PeMMAS: A Tool for Studying the Performance of Multiagent Systems Developed in JADE

Alejandro Carrasco, *Member, IEEE*, Mª Dolores Hernández, *Member, IEEE*, Mª del Carmen Romero-Ternero, *Member, IEEE*, Francisco Sivianes, *Member, IEEE*, David Oviedo, and José Ignacio Escudero

Abstract:**This paper describes the performance measurement for multiagent systems (PeMMAS) tool, a system designed to study and measure the performance of any multiagent system (MAS) de-veloped in JADE. The tool itself is another MAS which is deployed and coexists alongside the one being studied. This characteristic allows us to adapt PeMMAS to any scenario in which MAS de-ployment in JADE is used. PeMMAS extracts information from the target MAS regarding the use of system resources, the flight time for comprehensive messages according to agent type, as well as the processing time for actions. After processing this information, PeMMAS sends a report to the final user for subsequent analysis.**

Index Terms**—Distributed artificial intelligence, multiagent systems (MAS), performance systems, telecontrol.**

I. Introduction

Ourgroup has been working to improve telecontrol oper-ation and operator interfaces in control centers of power facilities for a long time. In fact, we have developed several solutions integrating multimedia information in the interfaces used by telecontrol operators [1], [2].

In our search for a distributed solution to collect data from different elements of the facilities, we realized that multiagent systems (MAS) could provide a suitable solution and we decided to adopt this paradigm.

This led us to design an MAS, called CARISMA (the Spanish acronym for Remote Automatic Control of Solar Facilities with multiagent technology) as part of a research project supported by the Spanish government. This project aimed to apply the multiagent paradigm to develop integrated systems to automat-ically supervise and control renewable energy facilities, in our case photovoltaic, by using distributed intelligence.

CARISMA provides telecontrol operators with an interface to support the automatic supervision of the correct operation of the facility being telecontrolled. Its multiagent architecture makes it suitable for environmental monitoring applications and, in particular, solar facility maintenance.

We therefore decided to use a platform for the development and execution of MAS in Java called JADE [3]. This platform is widely used in many applications because of its flexibility, its good documentation, and its ability to meet standards of communication among FIPA agents [4]. Its strong points include the following.

1) JADE provides a stable layer to develop distributed applications making lower layer complexity transparent for the programmer.
2) JADE facilitates issues such as agent coordination, security, communication, mobility, redundancy, and other basic services in a distributed architecture.
3) JADE is open code. Hence, many people contribute to develop and maintain it.
4) As JADE has been developed in Java, it benefits from the properties of this technology.
5) JADE is FIPA compliant; therefore, it can communicate with agents developed in other environments that also implement FIPA.
6) JADE API is intuitive, easy to learn, and simple to use, thereby reducing development time.
7) JADE allows easy integration to other libraries to implement logic reasoning, and to PROTÉGÉ to develop ontologies.

Once CARISMA had been developed, implemented, and tested, we had to measure its performance to find out whether it was working correctly. The reason for this was that under this scenario, many events occur which lead agents to interact among themselves. They may send each other messages, perform actions autonomously, or have to propagate rules for their inference engine, among other possibilities. This requirement led us to create performance measurement for multiagent systems (PeMMAS). Thus, this tool is closely linked to the performance study of the MASs developed in JADE and, taking advantage of the characteristics of JADE, we have also implemented PeMMAS with JADE.

In short, we wanted to create a mechanism (a tool) to study the performance of a MAS developed in JADE in run time. Therefore, PeMMAS offers an interface that can be used by users who work monitoring and supervising a certain environment with connected elements. In our case, the telecontrol operator is the final user. In general, this tool provides an interface to setup, start, and stop a session test to analyze a MAS developed in JADE.

Fig. 1 shows the general layout for the use of PeMMAS with our CARISMA system and Figs. 6 and 10 show the PeMMAS user interface to report analysis results.

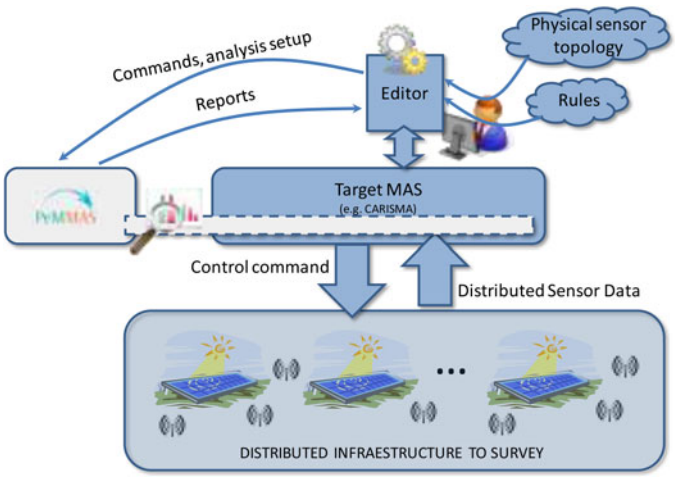

Fig. 1. General layout for using PeMMAS to analyze a MAS (CARISMA in this case).

Section II gives an overview of the state of MASs, JADE, and the different existing performance evaluation systems. Section III then provides a thorough examination of what a testing system is, how it works, and why one is needed. After explaining how the system works, Section IV describes the tests that were performed, how they behaved within the system, and the results obtained. Finally, the paper provides the conclusions about the work carried out.

II. State of the Art

Nowadays, MAS technology is used in a multitude of fields, from video games and films to defense systems, among many others. Its application is ideal for environments where a large amount of information has to be gathered from different points in a system and used to make calculations. Each agent consists of an independent entity capable of acting on its own. It has a limited vision of its surroundings, never a complete vision of the global system. Agents can be provided with specific knowledge about their environment (ontology), or they can share an available collection of comprehensive knowledge [5].

The scientific literature includes many applications based on the multiagent paradigm, from those that solve complex problems, such as recognition of suspicious human behavior [6], to the application of existing technology for MAS improvement, such as the use of diffused logic to make agents capable of reaching objectives in an optimal manner [7]. Some applications include the control of diverse environments, as seen for example in Abras *et al.* [8] which demonstrates a MAS responsible to control energy use in a home, while trying to use the optimal number of devices at any moment and informing with alarms if problems arise. In the field of power utilities, our research group proposed a MAS in [9] and [10] to supervise electrical facilities using a SCADA (supervisory control and data acquisition). In [11], an MAS was proposed capable of adjusting production to electricity demand and rationalizing the appropriate use of the installation based on market theories. In this field of MAS for markets, the year 1999 saw the introduction [12] of an MAS model capable of predicting changes in the stock markets, with a certain margin of error.

There is an official web site [3] for JADE, the environment of development and execution based on Java, where the number of companies and universities currently using this software for different purposes is shown. In [13], we can see an example of JADE being used to substitute most human decision making in an e-commerce system. In the field of the Internet, in [14], JADE is integrated along with a platform that allows for the installation of web services in servers during execution by using agents. Furthermore, in [15], a modification of this software is presented that includes a belief-desire-intention architecture for the agents.

We did not find much literature about tools developed to study MAS performance from the point of view of processing times, communication parameters, and system limitations. For example, Such *et al.* [16] presented a system to allow for the evaluation of MAS performance, but it was only based on message flight time. There have also been studies of MAS performance such as the one in [17], which presented an application of MASs in sensor networks. It included a study of MAS performance but only in terms of network congestion. The authors in [18] proposed network optimization using MAS, but they based this on communication and did not provide a complete study of performance. In [19], Serrano *et al.* proposed some techniques to analyze software performance in terms of agent behaviors in order to extract association rules.

In this paper, we propose a MAS developed in JADE to measure the performance of different parameters in any MAS that has also been developed in JADE. This is not a study of JADE efficiency as seen in [20], rather a way of checking whether the operation of a MAS developed JADE serves its purpose.

III. Tools Used to Study Performance

The PeMMAS system is a tool with a complex development that has been designed to help detect problems related to the functioning of an MAS. The paper discusses later how the idea to develop this system evolved, what it consists of, what measurements we can study using the PeMMAS, and finally provides a detailed explanation of how it works.

A. Antecedents and Motivation

In the paradigm of traditional programming, it is common practice to use different to measure software performance. These are known as "profiling" and the majority of programs such as Eclipse® and NetBeans® already incorporate their own tools to achieve this objective. They work by using time keepers to measure the length of the execution process and observe how CPU load and RAM memory vary. With this information, it is relatively simple to study parts of the code that execute more slowly. Nevertheless, one has to remember that with this type of programming, the code is always executed within the machine from which it is launched.

In the multiagent paradigm, agents are capable of moving from one place to another. This process is called "migration,"

and it is done through the network. This means that an agent begins execution within one machine, then migrates to another, and continues with the execution in a new destination. The agents' ability to migrate from one place to another is one of the main advantages of MASs, because it provides them with great flexibility to adapt to almost any scenario. However, this capacity can lead to problems when measuring process times.

Time keepers simply keep track of seconds. They can count seconds from the moment a machine is turned ON or OFF from a specified date of reference. For example, the date of reference of the UNIX system as well as that of many programming languages such as Java is 1 January, 1970. *A priori*, we might think that if all machines use the same date of reference, it should not matter whether an agent registers a time keeper, then migrates and registers with another machine to verify how long the process has taken. Nevertheless, slight to catastrophic errors can occur depending on the synchronization of the clocks in the machines.

All these problems are controlled by profiling tools that are included in the programming areas; however, the most widely used ones are not prepared for MASs. Although some of these tools are prepared, such as the INGENIAS platform [21], they are incapable of a performance study if the system has migrating agents.

Bearing in mind all these drawbacks, when an MAS is designed, one of the most complicated aspects is knowing whether it is working correctly. Although certain manual tests can be carried out to track flight times of messages among agents as well as how long certain tasks take, these processes are tedious and slow. In addition, an MAS is generally used when knowledge has to be distributed and the process divided into many parts, making manual measurements in this environment virtually impossible. This is why we have created this measuring tool to facilitate the task of checking whether a multiagent developed in JADE is functioning completely, reliably, and properly.

B. System Architecture

Despite PeMMAS development having been complex overall, its architecture has ultimately been designed in a very simple way. It only has two kinds of agents: tester agent (ATester), which is the main agent, and agent test board (ATP), which has as many instances as are required.

This simplicity provides the tool with two desirable characteristics from the point of view of any software development: flexibility and scalability.

Our objective was to create a tool capable of studying the performance of a MAS developed in JADE. Our proposal meant creating a new MAS deployed under the same implementation as the one being studied. Thus, PeMMAS architecture depends solely and exclusively on the architecture of the target MAS.

This is the optimal option since MAS are capable of adapting or even reconfiguring themselves if changes are produced in the system [22]. Thus, we have obtained a flexible tool that is scalable and can be adapted to any MAS that has been developed in JADE, as well as any testing environment we desire.

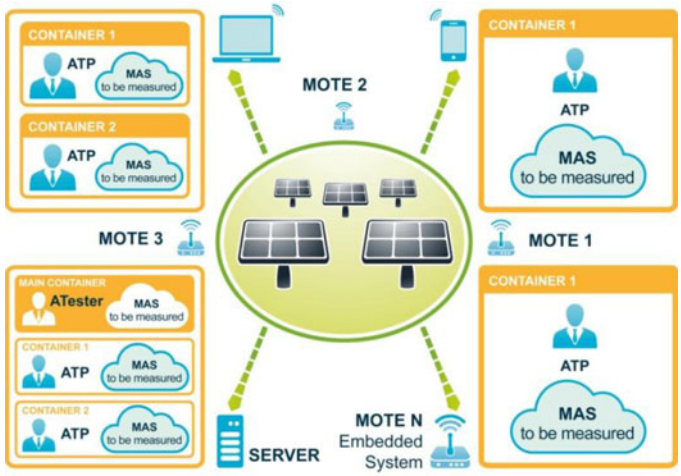

Fig. 2. Example of MAS architecture (CARISMA in this case) with integrated PeMMAS.

To achieve such a flexible system, everything has to be as generic as possible, or proper functioning will suffer. For this reason, we have employed an element found in all MASs developed in JADE: the container.

JADE containers are simply independent virtual areas where agents reside. Our proposal is to define a container for each physical machine found in the target system (the one being analyzed). This way, we can visually control how our agents are deployed. All JADE MASs have at least one "Main-Container." In this main container, there are a few internal JADE agents necessary for it to work properly. This is where the main agent of our measuring tool (ATester) will be incorporated.

As mentioned earlier, additional containers will be distributed in the area, along with the main container, either in the same machine or in other physical machines. They will have different names and agents belonging to the target system in their interiors. In each of these secondary containers, we are going to incorporate another type of agent, ATP, as shown in Fig. 2.

This figure shows CARISMA architecture, which is the target system in our case. It shows a set of solar panels from which a set of values (temperature, humidity, solar irradiation, etc.) is collected. These values are gathered by basic microcontroller systems (motes) and sent through Zigbee and other wireless protocols to the different CARISMA agents located in different kinds of hardware devices, such as embedded systems, mobile terminals, laptops, servers, etc.

From the point of view of the PeMMAS tool, the interesting point is the total number of containers that exists in the target-system infrastructure. This is because an ATester agent is created in the main container and an ATP is created for each secondary container existing in the target MAS.

To summarize, our MAS has two kinds of agents.

1) ATP (Agent Test Board): This type of agent will reside in secondary containers. There will be as many ATPs deployed as there are containers (other than main containers) in the system. We have named this "-board," since it will typically be present in each of the physical hardware devices. This agent is responsible to activate the testing mode

(which is explained later in this article) to begin collecting data about the MAS we are going to measure. When the main PeMMAS agent sends a finalization message to an ATP, this will then send all of the information collected for subsequent processing.

2) ATester (Tester Agent): This agent resides only in the main container and it is in contact with the user's interface. When ATester receives the order to start measuring system performance from the user, ATester sends a migration order to all the ATPs (one for each container in the target MAS) and waits for the stop order. When that order arrives, ATester requests data from all the ATPs and processes it. A summary of this data is displayed on a screen for the user and a report file with the detailed results is generated.

As we can see in Fig. 2, it does not matter how the MAS to be measured is deployed. It is represented as a cloud because, for these purposes, its characteristics and number of agents are irrelevant. We are simply going to place an agent in each container. It does not even matter if there is more than one container in each physical device since we will be guided by the number of containers that exist in the system, whatever that is.

C. Measurements Used

PeMMAS has an internally defined list of parameters to be determined. These measurements are divided into six types in an attempt to cover all the target information of the study which is typical in any MAS, regardless of what it was designed for. The following is a list of all parameter types which are described later with each measurement:

1) system functioning;
2) system limits;
3) system reliability;
4) communication;
5) processing times;
6) system dynamics.

1) System Functioning: The objective of these measurements is to check for the correct functioning of the different systems, and to process and analyze if there are errors in different parts of the code.

1) [FUNC_BEHAVIOUR]: This function controls execution errors in the programmed actions. If they exist, it stores the action, the agent that had this behavior, and the error that was produced. Its value is the number of errors that have occurred.
2) [FUNC_STATES]: This function measures the time agents spend in different states that are possible in a MAS: start up, blocked, active, migrating, and finalizing. Its value is the percentage of time that the agent stays in different states.
3) [FUNC_INTERACTIONS]: This function monitors the interaction between two agents with a focus on the type of messages exchanged, and whether these messages are expected or not. It tries to list the actions of each agent that produce interaction with other agents. Its value is the number of interactions between two agents.

2) System Limits: The following provides an idea of the MAS limits to be measured. The maximum number and kind of agents allowed affect the performance of the physical systems.

1) [LIM_NUMBER_AGENTS]: This measurement stores the number of agents found in the system upon stopping the test session, both throughout the system and in each container. Its value is the total number of agents in the whole system just before stopping the test session.
2) [LIM_CAPACITY_PROC]: This measurement stores the percentage of CPU used upon stopping the test session, both throughout the system and in each physical machine. Its value is the percentage of CPU used.
3) [LIM_CAPACITY_STORAGE]: This stores both the RAM memory used and available upon stopping the test session, both throughout the system and in each physical hardware device. Its value is the percentage of free memory in the target system.

3) System Reliability: The following measurements to be looked at aim to determine the reliability of the entire response offered by the system in various kinds of possible scenarios. In all cases, the measurement unit is the millisecond.

1) [FIAB_TMF]: This is the average time between system failures. Failures that entail some type of system block will be treated independently from those which provide incorrect answers.
2) [FIAB_DISP]: Average time between failures that entail a system crash and, as a result, generate a restart to be ready again. These can be distinguished by:
   a) [FIAB_DISP_MAS]: Availability of MAS/ platform.
   b) [FIAB_DISP_AGENT]: Availability of agent which can also be distinguished by agent type.

4) Communication: In an MAS, it is crucial for communication among agents to be as efficient as possible. It is therefore just as important to study thoroughly the precision of the flight times of all messages produced in the system. The following is a list of all measurements to be gathered. The value of all these communication parameters is given in milliseconds, barring the last one.

1) [COM_REQUEST]: Flight time (minimum, average, and maximum), of a REQUEST message, that is to say the time it takes from leaving one agent to arriving at another.
2) [COM_RESPONSE]: Flight time (minimum, average, and maximum) of a RESPONSE message.
3) [COM_AGENTS]: Flight time (minimum, average, and maximum) of messages between any two agents.
4) [COM_MAS]: If the multiagent to be measured possesses an organizational hierarchy system, this parameter measures the flight time (minimum, average, and maximum) of the messages sent from the agents that are higher in the hierarchy to those that are lower.
5) [COM_MEDIA_LAYERS]: Average flight time among agents from different levels of the hierarchy. If the MAS to be measured has a strict communication hierarchy in which the agents cannot communicate with others at the same time, this parameter will coincide with the average time recorded in [COM_AGENTS].

TABLE I
PEMMAS METHODS

PeMMAS Methods	
System Functioning	**System Limits**
[FUNC_BEHAVIOUR]: Errors in behaviour execution. [FUNC_STATES]: State of the agents. [FUNC_INTERACTIONS]: Control of interactions among agents.	[LIM_NUMBER_AGENTS]: Total number of agents [LIM_CAPACITY_PROC]: CPU use. [LIM_CAPACITY_STORAGE]: Available RAM memory.
Communication (flight time)	**Process times**
[COM_REQUEST]: Request messages. [COM_RESPONSE]: Response messages. [COM_AGENTS]: Between two agents regardless of type. [COM_MAS]: Between the superior and inferior agents of the hierarchy. [COM_MEDIA_LAYERS]: Between two agents of different types. [COM_MEDIA_ZONES]: Among agents from different containers. [COM_MEDIA_MOVE_AGENT]: Time needed to communicate migration. [COM_MEDIA_MSG_AGENT]: Number of messages exchanged.	[PROC_START_AGENT]: In agent start up. [PROC_END_AGENT]: In agent finalization. [PROC_RESPONSE_MSG]: To generate an answer to a request. [PROC_REQUEST_MSG]: To generate a request. [PROC_MEDIA_TASK_xx]: To configure the Parameter. [PROC_READ_SENSOR]: To read a sensor. [PROC_EXECUTE_ACTUATOR]: To execute a physical action.
System Reliability	**System Dynamics**
[FIAB_TMF]: Average time between system errors. [FIAB_DISP]: Average recovery time after a crash.	[SYS_INI_PLATFORM]: Average time needed to initialize a platform. [SYS_POLL_AGENT]: Average time needed to inquire about agent availability. [SYS_REC_COM_PLATFORM]: Average recovery time needed for a platform with communication failure. [SYS_REC_PLATFORM]: Average recovery time needed for a platform with hardware failure.

Summary of the methods used by PeMMAS classified by type.

6) [COM_MEDIA_ZONES]: Average communication time among agents in different zones (or containers) of the system.
7) [COM_MEDIA_MOVE_AGENT]: Average time to move an agent from one container to another in the system.
8) [COM_MEDIA_MSG_AGENT]: Average number of messages exchanged among the same kinds of agents. If the MAS to be measured has a strict communication hierarchy, this parameter will equal 0, since messages of this type cannot be sent. Its value is that average.

5) Processing Times: The following is a list of measurements designated to find the times required by an agent to carry out a given task. The value of all these parameters is given in milliseconds.

1) [PROC_START_AGENT]: Processing time necessary for an agent to be in active mode, that is to say, the start-up time.
2) [PROC_END_AGENT]: Processing time necessary to stop the activity of an agent, that is to say, the time it takes from the moment the elimination of an agent is ordered until this action is actually carried out.
3) [PROC_RESPONSE_MSG]: Processing time necessary to generate a RESPONSE type answer after receiving a REQUEST message.
4) [PROC_REQUEST_MSG]: Processing time necessary to create a request for another agent.
5) [PROC_MEDIA_TASK_xx]: Average process time necessary for an agent to perform certain tasks (indicated by "xx"). This type of parameter is reserved for when we wish to measure times of a specific MAS task we want to evaluate.
6) [PROC_READ_SENSOR]: If there is an agent responsible to read the information at given intervals in a MAS, this parameter records average times that the mentioned agent takes to do the reading.
7) [PROC_EXECUTE_ACTUATOR]: If an MAS that is going to be evaluated has an agent deployed that is solely responsible to carry out actions, this involves physical changes in the environment.

6) System Dynamics: The following is a list of measurements designated to find the times required for specific events that could occur in the system. The value of all these parameters is given in milliseconds.

1) [SYS_INI_PLATFORM]: The average time needed to initialize a platform, from the feed or from the point of restart, until it is ready.
2) [SYS_POLL_AGENT]: Time needed for an interview of agent availability. Average time needed to know which agents are available to test or control.
3) [SYS_REC_COM_PLATFORM]: Average recovery time of a platform that has crashed due to failure in the communication system.
4) [SYS_REC_PLATFORM]: Average recovery time of a platform that has crashed due to another kind of failure (hardware or software failure, a problem with the feed, etc.).

Table I shows a summary of the methods used by PeMMAS classified by type.

D. How Does PeMMAS Work?

Although the PeMMAS system has been prepared to work with any MAS developed in JADE, certain elements must be included in the MAS that is going to be measured.

First, the comprehensive Boolean variables must be defined:

1) Tester mode: This variable allows the agents to store text files with the measurements gathered in the physical machine where they are found. They can be activated without executing the PeMMAS, the only difference is that the information is always gathered although it will not be sent to the main agent for processing.
2) Testing: This variable is activated when the PeMMAS begins the testing session and registers a false value when it concludes.

Once the comprehensive variables have been defined, a library with the MAS aspect to be measured should be included. This library contains a series of utilities that facilitate the task of measuring the desired parameters for the user. There are basically two kinds of parameters to be measured: those that require a time and those that only need text. Therefore, the library contains the methods required for timer use and text writing.

Now that we have obtained the tools to gather information, we have to focus on the desired points of the code with the necessary lines to store the parameters with the appropriate format. For example, when measuring the flight time of a message from one agent to another, various aspects may be of interest, such as the origin of the agent, the agent's destination, the type of message (REQUEST, AGREE, etc.), the time it is sent, and the behavior requested by the message. All of this information has to be linked together according to the determined format so that the ATester will be able to process it all later and show the correct results.

Finally, before carrying out the first test session, the most precise synchronization possible of all platforms is fundamental. We used a network time protocol (NTP) server located in Spain due to its geographic proximity. Thanks to this server, we were able to calculate the differences between its clock and the clock in our machine. Nevertheless, Java does not allow the internal time of all its operating systems to be manipulated in a normal way. To remedy this, we created a function that sets the time with the applied difference. We can employ this function whenever we need to note the time.

At this point, we can create the ATester agent in the *"Main-Container"* using the JADE libraries. When the ATester is created, it contacts the AMS[1] agent of JADE to find out how many containers there are in the MAS we want to measure. When this number has been obtained, the *"Main-Container"* creates as many ATPs as there are containers. After creating all of the ATPs, each one is sent in the order of migration to the corresponding container. After they have all migrated, as shown in Fig. 3, the ATester becomes inactive as it waits for the finalization orders.

When the ATP arrives at its destination container, the "Tester mode" and "Testing" Boolean variables activate. This simple

[1]Agent Management System: an internal JADE agent that takes control of the agents working in the system.

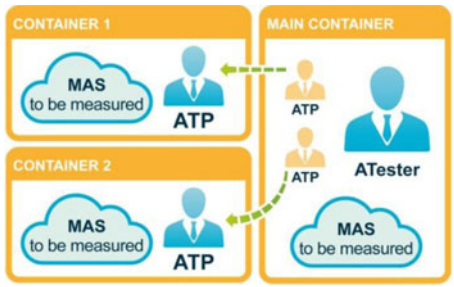

Fig. 3. ATP migrates from the main-container.

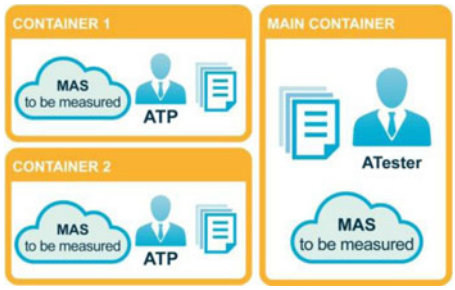

Fig. 4. One file for each measured parameter is saved in each container.

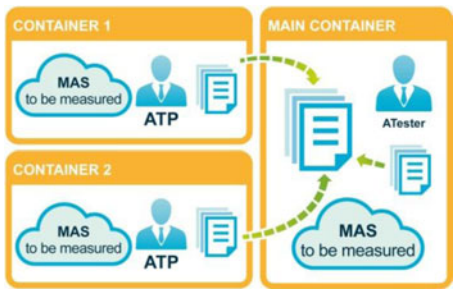

Fig. 5. ATester combines all received lines.

operation allows all the MAS agents we are measuring to begin storing information due to the lines of code that have been previously integrated along with the codes of the MAS that are to be measured. For each defined test parameter, a file consisting of different text will be created, as seen in Fig. 4. After this point, the ATP waits for the finalization order, just like the ATester agent.

Once the established testing time is over, the interface stops the test session, and a finalization message is sent to the ATester. Upon receiving the message, the ATester sends another message to the ATPs. This message indicates that the ATPs have to begin sending all the information from the different physical machines to the ATester. At this point, each ATP scans the folder in which all text files with the stored data are located. It then reads each text file line by line and at 25 line intervals, it sends a message to the ATester. The ATester is in charge of collecting these messages and combining all the received lines to create a single text file for each testing parameter that has been read, as seen in Fig. 5. Thus, if we want to read ten test parameters, we will have ten text files in each physical machine, or one per parameter. When all the lines have been sent to the ATester, it groups them

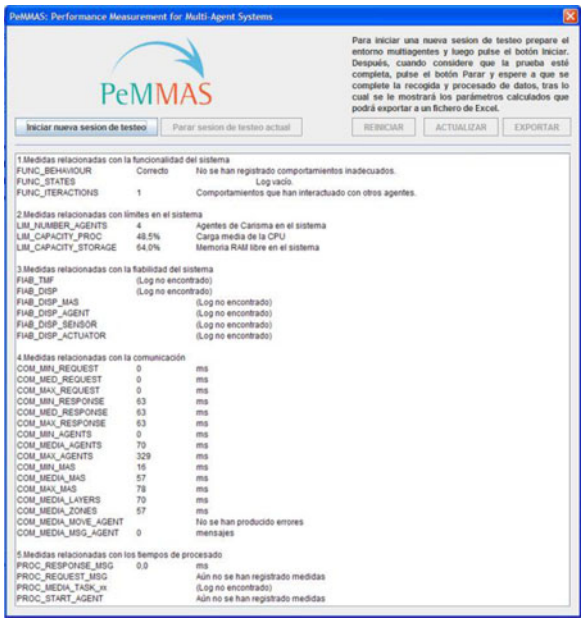

Fig. 6. PeMMAS interface with all the calculated data.

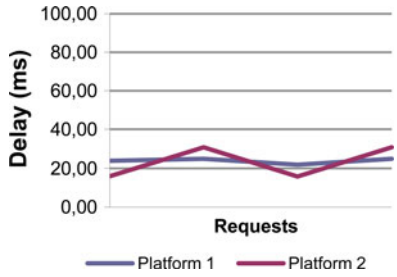

Fig. 7. Delay in the NTP server request. Simultaneous requests in two platforms.

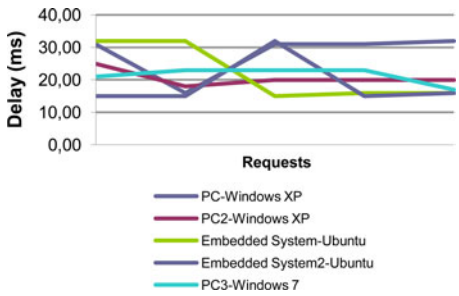

Fig. 8. Delay in the NTP server request. Nonsimultaneous requests of five platforms.

by parameter to obtain the ten text files in which all lines are combined.

After a few minutes, depending on how long each testing session lasts, the ATester will begin to process the data when all lines of the text have arrived. This process varies depending on the parameters used. Some only need to add up all the registered values, while others require averages and others, such as the communication parameters, require a more complicated process since they have to look for pairs of messages (sent/received). All calculated data is stored in a report file to facilitate subsequent study. In addition, a screen presents a summary of all calculated data for rapid user analysis, as seen in Fig. 6.

In the last phase of the process, the ATester creates a new folder with a newly created report file available, along with the other files containing the text of all the different parameters that have been gathered in the testing session. Finally, it compresses the folder and generates a .zip file with all the necessary information for later in-depth study.

At this point, the test is finalized and processed. As a result, all PeMMAS agents die including the ATester and all ATPs. Thus, the system is prepared to initiate a new test session.

IV. System Tests

The PeMMAS system attempts to evaluate the performance of any MAS; therefore, it needs to be optimized to ensure there is no interference with the measurements. To achieve this, we need to address two critical points: synchronization and data dispatch.

A. NTP Synchronization

A crucial aspect in the study of performance output is based on the calculation of flight time of all messages produced by the system. To complete this task successfully, we need to use time indications for the exit and entrance of the messages, therefore checking the time they take to arrive. Nevertheless, as mentioned earlier, taking this step alone is unviable since the watches are not synchronized and could record negative flight times, which is almost impossible.

As a solution, we decided to synchronize all system equipment to an NTP server. These servers use NTP protocol to maintain synchronization with an external element, such as an automatic clock or a GPS clock. In this case, we used the NTP "hora.roa.es" server [23] that provides the official time in Spain.

As we began the execution of the system, and even before loading the JADE element, the NTP server was consulted and the time difference was calculated. After this point, whenever we wanted to use the time, we obtained the most precise results possible in all systems because any differences had already been accounted for.

To check that the system was working properly, we carried out tests with different platforms to measure the delay in each one. The possibility of the time request being made simultaneously from all the physical platforms should be noted, since this could produce eventual delays. Results are shown in Figs. 7 and 8.

As one can see, the delays among the different platforms are consistently very regular, with a 15 to 31 ms delay range per request. Likewise, delays are closer for similar operative systems.

Our specifications indicate that there is a 10–15 ms margin of uncertainty in the results. Nevertheless, although we are referring to communications in the local network, the entrance and exit of the message is actually measured long before leaving the network since an internal JADE code is executed beforehand and this produces delays. For this reason, in the majority of cases, the 10–15 ms in these delays are insignificant.

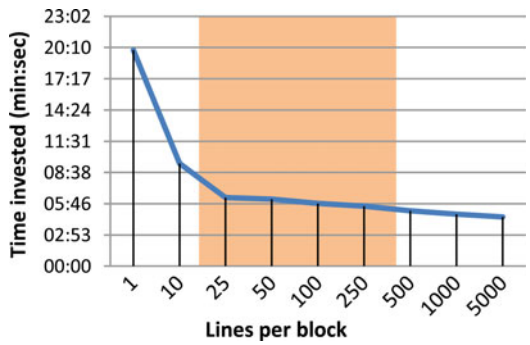

Fig. 9. Time spent sending 50 000 lines of text in various blocks of quantity.

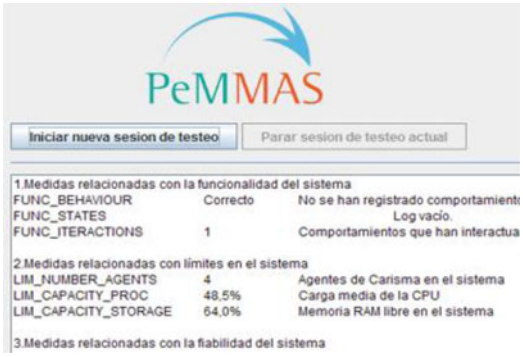

Fig. 10. Detail of the interface with the obtained results.

B. Sending Data

We mentioned earlier that one of the last phases of the entire testing session consisted of sending all of the gathered data to the ATester. This process is relatively important because while some testing parameters have an average value of thousands of pieces of information, others only have a total of two or three. In these cases, the loss of a message enroute is fatal and unacceptable.

This information is sent line by line by reading text files. Bearing in mind that a test session can gather dozens of thousands of messages, and that they have to be sent in a reasonable amount of time, sending the lines one by one is unfeasible because it would take too long.

To overcome these problems, we have implemented a stop&wait protocol in which the ATP waits for a confirmation (ACK), of each message sent to the ATester. This way, the following message is not sent until confirmation is received. Additionally, a time-out has been included; if a message is not received in 100 ms, it is resent with a maximum of three attempts. Instead of sending the lines one by one, they are sent in groups of 25. The idea of using 25 lines comes from the tests carried out in the system to study how it reacts. Results are shown in Fig. 9.

One can see that increasing the number of lines in each message group decreases the logarithmic form of the time it takes for the ATester to receive them all. Nevertheless, when too many lines are sent simultaneously, JADE cannot manage such a large block of messages and begins to discard messages. In contrast, as the number of lines in the block decreases the time taken to gather all the information increases significantly.

To avoid these extremes, we feel that an acceptable interval lies between 20 and 500 lines per block as seen in Fig. 9. This interval collects 50 000 lines in 20-line blocks in the approximately 6 to 7 min used when making blocks of 500. Therefore, bearing in mind that there is a 1-min time difference, we decided to set the number of messages at 25, thus avoiding problems while maintaining an acceptable time. This way, we are not overloading the network, we obtain an acceptable time, and we avoid problems with JADE related to block size.

V. Results

After optimizing the most important aspects of PeMMAS, we tried it out in a real system. As we mentioned earlier, we selected the CARISMA system, a MAS developed at the University of Seville which has a hierarchy structure of communication with various types of agents and can be found distributed among various containers in different physical hardware devices.

PeMMAS was integrated in the interface of the CARISMA user to make it simpler to use. As a result, we were able to initiate the execution of the MAS to be measured with its own interface. After everything had been configured, PeMMAS was executed to begin the tests. While all the data was being gathered, CARISMA continued working without any problem. This is sometimes necessary when something specific that requires external interaction, whether it be real or simulated, needs to be tested.

The results obtained were unexpected in that several communication parameters that should have obtained similar values, did in fact show very different values. However, the unexpected aspects were not provoked by PeMMAS, but by CARISMA. We therefore realized that CARISMA had problems with communication and with the processing of received messages. This meant that the developers of CARISMA were able to make improvements to its system and find solutions to a number of errors. For instance, after detecting that the manufacturer driver of the management software for solar panels was a bottleneck for our system, one adopted solution was to move the affected agent to another kind of higher-performance device.

After a series of modifications to CARISMA, we repeated the testing and this time, we did obtain the expected results. PeMMAS had completed its task and we now know that CARISMA is working properly.

When CARISMA is closed, we will publish our work presenting all the numeric results obtained and the experiences arising during the series of tests.

VI. Conclusion

When we began to develop PeMMAS, we based our thinking on the fact that no tool had been developed to study the performance of multi-agent systems in a distributed way. Now, we can affirm that we have created a tool that:

1) is capable of measuring any multi-agent system that has been developed in JADE;

2) has the capacity to distribute itself and in this way can adapt to any environment;
3) checks whether communication is carried out properly in a multitude of situations;
4) ensures that agent behavior does not produce any execution errors;
5) analyses MAS reaction times in the event of a system crash;
6) measures the use of the MAS and studies system resources, such as the CPU and the RAM memory;
7) generates a report with all the results obtained after processing the data;
8) saves the data gathered in case the user should wish to conduct further study.

Looking ahead, we propose improvements in certain aspects of PeMMAS. For example, we would like to increase the precision of synchronization as much as possible. Additionally, we want to adapt our system to platforms other than JADE to make PeMMAS adaptable to as wide a range of MASs as possible. When we achieve this, we will be closer to our objective: to make PeMMAS capable of measuring the performance of any multi-agent system, regardless of its implementation.

ACKNOWLEDGMENT

The authors would like to thank those at the Aljamir Software Company for their collaboration in the development of the tool and the Java knowledge transfer platforms.

REFERENCES

[1] J. I. Escudero, J. A. Rodríguez, and M. C. Romero, "IDOLO: Multimedia data deployment on SCADA systems," presented at the IEEE Power Eng. Syst. Conf. Expo., New York, NY, USA, 2004.
[2] M. C. Romero, F. Sivianes, A. Carrasco, M. D. Hernández, and J. I. Escudero, "Multi-agent system and embedded system technologies for automatic surveillance," presented at the 10th Int. Conf. Enterprise Inf. Syst., Barcelona, Spain, 2008.
[3] JADE. (2013). *Java Agent DEvelopment Framework* [Online]. Available: http://jade.tilab.com/
[4] FIPA. (2013). *Foundation for Intelligent Physical Agents* [Online]. Available: http://www.fipa.org
[5] M. Wooldridge, *An Introduction to Multiagent System*. Hoboken, NJ, USA: Wiley, 2002.
[6] O. Popoola, "Video-based abnormal human behavior recognition—a review," *IEEE Trans. Syst., Man Cybern. C, Appl. Rev.*, vol. 31, no. 3, pp. 383–391, Aug. 2001.
[7] J. Y. Kuo, H. K. Cheng, Y. Y. FanJiang, and S. P. Ma, "Multi-agent automatic negotiation and argumentation for courses scheduling," in *Proc. IEEE Int. Conf. Fuzzy Syst.*, Jun. 27–30, 2011, pp. 2690–2695.
[8] S. Abras, S. Ploix, S. Pesty, and M. Jacomino, "A multiagent home automation system for power management," in *Proc. Int. Conf. Informatics Control, Autom. Robot., Intell. Control Syst. Optim.*, 2006, pp. 3–8.
[9] A. Carrasco, M. C. Romero-Ternero, F. Sivianes, M. D. Hernandez, and J. I. Escudero, "Multiagent and embedded system technologies applied to improve the management of power systems," in *Proc. Int. J. Digital Content Technol. Appl.*, 2010, vol. 4, no. 1, pp. 79–85.
[10] M. C. Romero, F. Sivianes, A. Carrasco, M. D. Hernandez, and J. I. Escudero, "Managing emergency response operations for electric utility maintenance," *IEEE Ind. Electron. Mag.*, vol. 3, no. 3, pp. 15–18, Sep. 2009.
[11] J. K. Kok, C. J. Warmer, and I. G. Kamphuis, "PowerMatcher: multiagent control in the electricity infrastructure," in *Proc. 4th Int. Joint Conf. Auton. Agents Multiagent Syst.*, New York, NY, USA, 2005, pp. 75–82.
[12] T. Lux and M. Marchesi, "Scaling and criticality in a stochastic multi-agent model of a financial market," *Nature*, vol. 397, pp. 498–500, 1999.
[13] E. Cortese, F. Quarta, G. Vitaglione, and P. Vrba, "Scalability and performance of the JADE message transport system," *Anal. Suitability Holonic Manuf. Syst., Exp.*, vol. 3, no. 3, pp. 52–65, 2002.
[14] X. T. Nguyen, R. Kowalczyk, M. B. Chhetri, and A. Grant, "WS2JADE: A tool for run-time deployment and control of web services as JADE agent services," *Softw. Agent-Based Appl., Platforms Develop. Kits*, pp. 223–251, 2005.
[15] A. Pokahr, L. Braubach, and W. Lamersdorf, "Jadex: Implementing a BDI-Infrastructure for JADE Agents," *EXP—Search Innovation*, vol. 3, pp. 76–85, 2003.
[16] J. M. Such, J. M. Alberola, L. Mulet, A. Espinosa, A. García-Fornes, and V. Botti, "Large-scale multiagent platform benchmarks," in *Languages, Methodologies, and Development Tools for Multi-Agent Systems*. New York, NY, USA: Springer, 2007, pp. 192–204.
[17] O. Hairong, S. Iyengar, and K. Chakrabarty, "Multiresolution data integration using mobile agents in distributed sensor networks," *IEEE Trans. Syst., Man Cybern. C, Appl. Rev.*, vol. 31, no. 3, pp. 383–391, Aug. 2001.
[18] A. Nedic, A. Ozdaglar, and A. P. Parrilo, "Constrained consensus and optimization in multiagent networks," *IEEE Trans. Autom. Control*, vol. 55, no. 4, pp. 922–938, Apr. 2010.
[19] E. Serrano, J. J. Gómez-Sanz, J. A. Botía, and J. Pavon, "Intelligent data analysis applied to debug complex software systems," *Neurocomputing*, vol. 72, no. 13–15, pp. 2785–2795, 2009.
[20] K. Chmiel, D. Tomiak, M. Gawinecki, P. Kaczmarek, M. Szymczak, and M. Paprzycki, "Testing the efficiency of JADE agent platform," in *Proc. Int. Symp. Parallel Distrib. Comput. Conf.*, Los Alamitos, CA, USA, 2004, pp. 49–57.
[21] INGENIAS Development Kit. (2013). [Online]. Available: http://grasia.fdi.ucm.es/main/?q=en/node/127
[22] M. Khalgui and H. M. Hanisch, "Reconfiguration protocol for multiagent control software architectures," *IEEE Trans. Syst., Man Cybern. C, Appl. Rev.*, vol. 41, no. 1, pp. 70–80, Jan. 2011.
[23] ROA Time. (2012). [Online]. Available: http://en.wikipedia.org/wiki/ROA_Time