Scalability and Performance of JADE Message Transport System

E.Cortese
Telecom Italia LAB
Centro Direzionale isola F7
80143 Naples – Italy
+390819718364
Elisabetta.Cortese@TILAB.com

F.Quarta
Telecom Italia LAB
Centro Direzionale isola F7
80143 Naples – Italy
+390819718362
Filippo.Quarta@TILAB.com

G.Vitaglione
Telecom Italia LAB
Centro Direzionale isola F7
80143 Naples – Italy
+390819718354
Giosue.Vitaglione@TILAB.com

ABSTRACT

In this paper, we present the results of scalability and performance measurements of the messaging transport system of JADE, a FIPA compliant agent platform. A brief description of JADE messaging architecture, the caching mechanism, and the testbed used for the measurements is provided. The results of message round-trip measurements for several platform and load configurations, and different MTPs are then presented and analyzed. The obtained results show that JADE well performs in scalability in several scenarios (intra- and inter-platform). Messaging performance for intra-platform configuration highlight that JADE does not introduce relevant overhead compared to its underlining technology Java RMI, and that it well exploits different configurations of the agent platforms.

General Terms

Algorithms, Measurement, Performance, Design.

Keywords

Scalability, Agent communication, Applications, Agent messaging, Performance, Benchmark, Roundtrip, Measurement.

1. INTRODUCTION

Software agents provide an ideal mechanism for implementing heterogeneous and complex distributed systems. Such systems cannot be easily developed using traditional software technologies because of their limits in coping with distribution and interoperability. The agent technology is well suited for applications that are based on communication of loosely-coupled systems.

The Foundation for Intelligent Physical Agents (FIPA) [1] is an international non-profit association of companies and organizations sharing the effort to produce specifications of generic agent technologies. Three mandatory roles are present into a FIPA compliant agent platform: AMS (Agent Management System) that controls access and use of the platform, DF (Directory Facilitator) that provides yellow pages service and ACC (Agent Communication Channel) that provides the Message Transport Service for FIPA ACL messages delivery between agents living into different agent platforms. In order to promote interoperability between agent platforms, a number of standard MTPs (Message Transport Protocols) and MTP interfaces have been defined by FIPA, in particular an MTP based on the IIOP protocol defined by OMG, which is extensively used in this work. FIPA does not define nor require a specific protocol for intra-platform message delivery and each implementation can choose any IMTP (Internal Message Transport Protocol).

JADE (Java Agent DEvelopment Framework) [2] is a FIPA compliant agent platform that makes multi-agent systems development easier by providing a middleware with a set of standard agent platform services, including the ACC. JADE is completely written in Java and can run on both PCs and wireless devices that support Java ME (Micro Edition) using the package developed by the Leap Project (Lightweight Extensible Agent Platform) [3].

In this paper we briefly describe JADE messaging architecture (section 2) highlighting some relevant design choices adopted by JADE developers. We describe the testbed we used for measurements (section 3), and discuss the results. The results provided by JADE have been compared to RMI (Remote Method Invocation) as a baseline for appreciating the values of the reported absolute times (section 4). Finally we discuss the scalability that JADE has showed during this tests (section 5).

2. JADE Messaging Architecture

JADE is a "platform centric" middleware for multi-agent system development. In fact JADE agents are executed in a run-time environment where they share resources and services like threads scheduling and messaging support.

A JADE platform is a distributed environment composed of several run-time c*ontainers* launched over one or more hosts across a network (figure 1). A Java Virtual Machine can host one or more containers, and each container provides the run-time environment and the services for one or more agents. In the platform, an important role is played by the *Main Container* where the FIPA service agents live (i.e. the AMS and the DF agent). The GUID (Global Unique IDentifier) is the logical reference of an agent, independent to the agent location, that is used into the ACLMessage and then resolved by the platform into a physical address transparently to the programmer.

For a better understanding of JADE messaging architecture (figure 2), we distinguish between intra-platform and inter-platform agents communication.

The first case involves agents living in the same platform and JADE uses its internal message transport protocols (IMTPs) for implementing delivery services. In this delivery process the middleware chooses the proper message transport mechanisms, taking into account the location of the agents, in order to minimize delivery time. According to that, JADE delivers messages as follows:

- ✓ using event passing when both the sender and the receiver agents live in the same container. That implies also that the message is not serialized, but cloned, and the new object reference is passed to the receiver;
- ✓ using RMI for inter-container communications, i.e. when sender and receiver live in different containers.

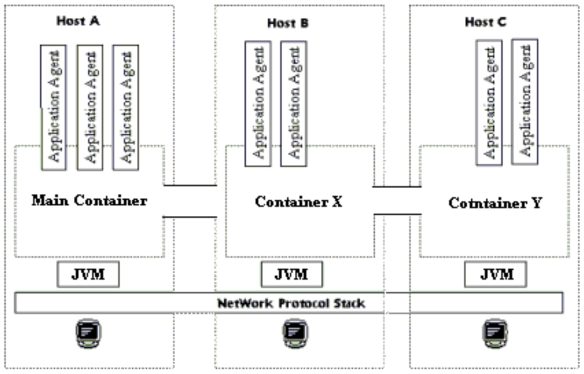

Figure 1: A JADE platform

In the inter-platform scenario, interaction among agents is achieved by the Agent Communication Channel (ACC), which is physically distributed across all the containers of the platform. In fact, each container can be launched with one or more message transport protocols (MTPs) and the entire platform is able to internally route the messages and select the best MTP (which might be even on a different container) for each situation. The following MTPs are currently available for JADE:

- ✓ CORBA IIOP MTP based on standard Sun ORB provided with the JDK. (the default installation)
- ✓ CORBA IIOP MTP based on ORBACUS [7].
- ✓ HTTP-based MTP.

MTPs can also be activated and deactivated at run-time for specific administrative purposes (e.g. temporarily disabling a port or changing the port for external communication). This functionality, combined to the internal routing capability, provides a transparent way to distribute the platform partly outside a firewall (i.e. only a container with the external MTP and without agents) and partly behind it (i.e. all the containers with the agents).

JADE provides a Java interface both for implementing new ad-hoc IMTP and MTP. An alternative IMTP, for example, has been implemented for JADE integrated with LEAP in order to provide inter-container communication in wireless environments, where RMI is not available.

A requirement for JADE was to avoid the main container to be a bottleneck for messaging and platform management. Thus the ACC is distributed and MTPs are "pluggable" to any container, not necessarily "Main Container". Also very important is the cache in any container for mapping a GUID with a physical address. The main container is the only one that has got the complete GUID-physical address association for the whole platform, and it is contacted by the other containers only in a cache-miss case, that happens only the first time a message is sent to a certain agent on a different container. JADE performs all that transparently to the programmer, and the caching mechanism provides good performance, as also shown by the present work.

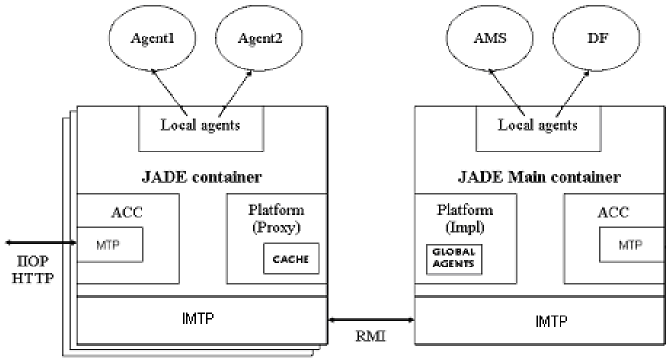

Figure 2: Components of Jade Messaging Architecture

3. EXPERIMENT DESCRIPTION

3.1 Goal

We intend to evaluate the performance of the JADE messaging subsystem for typical configuration scenarios.

3.2 Testbed

We measured the roundtrip time defined as the required time for a circular exchange of an ACL message between a Sender agent and a Receiver agent.

In order to evaluate the scalability of the platform, we started with a single couple Sender/Receiver, then we increased the number of couples and observed how the *roundtrip* time grows.

For each measurement we want to obtain the avgRTT (average roundtrip time) defined as the total measurement time divided by the number of times a message is exchanged for each couple and by the number of couples. In other terms, avgRTT is the average time needed to send a message and receive the reply.

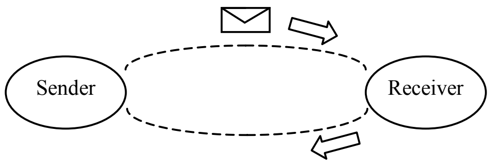

Figure 3: ACL message round trip

As shown in Figure 4, the Sender i-th communicates only with Receiver i-th, every couple exchanges 10000 messages. The content field of each message is filled with a string of 7 characters.

Both agents are implemented by using the *CyclicBehaviour* class (the JADE abstraction for a repetitive cyclic agent task) to handle incoming messages. In detail, the steps executed by the two agents, *sender* and *receiver,* are the following:

The *Sender* agent that has the *initiator role*:

Sender:

```
Create a message to send
Take start time
For all number of message to send
        Send message to receiver
        Wait the reply message from receiver
Take finish time
```

The Receiver agent has *responder role*:

Receiver:

```
wait the message from sender
send the reply message back to sender
```

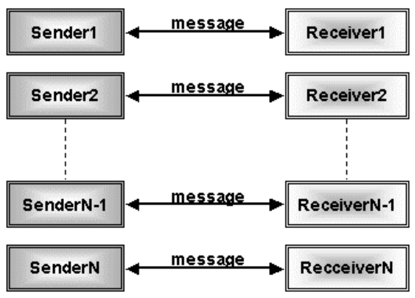

Figure 4: Testbed Architecture

3.3 Scenarios

The measurement is repeated for several configurations, with the agents situated into the same platform or in different platforms. So, we distinguish between:

- intra-platform communication
- inter-platform communication

When all agents run into a single platform, we have two sub-cases:

- Communication between agents both living in the same container
- Communication between agents living in two different containers.

3.4 Measurement issues

3.4.1 *Time measurements in Java*

In order to measure time intervals we used the Java method: *long System.currentTimeMillis(),* that returns the number of milliseconds passed since 1st Jan 1970, as common in Unix systems. Different JVM implementations can provide different precision levels. Sun SDK 1.4 on Windows 2000, for example, provides a precision of 10ms.

Since the duration of our phenomena is in the order of magnitude of milliseconds, we need to measure the roundtrip time as average, taking the time of a message to complete N roundtrips, and dividing it by N. The value of N has to be chosen high enough so that the measurement error introduced by Java does not influence our measurement.

3.4.2 *Measurement at "Full Load"*

During a measurement we can identify three intervals, as shown in Figure 5. Na(t) is the number of active agents and N is the total number of agents.

During T_1 all the agents are sequentially created (and start competing for the CPU), and after a little while (thicker line) they actually start exchanging ACL messages. In this phase two different phenomena can be experienced that tend to distort the measurements: the measured avgRTT tends to be lower (i.e. faster exchange) than reality because not all the couples are born yet; the avgRTT tends to appear higher (i.e. slower exchange) because agent's creation takes CPU.

During the interval T_2 all the couples are created and are ready to exchange messages.

In the T_3 phase the measurement is again influenced by the lower number of agents competing for the CPU, and by the agent destruction time, similarly to the T_1 phase. We want to focus on the messaging subsystem performance, so other effects on avgRTT should not be taken into account. Otherwise we would include someway the agent creation/destruction performance in the avgRTT measurement.

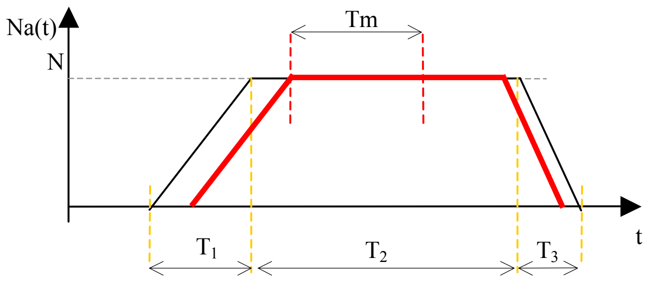

Figure 5: Na(t) is the number of active agents during the three considered intervals. The thicker line shows the number of agents that are actually sending messages.

Given the aforementioned, for each couple we measure the avgRTT only during the interval T_m when the system is working at "Full Load" i.e. all the couples are actually exchanging messages. In particular we make T_m last until 10000 messages are exchanged. The algorithm has been implemented so that a couple, after taking the "finish time" at the and of T_m, does not stop exchanging messages in order to keep the "Full Load" condition for the other couples that are still working in the T_m phase. The program stops as soon as all the couples completed the Tm interval. This has been implemented by using a `static` variable in the Sender class, incremented by each agent when starts sending messages. By doing that, each sender knows how many couples are actually sending messages.

It has been verified that values obtained with the mechanism we used, are actually different than calculating the avgRTT as the total time (T_1+ T_2+ T_3) divided by (exchanged messages * number of couples).

3.4.3 *Just-in-Time compilation effects*

The JVM we used (Sun SDK 1.4) offers, as default, the *HotSpot* technology, that provides a "Just-in-Time" compilation. Most used parts of code are compiled at run-time by the JVM to native code, in order to speed up their execution.

Running a roundtrip time measurement with 50 couples of JADE agents, with JIT (as default) or disabling it (-Xint option), for 10000, 30000 and 50000 trips, we obtained these values:

# Trips	*10000*	*30000*	*50000*
With JIT	13,4	11,9	11,6
NO JIT (-Xint)	64,1	63,9	63,8

Table 1: Measured values of avgRTT for different numbers of trips (times a message is exchanged) for each couple. Considered 50 couples, values in ms.

The variations between 10000 and 50000 trips are:

<u>With JIT:</u> -12,9 %   <u>NO JIT:</u> -0,5 %

The first value shows that the performance improvement due to the JIT compilation is higher for longer measurement. This is because at the beginning the compiler takes time (and CPU) to produce native code, but in the long run this becomes more and more convenient because the compilation overhead is spread over a longer period of time. On the other hand, without JIT compilation, as we expected, the avgRTT doesn't vary significantly with the number of trips. Another consideration is the big performance improvement due to the JIT compilation for our testbed.

In order to make the results comparable, we run all the measurements with 10000 trips for each couple of agents, without disabling the JIT compilation.

3.5 Hardware & software configuration

We used two HP VECTRA personal computers connected by a 100Mbit/s Ethernet LAN. The two hosts were in a proper VLAN so that the network traffic generated by other host did not affect the measures. The following table provides information, obtained with MS System Info, which shows the characteristics of the two PCs.

Model	HP Vectra
Processor	x86 Family 6 Model 8 Stepping 6 GenuineIntel ~800 MHZ
BIOS version	PhoenixBIOS 4.0 Release 6.0
Total memory	256 MB
Operating System	Microsoft Windows 2000 Professional
OS version	5.0.2195 Service Pack 1 Build 2195
Java	Sun SDK 1.4
JADE	2.5

Table 2: Hardware and Software testbed configuration

4. EXPERIMENT RESULTS

4.1 JADE

4.1.1 Intra Platform

In this section we present the measured round-trip time, when the agents run in the same platform. There are three result sets, as shown in Figure 6, one for each configuration:

- One container
- Two containers on the same host
- Two containers on two different hosts

All the following figures are in milliseconds and showed low standard deviations during the tests. For every graph, the X axis contains the number of couples and the Y axis the related average roundtrip time (avgRTT) expressed in milliseconds (ms).

At first glance, we can see that the avgRTT measured when the agents run into same container is very low, indeed JADE optimizes on agents localization and uses event passing when the agents are in the same container. Furthermore, the middleware showed a linear growth of the roundtrip time as function of the number of couples in all cases, in the considered range.

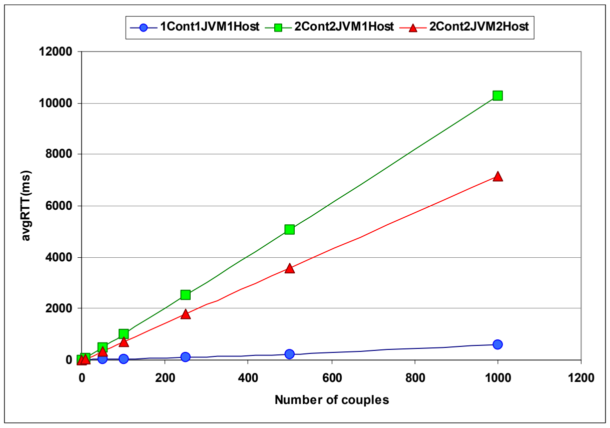

Figure 6: Round Trip time - Intra Platform communication

When the communication happens between agents on two different containers, JADE uses RMI to send the messages. Roundtrip time for the two hosts configuration results lower than the 1-host case because the computation load is split between two CPUs.

The low growth rate of avgRTT for the *same container* configuration, compared to the other cases, shows that the messaging architecture of this middleware can support high load, without heavy performance degradation.

4.1.2 Inter Platform

The graph in Figure 7 shows the roundtrip time between agents on different platforms, each living on a different host.

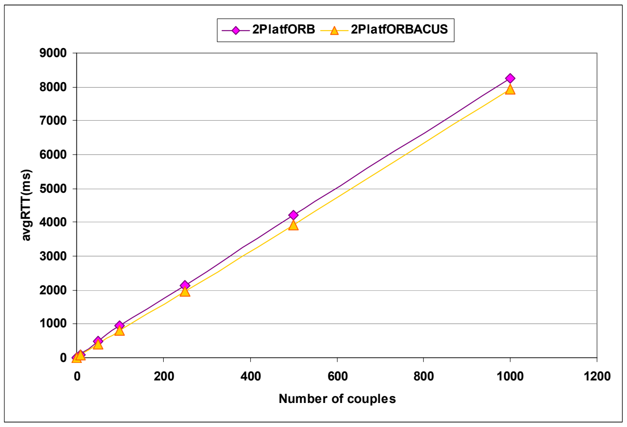

Figure 7: Round Trip Time -Inter Platform communication.

Since it is possible to change transparently the Message Transport Protocol (MTP), we have compared the results obtained by using the ORB produced by Sun Microsystems to ORBacus produced by IONA without affecting in any way the agents' code. The results show that ORBacus performance are slightly better.

JADE uses MTPs that are compliant to FIPA specifications. This implies that JADE for inter-platform communication has to add an envelope to the message, code it via StringACLEncoding before delivering and then parse it at the receiver side. Considered that the roundtrip time for inter-platform scenario is very similar to the intra-platform one, the FIPA-MTP based upon IIOP has good performance and JADE implementation of the FIPA specifications is very efficient.

4.1.3 Intra Platform and Inter Platform

Comparing the results for the two containers on two hosts configuration, to the case with two containers belonging to different platforms on two hosts, we can see that performance are similar (Figure 8).

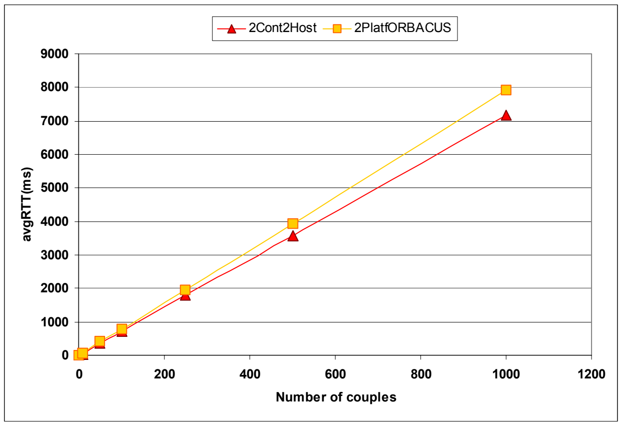

Figure 8: Comparison between 2 Container and 2 Platform communication

Multi agent systems are considered to be a valuable technology for integrating loosely coupled systems, like information systems of different departments or B2B process integration. This kind of applications requires agents on different platforms that might be under the control of different organizations, to interact efficiently. Since JADE inter-platform communication performs as well as the intra-platform one, the middleware can be efficiently used as enabling technology for these scenarios.

4.1.4 Comparison with a RMI implementation of the testbed

As term of comparison we implemented a version of the testbed by using RMI without the facilities offered by the JADE middleware. In more details, we set up this implementation for two main reasons:

- have a baseline for appreciating the values of the reported absolute times, based upon the most used JAVA features for agent implementation, like multithreading and Java-RMI;
- try to understand the overhead introduced by the standard JADE IMTP in respect to its underlining technology.

The implementation is composed of two agent classes that are substantially Java remote objects with an active thread (Figure 9). In order to perform the circular exchange of messages, each agent has a message queue and a thread that monitors its queue. When the thread is notified that there is a message in the queue, it calls a remote method on the other agent with an ACL message object as parameter, in order to fill the queue of the counterpart. This loop is repeated several times to get the avgRTT.

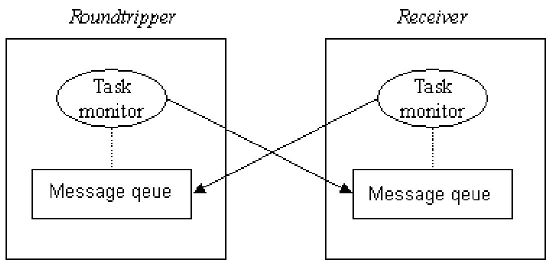

Figure 9: Implementation of the RMI testbed

We have considered two scenarios: "agents" on the same java virtual machine and "agents" on two different hosts. These two configurations show the effect of using the same transport mechanism for communication between agents that are near or far. "Near" means that agents run in the same JVM/Host and "far" means that they are executing on different ones. Active objects on the same JVM, communicate always through RMI, and don't take advantage of agents' locality.

The obtained results (Figure 10) show how better is JADE in exploiting the locality of the communication in respect to the RMI implementation of the testbed. Application developers can take advantage from that, getting better performance by grouping in the same container agents that need to interact a lot. Agent mobility can be also used for dynamically grouping agents at run-time.

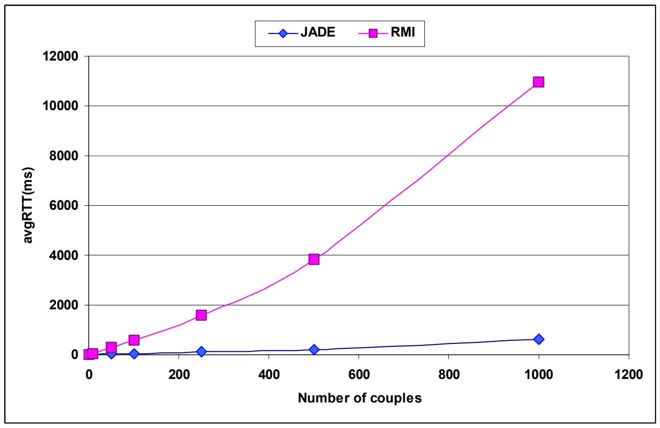

Figure 10: Comparison for "near" agents

Considering agents on two different hosts, JADE shows higher roundtrip time (Figure 11), but still has the same order of magnitude. This is due to the higher level of abstraction introduced by the middleware to support message routing based on agent unique identifiers, encoding of ACL message (as request by FIPA specification) and the possibility to use other internal message transport protocols than RMI. Our RMI testbed implementation simply relies on rmiregistry to locate the agents.

It is worth to notice that this overhead introduced by JADE is well balanced by the fact that the JADE platform hides to the developer all tedious issues of dealing with remote method calls and interfaces, thread synchronization, resulting in a shorter development time.

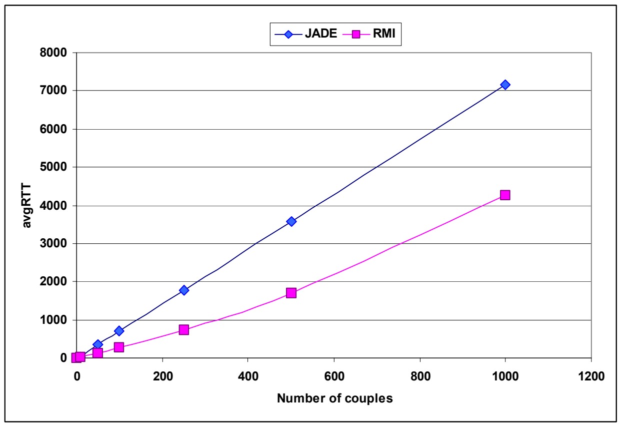

Figure 11: Comparison for "far" agents

5. CONCLUSION

We developed a simple testbed in order to evaluate performance and scalability of JADE messaging subsystem. Specifically, we measured the average roundtrip time for a message exchange between a couple of agents. The algorithm we used and some measurement issues were described. The results of measurements made using several configuration scenarios, both intra- and inter- platform, were analysed. In order to have a term of comparison, some of the obtained results were compared to the performance of an implementation of the testbed based on agents as simple "active objects" communicating through RMI.

We found that, in the considered range, JADE scales linearly, equally well in all the configuration scenarios, and its scalability is similar to RMI when agents are distributed across the network.

JADE messaging performance, when both sender and receiver agents live in the same container, are very good and much superior compared to those provided by the RMI experiments (see Figure 10). On the other hand, JADE inter-container communication provides performance comparable with pure RMI with an overhead well balanced by the benefits of a shorter development time and ease of use.

Communication between different platforms, via the IIOP-MTP, does not suffer any performance degradation, and enables an efficient integration of Multi-MAS. In fact, the measured values are comparable to JADE inter-container transport based on RMI. Furthermore, it has been easy to change message transport protocol for inter-platform communication, plugging-in a different MTP, without affecting the testbed code.

We also measured a little performance improvement from Sun ORB implementation to Iona ORBacus.

Locality of agents allows an efficient MAS deployment improving messaging performance by distributing properly the agents into containers at design time or, by using intra-platform agent mobility provided by JADE, also at run-time.

Finally, these measurements showed good JADE message system performance, and highlighted that it is a good candidate for developing heavy-load distributed applications.

All the source code of the performed tests can be downloaded from the official JADE web site. [2]

Tests will be extended as future work to evaluate other MTPs, in particular the HTTP-based, and to finalize some specific design choices of JADE, such as cooperative scheduling of the agent tasks (i.e. the behaviours).

6. ACKNOWLEDGMENTS

We would like to thank the JADE developers team for the great support during our work and all people who contribute to JADE growth by actively participating to the mailing list.

We acknowledge that this work has been partially supported by the Italian M.I.U.R. (Ministero dell'Istruzione dell'Università e della Ricerca) through the Te.S.C.He.T. Project (Technology System for Cultural Heritage in Tourism).

7. REFERENCES

[1] FIPA - Foundation for Intelligent Physical Agents. http://www.fipa.org.

[2] JADE, Java Agent DEvelopment framework. http://jade.cselt.it

[3] LEAP, Lightweight Extensible Agent Platform

http://leap.crm-paris.com/

[4] F.Bellifemine, A.Poggi, G.Rimassa, P.Turci, "An Object Oriented Framework to Realize Agent System": Proceedings of WOA 2000 Workshop, Parma, May 2000, pagg. 52-57.

[5] F. Bellifemine, A. Poggi, G. Rimassa. "Developing Multi-Agent Systems with a FIPA-compliant Agent Framework". Software: Practice & Experience, 31:103-128, 2001.

[6] R.Deters, "Scalability & Multi-Agent Systems", 2nd International Workshop Infrastructure for Agents, MAS and Scalable MAS. 5th int.conference on Autonomous Agents, May-June 2001.

[7] ORBacus, IONA Techologies, http://www.orbacus.com/

[8] Niek Wijngaards, Maarten van Steen, Frances Brazier, "On MAS Scalability", Proc.2nd Int'l Workshop on Infrastructure for Agents, MAS and Scalable MAS. May 2001.

[9] P.J.Turner, N.R.Jennings, "Improving Scalability of Multi-Agent Systems", Proc.1st Int'l Workshop Infrastructure for Scalable Multi-Agent Systems. June 2000.

[10] O.F.Rana, K.Stout, "What is Scalability in Multi-Agent Systems", Autonomous Agents 2000, June'00, ACM Press.

[11] L.C.Lee,H.S.Nwana,D.T.Ndumu, P De Wilde , "The stability, scalability and performance of multi-agent Systems", BT Technol J Vol 16 No 3 July 1998 94

[12] O.Shehory, "A Scalable Agent Location Mechanism", Intelligent Agents VI, vol.1757 of Lecture Notes in AI, pages.162.172, Springer-Verlag, Berlin, Germany, 1999

[13] B.Neuman, "Scale in Distributed Systems", in Readings in Distributed Computing Systems, pag.463-489, IEEE Computer Society Press, Los Alamitos, CA, 1994