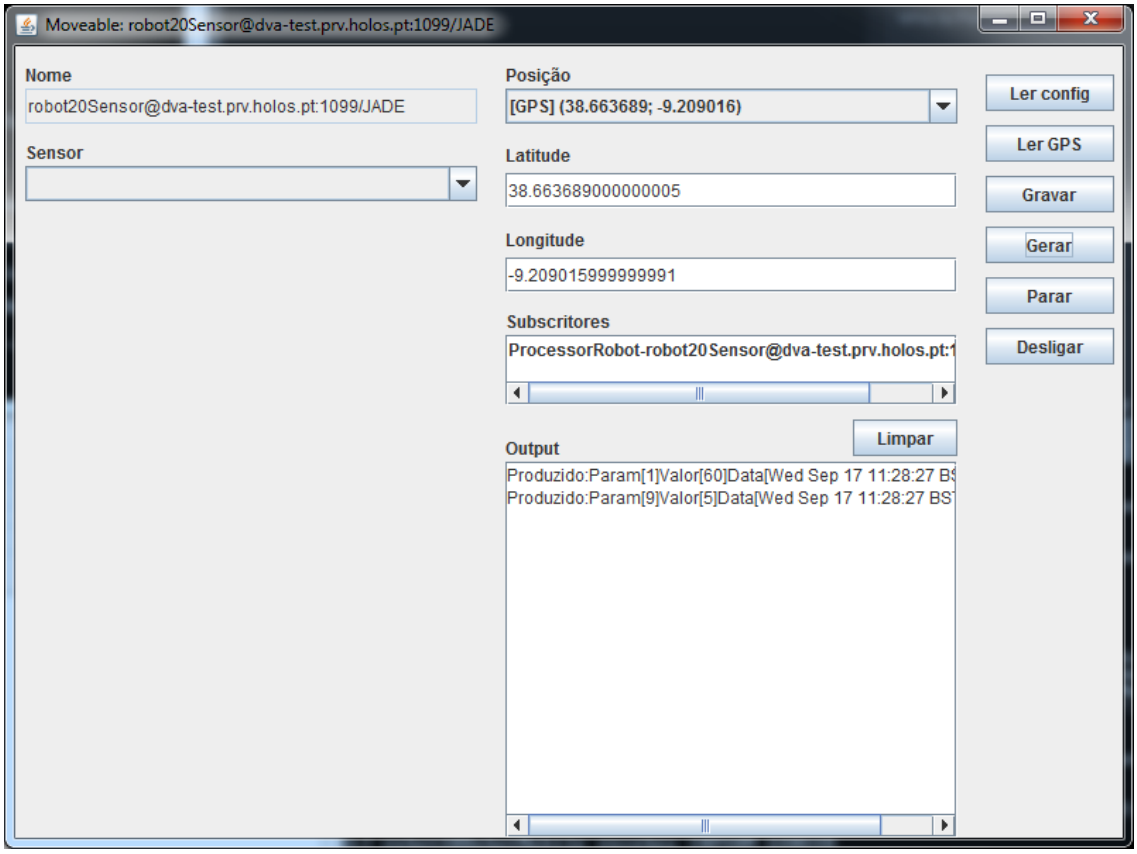

Figure 17 - RobotSensorAgent Interface at Simulation 1

In this simulation it was also tested the behavior of the system in case of a failure in the registration process. It is shown in Figure 18, the *RobotManagerAgent*'s response when the robot simulator tries to register in the system for the second time with the same ID. As can be seen, after validating the registration request to a device ID *simulacao20* with an *ACK* message, the *RobotManagerAgent* has not validated its second registration application and returned with an error message, due to the error.

It is shown in Figure 19 the *RobotSensorAgent*'s behavior when there is a failure in its setup, especially in the registration process showed in Figure 12. When this fault exists, the *RobotSensorAgent* is not aware of what sensors are available at the robot, then there is no point in having this robot registered in the system. So when this failure happens, this agent shuts down, and notifies the *RobotManagerAgent* to unregister the device and kill their *RobotMobileAgent*.

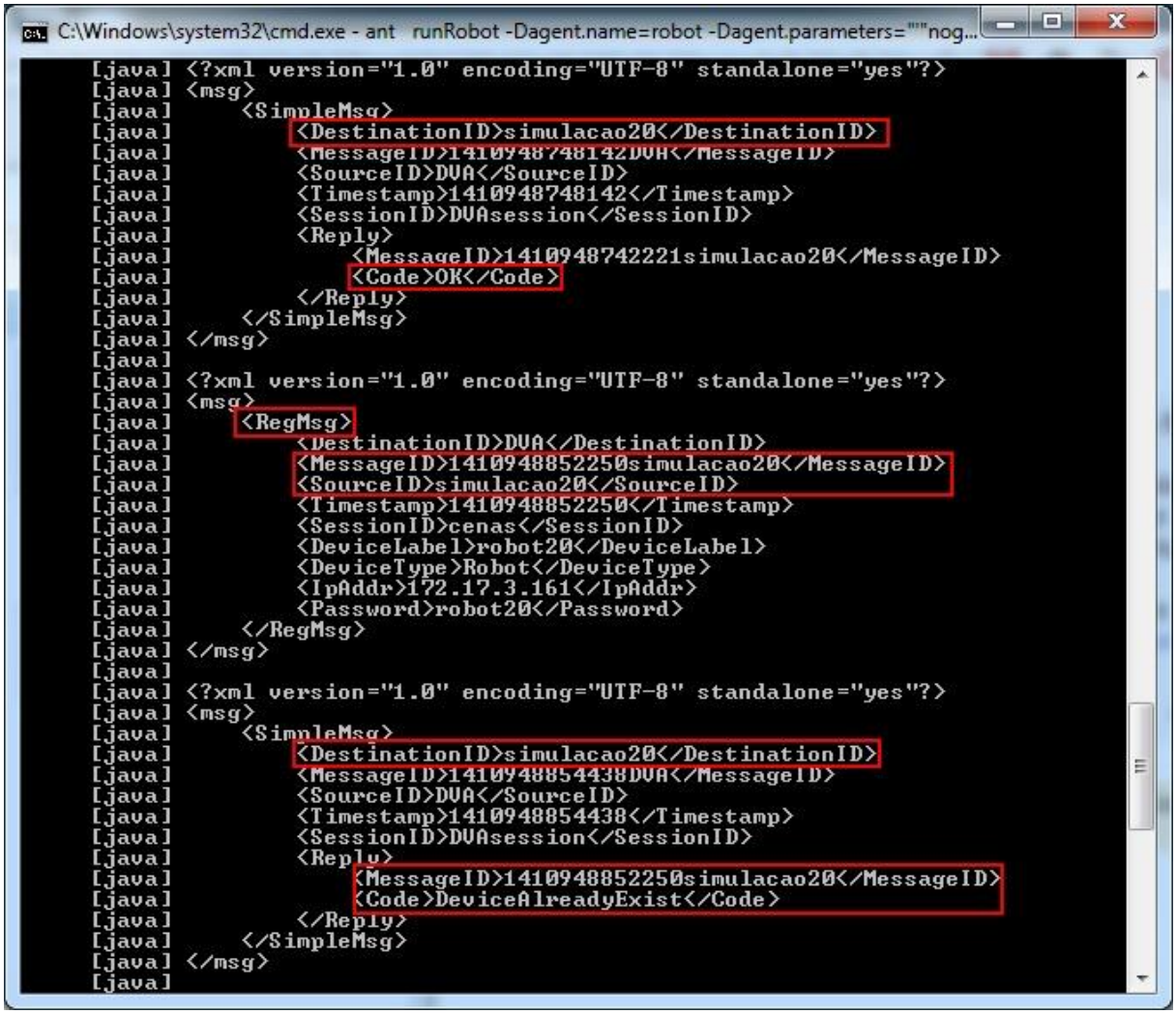

Figure 18 - Invalid registration simulation

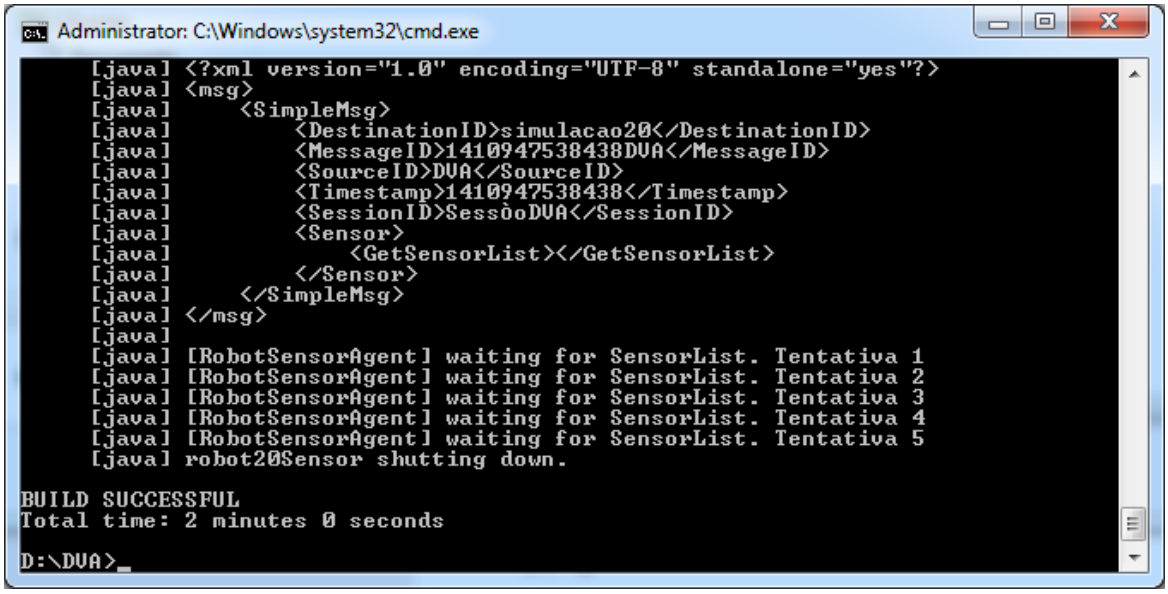

Figure 19 - RobotSensorAgent behaviour at an incomplete registration

5.2. Simulation 2

Once the Simulation 1 was performed, we can conclude that the *RobotSensorAgent* is properly handling messages from robot simulator. This second simulation was performed on a scenario distributed over a LAN network, with agents being executed remotely by the test laptop at the DVA multi-agent platform, as shown in Figure 20 with *robot* agent as *RobotManagerAgent*, *robot20Sensor* agent as *RobotSensorAgent* and *robot20RobotMobile* as *RobotMobileAgent* of *robot20*'s simulation application. At this test, the *robot20*'s simulator was simulating the robot operating with two temperature sensors, one measuring 60ºC and another measuring 45ºC.

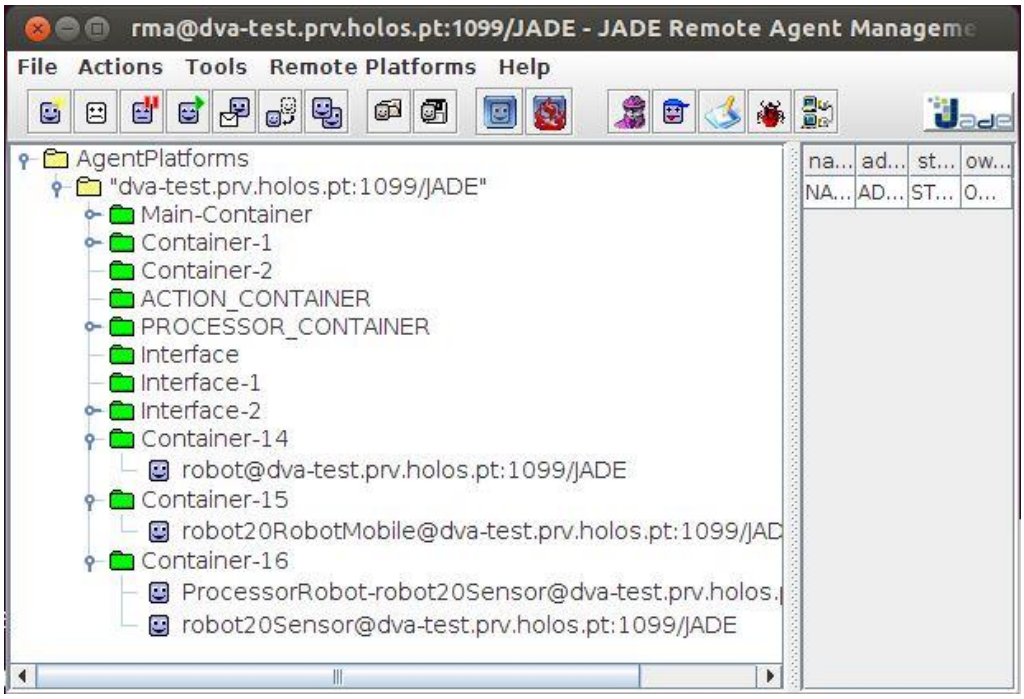

Figure 20 - JADE agent platform at dva-test computer

With agents running remotely on the DVA platform, it is possible to test whether the values of the sensors of the robot are fully integrated in the DVA system. During this simulation we observed the behavior of the DVA web application, which is the user interface of the record of all values read by the sensors of the DVA. The behavior of the web application of DVA was registered in Figure 21 and Figure 22.

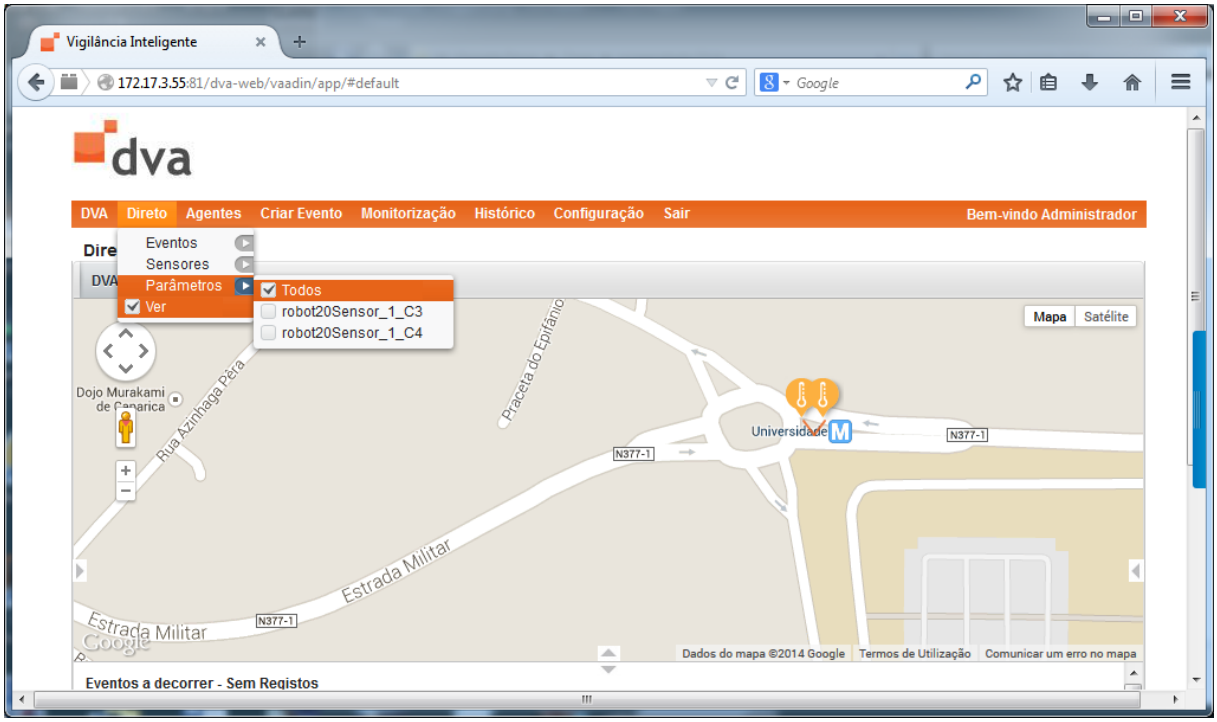

Figure 21 - DVA web application with robot sensor parameters displayed

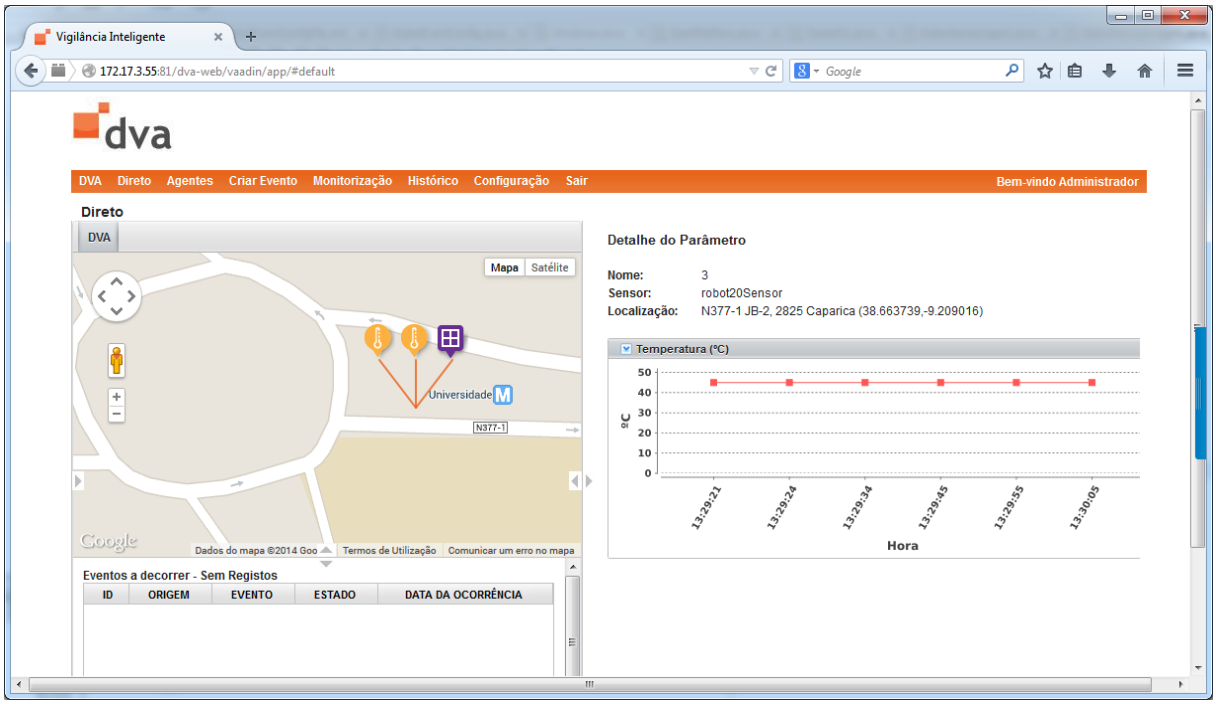

Figure 22 - DVA web application with parameters values displayed

At this simulation stage was also possible to observe the behavior of *RobotMobileAgent* when there is an event that can be handled by this agent. For this

simulation, the DVA has been configured to use *ROBOT* agents at Fire events. Thus, when a fire is detected event in the system, it creates a *ActionAgent* to treat this (usual DVA procedure) and this will send a *CFP* message to all agents of type *ROBOT*. Receiving this message, the *RobotMobileAgent* will check if it has the skills to handle this kind of events. To this end it uses the configuration that is in *RobotEventConfigurationFile* to know what kind of sensors need to be able to respond to events Fire-type. In the case of this simulation, the configuration file shown in Figure 6 was used.

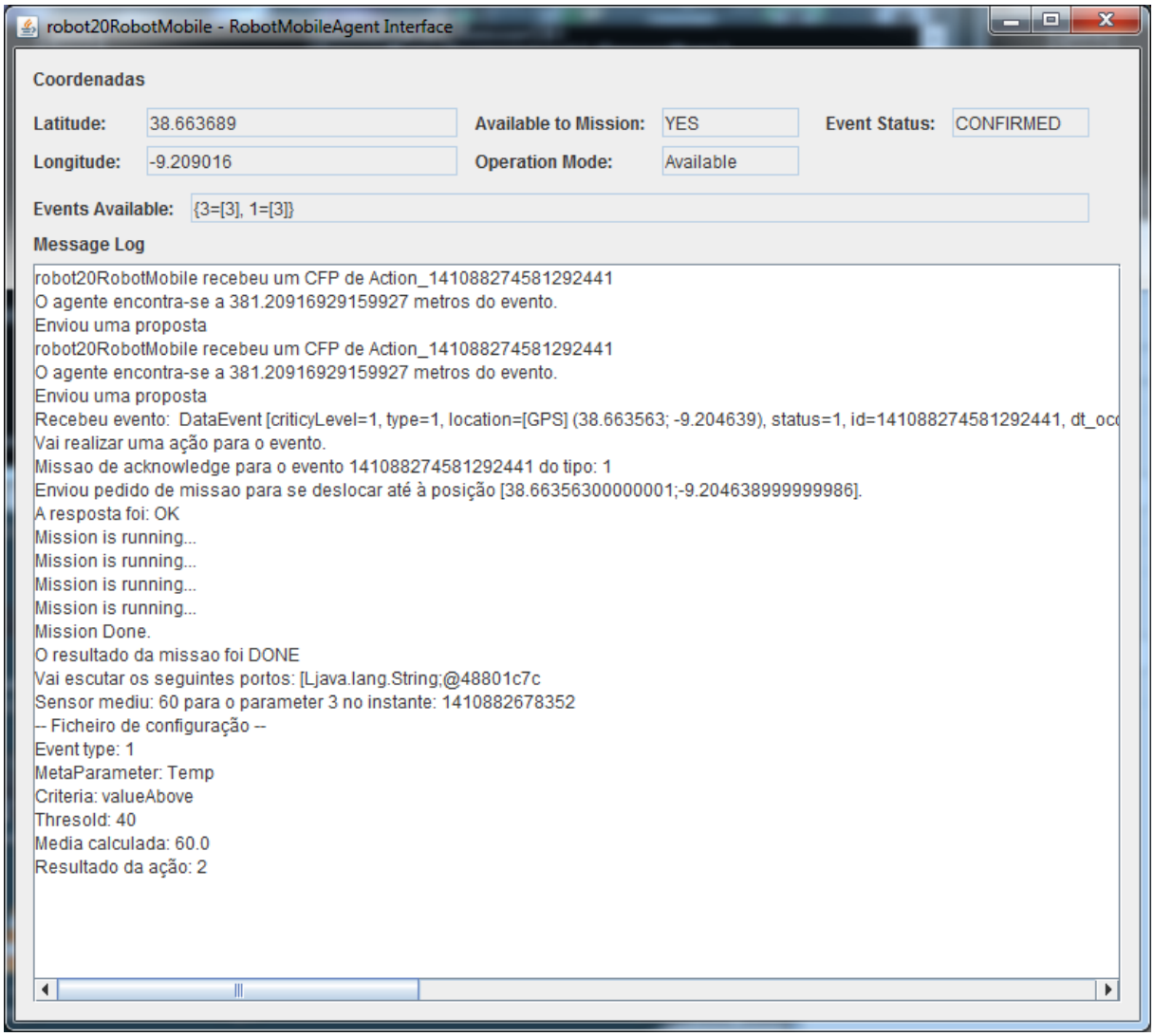

Figure 23 - RobotMobileAgent's interface when handled an event

The Figure 23 shows the behavior of *RobotMobileAgent* during this simulation. Analyzing the *MessageLogBox* of the agent interface, we can see that it responded to the *CFP* message of the *ActionAgent* with a proposal to respond to the event. This

proposal contains the distance of 381 meters from the agent to the event location. The *ActionAgent* responded positively to the proposal of this agent and then the it performed an acknowledge mission to handle this event. For such, it sent a *GoToGPSPosition* request to the robot move to event's location, and after receiving its acknowledge, began to monitor the status of the mission. When the mission was considered *DONE* by the robot, the *RobotMobileAgent* was to verify the value of the sensors to determine on the status of the event, taking into account the configuration loaded into the event configuration file. In Figure 24 it is possible to observe the event of Fire (13) was confirmed by the user "ServRobot Simulado", which corresponds to the simulator's account in the DVA system used in this simulation.

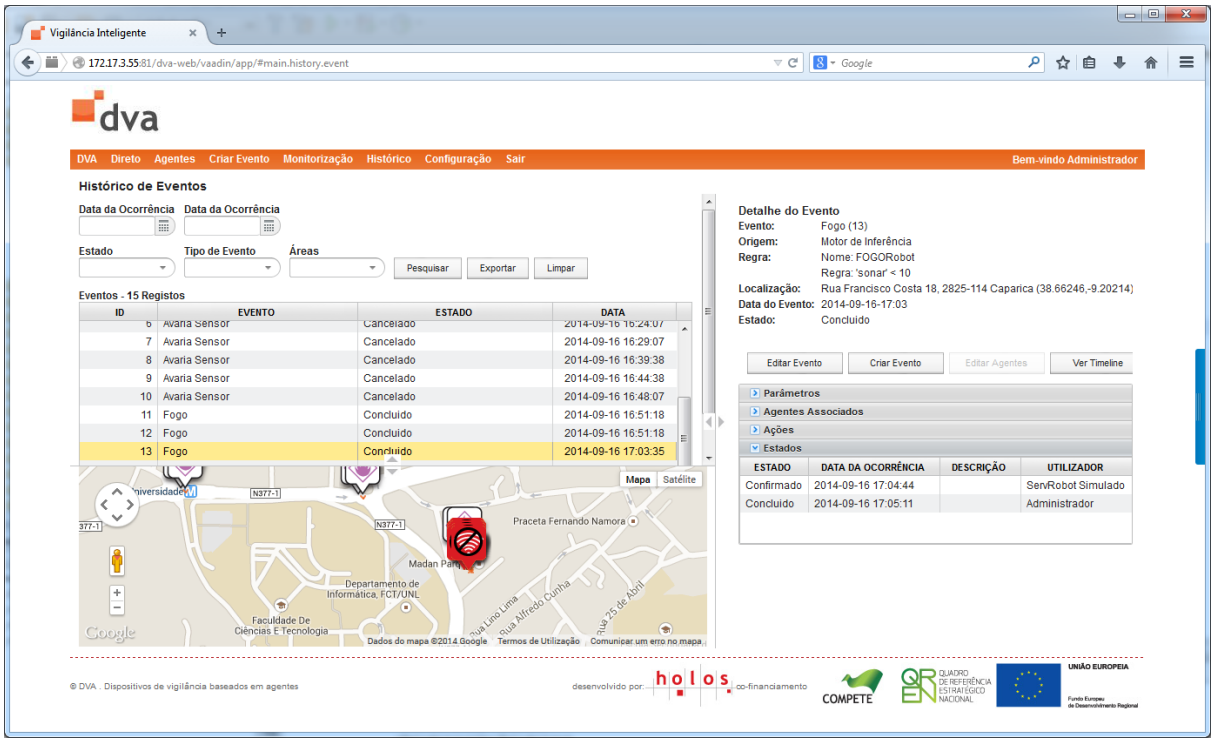

Figure 24 - DVA web application displaying event confirmed by the RobotMobileAgent

5.3. Experimental Test

In Simulation 2, the behavior of the system using a robot simulator was observed. In this experimental test, the simulator was replaced by ServRobot, which used the same *ZMQDriver* that simulator used to communicate with DVA agents. The remaining system elements of Simulation 2 remained.

Results were recorded in the following figures: Figure 25, Figure 26, Figure 27 and Figure 28.

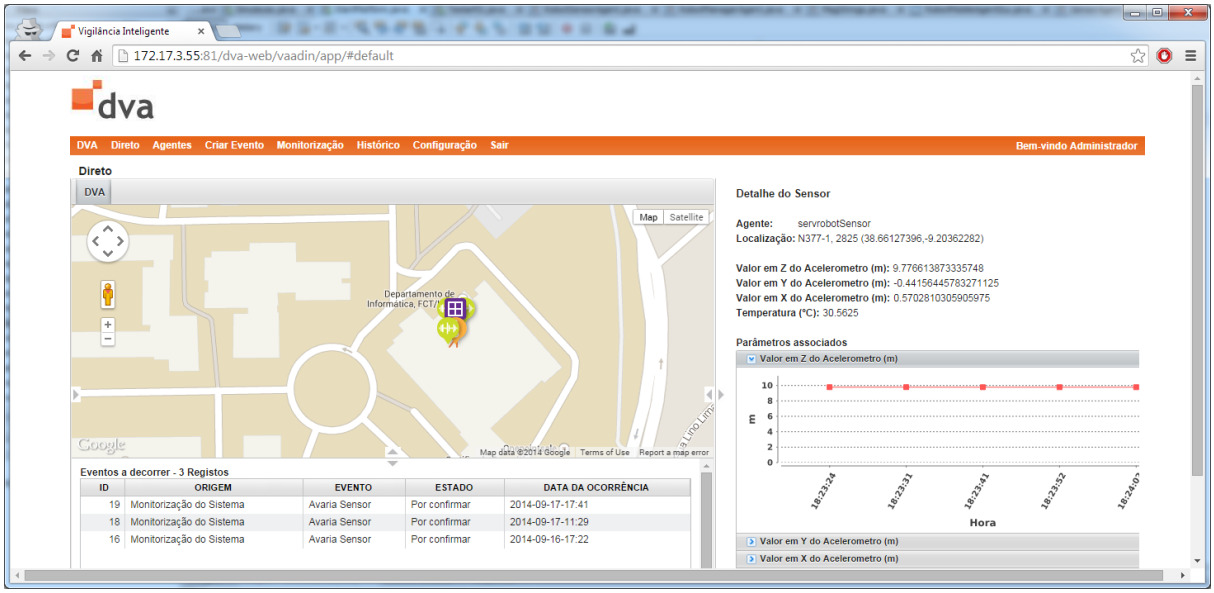

Figure 25 - DVA web application showing ServRobot's parameter values (Z-axis acceleration)

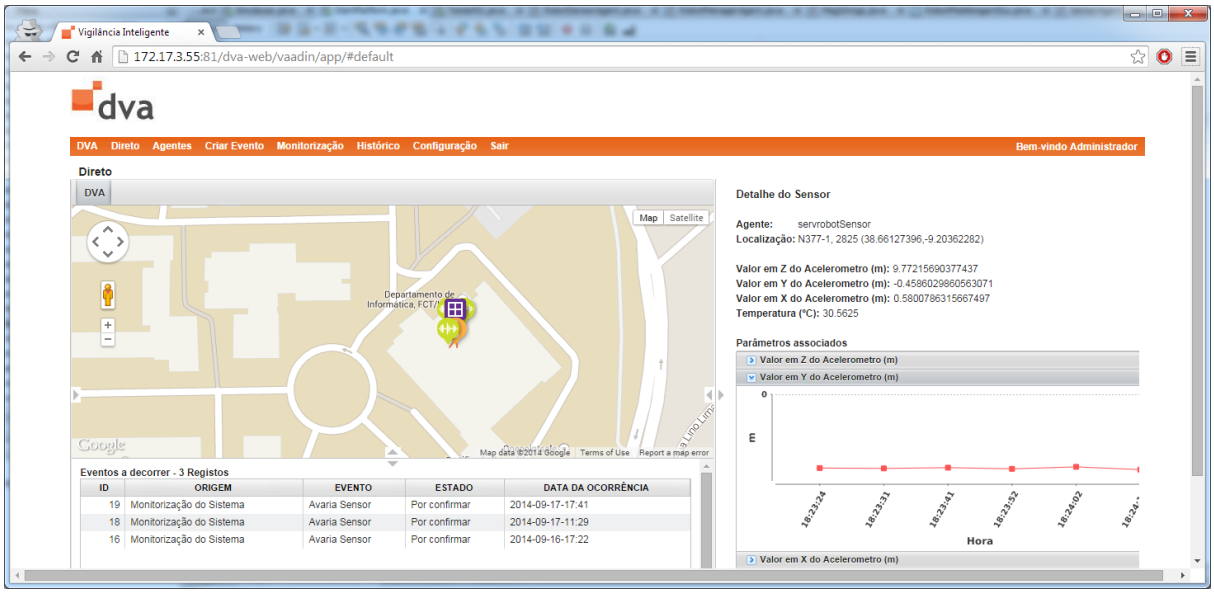

Figure 26 - DVA web application showing ServRobot's parameter values (Y-axis acceleration)

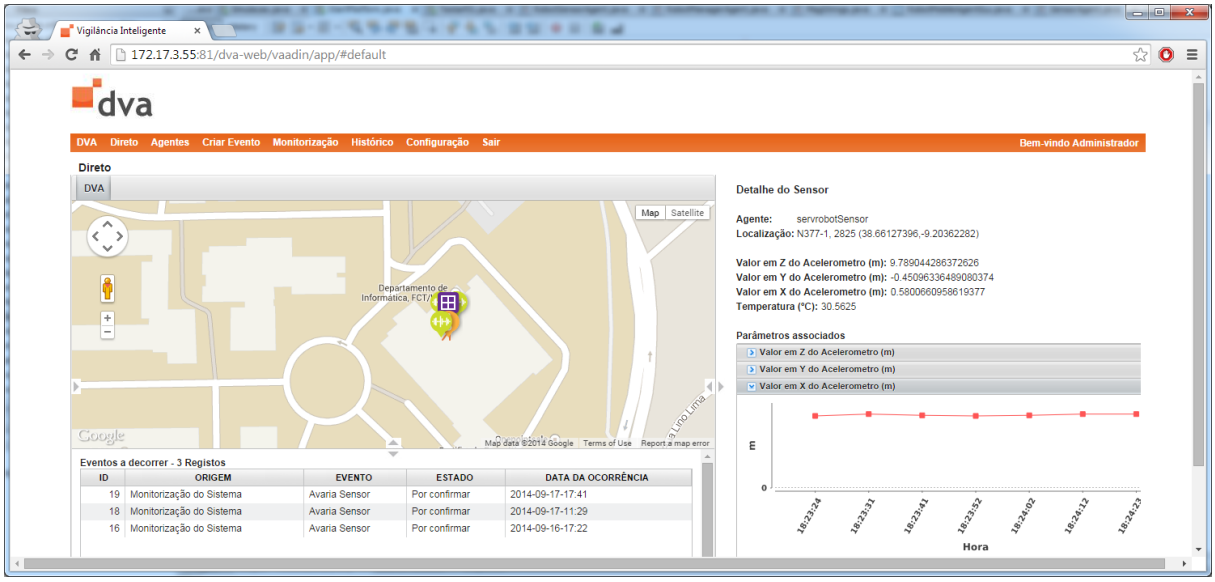

Figure 27 - DVA web application showing ServRobot's parameter values (X-axis acceleration)

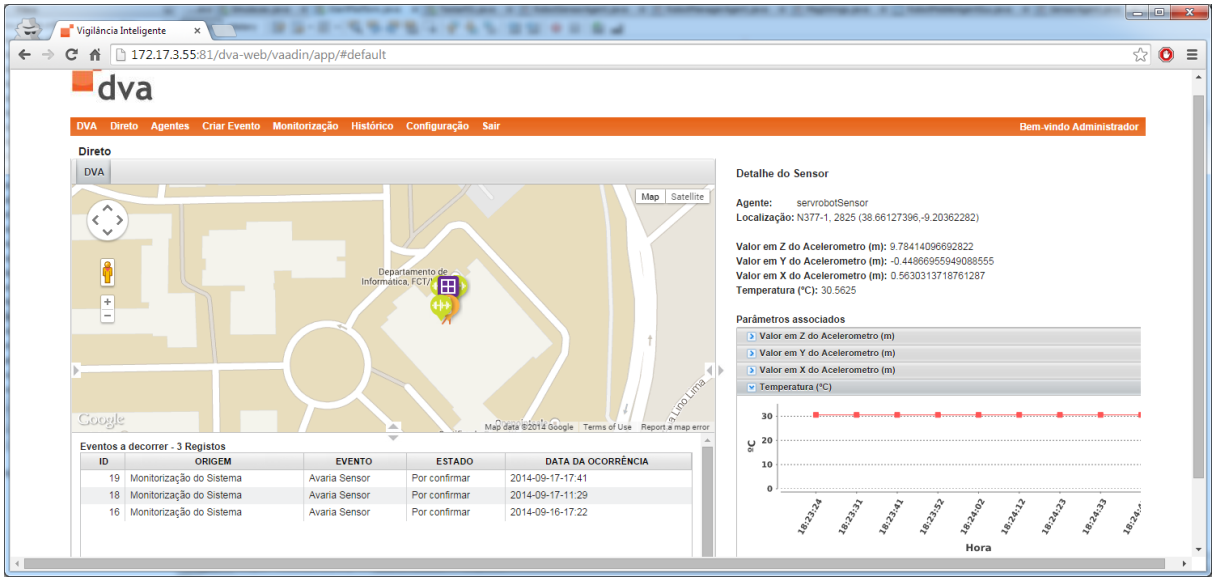

Figure 28 - DVA web application showing ServRobot's parameter values (temperature)

During this experimental test, the robot was stationary in a closed room. The values of the parameters of the accelerometer did not remain constant, because it is a very sensitive sensor (sensor specification), unlike the temperature sensor, which remained constant.

With the results shown in the figures above, it is possible to check that the robot was integrated into the DVA system successfully as a sensor with different parameters, in this case four parameters: X-axis acceleration, the Y-axis acceleration, Z-axis acceleration and temperature .

6

6. Conclusions and Future Work

This chapter serves to conclude this dissertation with a summary of conclusions from all the different phases of the development of this thesis and also with a summary of future work that could be developed following this, to complement this work.

6.1. Conclusions

Day after day, there is a greater concern to ensure the security of critical infrastructure and goods. But, despite the development of more sophisticated, secure and intelligent surveillance systems, human beings always have their crucial role in decision making and in particular actions. In other tasks such as surveillance, event confirmation, among others, the role of the human being may be replaced or supplemented by another element, thereby releasing the human element from a costly task, because of its monotony or difficulty, these tasks may be subject to failures. Another type of tasks where the human element can be replaced are tasks in environments with toxic gases, difficult access or very audible noise. This type of tasks, with greater difficulty in humans to perform successfully, are subject to harm their physical integrity.

To this end, the technological advances in the area of robotics can contribute to greater autonomy and efficiency of surveillance systems, complementing the role of the human element in security systems. With a mobile robot integrated at a surveillance system, it may be used as:

- a mobile sensor for detection of events;

- a mobile agent to confirm events;
- an agent for surveillance missions;
- in teleoperation.

The architecture proposed in this dissertation was implemented in the DVA system, allowing the integration of the ServRobot's mobile robot. Therefore it was concluded that integration with mobile robotic agents in operation, the system gains more autonomy in the detection and confirmation of events.

Before this integration, for events detection in a certain place, it was necessary to install a sensor for this purpose in that location. With this proposal, using a mobile robot equipped with sensors, we can remotely move the robot to the location you want to monitor, without the need to install new sensors. Thus there is a dynamic area of coverage reusing the same mobile robot, with different surveillance missions.

This integration also helped the DVA system autonomy with respect to the events confirmation. With this proposed architecture, the DVA system has the ability to assign an event not confirmed to a robotic agent, and this, using their ability to autonomous navigation may be directed to the location of the event and use its sensors to confirm or cancel the alarm. Thus, the DVA system becomes less dependent on a human mobile agent to confirm an event, as occurs in the traditional architecture. With a robotic agent integrated in the system, the DVA system can use its own means to confirm events, becoming more intelligent and autonomous.

However, this integration does not compromise the functioning of the DVA system if it does not have an available robot. In this case, the DVA system will work with a traditional architecture. The same applies to ServRobot, which will continue to have their functionalities standalone, without being integrated into the DVA system. This allows the mobile robot's ServRobot can either be used in integrated DVA system as standalone, allowing its reuse and application in different scenarios.

In summary, the integration of robotic agents can make surveillance systems more intelligent and autonomous, making them less dependent human agents, which may focus on other tasks which they are indispensable. The integration of an external robotic agent also enables its reuse in other applications, not limited to its operation in the surveillance system.

6.2. Future Work

The future work of this dissertation involves a continuation in the development of intelligence and autonomy of surveillance systems. For such learning systems may be developed, where the system can use its historical of events and actions performed in order to decide more quickly and effectively the action that will carry out to a particular event. A learning system would also contribute to other functionality: in case of a series or a pattern of events being detected in a given location, the surveillance system could automatically assign a surveillance mission of a mobile robot to the designated location, given that it is a location sensitive to events.

There may also be a future work within the framework of the integration of other types of robotic agents, in addition to terrestrial robots which was the main focus of the work of this dissertation. Aerial and aquatic robots can also be integrated into a surveillance system, and may be more effective than a terrestrial robot if the area to monitor is difficult to access by land.

The implementation of an inference engine for detection of environment and context with the robot is inserted, using sensory information captured by the robot, will also be helping in feature detection and confirmation of events, making the surveillance system more efficient these situations and sensing fewer false positives.

6.3. Scientific Contributions

As part of the development of this dissertation, the following scientific articles were published: [34,39].

7

7. References

[1] G. I. Zysman, J. A. Tarallo, R. E. Howard, J. Freidenfelds, R. A. Valenzuela, and P. M. Mankiewich, "♦ Technology Evolution for Mobile and Personal Communications," no. March, pp. 107–129, 2000.

[2] S. Avancha, J. Undercoffer, and A. Arora, "Wireless sensor networks," vol. 43, pp. 417–419, 2003.

[3] R. Khichar and S. S. Upadhyay, "Wireless sensor networks and their applications in geomatics : case study on developments in developing countries," pp. 43–48, 2010.

[4] R. Vidoni, F. García-Sánchez, A. Gasparetto, and R. Martínez-Béjar, "An intelligent framework to manage robotic autonomous agents," *Expert Systems with Applications*, vol. 38, no. 6, pp. 7430–7439, Jun. 2011.

[5] "History of Video Surveillance and CCTV." [Online]. Available: http://www.wecusurveillance.com/cctvhistory. [Accessed: 15-Sep-2014].

[6] K. Kim, S. Bae, and K. Huh, "Intelligent Surveillance and Security Robot Systems," pp. 0–3, 2010.

[7] M. Valera and S. A. Velastin, "Intelligent distributed surveillance systems : a review," no. 20041147, 2005.

[8] M. Leo, P. Spagnolo, T. D'Orazio, P. L. Mazzeo, and a. Distante, "Real-time smart surveillance using motion analysis," *Expert Systems*, vol. 27, no. 5, pp. 314–337, Nov. 2010.

[9] M. Tracking, X. Zhang, and S. Member, "Adaptive Control and Reconfiguration of Mobile Wireless Sensor Networks for Dynamic," vol. 56, no. 10, pp. 2429–2444, 2011.

[10] M. Leo, P. Spagnolo, T. Martiriggiano, A. Caroppo, and T. D. Orazio, "A System to Automatically Monitor Forbidden Areas," pp. 570–575, 2005.

[11] B. Balasubramanian and V. K. Garg, "Fault Tolerance in Distributed Systems Using Fused Data Structures," *IEEE Transactions on Parallel and Distributed Systems*, vol. 24, no. 4, pp. 701–715, Apr. 2013.

[12] T. Balch and R. C. Arkin, "Communication in reactive multiagent robotic systems," *Autonomous Robots*, vol. 1, no. 1, pp. 27–52, 1994.

[13] F. Mondada and L. M. Gambardella, "Swarm-Bot : A New Distributed Robotic Concept," pp. 193–221, 2004.

[14] A. Ghencea, V. Gaucan, and D. Pirvu, "Distributed Systems and Web Technologies," no. 5, pp. 1–14, 2011.

[15] L. F. Capretz, *Object-Oriented Software: Design*. World Scientific, 1996, p. 288.

[16] P. Taylor, A. Pokahr, and L. Braubach, "International Journal of Parallel , The active components approach for distributed systems development," no. March 2014, pp. 37–41, 2013.

[17] Y. Zhu, "Research on Web Service-Oriented Data Integration in the Distributed System," no. 2010.

[18] P. Leitão, "Agent-based distributed manufacturing control: A state-of-the-art survey," *Engineering Applications of Artificial Intelligence*, vol. 22, no. 7, pp. 979–991, Oct. 2009.

[19] R. J. Rabelo, L. M. Camarinha-matos, and H. Afsarmanesh, "Autonomous Systems Multi-agent-based agile scheduling," vol. 27, pp. 15–28, 1999.

[20] J. Chunfeng, "Research of Task Decomposition Based on Multi-Agent Cooperation," pp. 1942–1944, 2011.

[21] P. Taylor, "International Journal of Production Formal verification of negotiation protocols for multi-agent manufacturing systems," no. December, pp. 37–41, 2013.

[22] M. J. Shaw, "A distributed scheduling method for computer integrated manufacturing: the use of local area networks in cellular," *International Journal of Production Research. Sep87*, vol. 25, no. 9, pp. 1285–1303, 1987.

[23] F. Lin and D. H. Norrie, "Schema-based conversation modeling for agent-oriented manufacturing systems," vol. 46, pp. 259–274, 2001.

[24] P. Iñigo-Blasco, F. Diaz-del-Rio, M. C. Romero-Ternero, D. Cagigas-Muñiz, and S. Vicente-Diaz, "Robotics software frameworks for multi-agent robotic systems development," *Robotics and Autonomous Systems*, vol. 60, no. 6, pp. 803–821, Jun. 2012.

[25] F. For and I. Physical, "FIPA Contract Net Interaction Protocol Specification," 2002.

[26] F. For and I. Physical, "FOUNDATION FOR INTELLIGENT PHYSICAL AGENTS," 2002.

[27] R. Abielmona, E. Petriu, M. Harb, and S. Wesolkowski, "Mission-Driven Robotic Intelligent Sensor Agents for Territorial Security," *IEEE Computational Intelligence Magazine*, vol. 6, no. 1, pp. 55–67, 2011.

[28] Z. a. Baig, "Multi-agent systems for protecting critical infrastructures: A survey," *Journal of Network and Computer Applications*, vol. 35, no. 3, pp. 1151–1161, May 2012.

[29] M. Pipattanasomporn, H. Feroze, and S. Rahman, "Multi-agent systems in a distributed smart grid: Design and implementation," *2009 IEEE/PES Power Systems Conference and Exposition*, pp. 1–8, Mar. 2009.

[30] J. Silva, N. Lau, and A. J. R. Neves, "Localization techniques for autonomous mobile robots," *Eletectrónica e Telecomunicações*, vol. 5, no. 3, pp. 309–316, 2011.

[31] J. Kramer and M. Scheutz, "Development environments for autonomous mobile robots : A survey," pp. 101–132, 2007.

[32] F. B. Alberto and O. Gabriel, "Visual Navigation for Mobile Robots : A Survey," pp. 263–296, 2008.

[33] S. Coradeschi and A. Saffiotti, "The Future of AI Symbiotic Robotic Systems : Humans , Robots , and Smart Environments," pp. 20–22, 2006.

[34] B. Dias, B. Rodrigues, J. Claro, J. P. Pimentão, and P. Sousa, "Architecture and Message Protocol Proposal for Robot ' s Integration in Multi-Agent

Surveillance System," in *Rough Sets and Current Trends in Soft Computing*, 2014, pp. 366–373.

[35] W. Chao, "Multi-Agent Based Distributed Video Surveillance System over IP," 2008.

[36] T. Winkler and B. Rinner, "Security and Privacy Protection in Visual Sensor Networks : A Survey," vol. 47, no. 1, pp. 1–42, 2014.

[37] A. S. Vidya, "Video Surveillance System for Security Applications," vol. 74, no. 1, pp. 17–24, 2013.

[38] M. Bramberger, A. Doblander, A. Maier, and B. Rinner, "Smart Cameras for Surveillance Applications," 2006.

[39] J. Claro, B. Dias, B. Rodrigues, J. Paulo, and P. Sousa, "Autonomous robot integration in Surveillance System Architecture and communication protocol for systems cooperation," *16th International Power Electronics and Motion Control Conference and Exposition*, 2014.

[40] T. Bray and J. Paoli, "Extensible markup language (XML)," *W3C*, vol. 1, no. August, 2006.

[41] F. Wang, J. Li, and H. Homayounfar, "A space efficient XML DOM parser," *Data & Knowledge Engineering*, vol. 60, no. 1, pp. 185–207, Jan. 2007.

[42] M. Kostoulas and M. Matsa, "XML screamer: an integrated approach to high performance XML parsing, validation and deserialization," *World Wide Web Conference Committee*, pp. 93–102, 2006.

[43] N. Piël, "ZeroMQ an introduction," 2010. [Online]. Available: http://nichol.as/zeromq-an-introduction.

[44] E. Ort and B. Mehta, "Java Architecture for XML Binding (JAXB)," 2003. [Online]. Available: http://www.oracle.com/technetwork/articles/javase/index-140168.html. [Accessed: 23-Sep-2014].