Testing the Efficiency of JADE Agent Platform

Krzysztof Chmiel[a], Dominik Tomiak[a], Maciej Gawinecki[a], Paweł Kaczmarek[a], Michał Szymczak[a] and Marcin Paprzycki[a*]
[a]Department of Mathematics and Computer Science, Adam Mickiewicz University, Poznań, Poland
[*]Computer Science Department, Oklahoma State University, USA
E-mail: d118993@atos.wmid.amu.edu.pl, marcin@cs.okstate.edu

Abstract

Agent oriented programming is often described as the next breakthrough in development and implementation of large-scale complex software system. At the same time it is rather difficult to find successful applications of agent technology, in particular precisely when large-scale systems are considered. The aim of this paper is to investigate if one of the possible limits may be the scalability of existing agent technology. We have picked JADE agent platform as technology of choice and investigated its efficiency in a number of test cases. Results of our experiments are presented and discussed.

1. Introduction

For a number of years now, researchers promise that the agent technology is about to change the ways we construct software [2, 3] as well as have a much broader impact on the field of human-computer interaction [4, 5]. Some of the principle areas software agent technology is expected to impact are [1, 2, 3, 4, 5]:

- development and maintenance of complex systems,
- resource management,
- delivery of personalized content,
- e-commerce on a large and small scale.

Obviously, this list is far from exhaustive, however, the breadth and depth of these areas supports the claim that agent technology, if successful, can become the next "extreme event," leading to breakthroughs in a number of fields. The agent paradigm also promises to add a new dimension to our interaction with computers. Here, the promise of being able to deal with the information overload resulting from the exponential growth of the information available on the Internet, which has been pledged in the influential work of P. Maes [5], is particularly tempting.

Unfortunately, as it is easy to see, almost 10 years after publication of [5], promises furnished there did not materialize (regardless of the rapidly increasing number of conferences, workshops, publications, etc). To the contrary, it is relatively difficult to point to a successful large-scale implementation of agent systems (as understood in [1, 2, 5]). Moreover, what is particularly revealing, agent systems described in [5] as successful implementations of agents, for one reason or another, have never spread beyond the MIT Media Laboratory.

The starting point for this paper was an exchange of messages in one of electronic discussion groups devoted to a particular agent platform. One of the participants described an e-commerce system under development. In this system a personal agent was to be devoted to (instantiated for) each user logged into the system. The question was therefore asked if it is possible to scale the platform to 500+ agents. The response, from someone who clearly was a practitioner of agent-system applications was that "this is a wrong way of looking into the problem and one should not expect realistically to scale an agent application to this size." This response is fascinating as it seems to contradict one of the most basic tenets of agent system development where it is exactly that each user should be served by his/her "own" personal agent [3, 5]. Is it thus really the case that while agents are to be the breakthrough in development of software for large complex systems, it is also the case the using currently existing technology one cannot implement large-scale software systems?

This obviously is a question of scalability of agent systems. There exist a number of papers that discuss various aspects of this problem [6, 7, 8, 9, 10]. However, here we are not interested in an almost philosophical discussion of what is agent scalability found in most of these papers. We prefer to follow a more pragmatic route, where "a good agent system is an implemented agent system" (see also [11]). In this way we follow and expand work reported in [12, 13]. While there, focus was only on messaging between agents, we, first, study different scenarios involving messaging and also add tests of efficiency of agent creation and migration.

A methodological remark is in order. Since there exists no benchmarking suite to test performance of agent systems (similar to these found in scientific

computing; in particular in computational linear algebra), there is an open question as what to measure and why. Scenarios proposed here are not designed to necessarily become such benchmarks. Rather, we were interested in getting a broad understanding as to how our agent platform of choice behaves when the number of messages and agents is increasing as well as obtaining some general assessment of efficiency of agent migration.

To obtain such an insight we have selected one of the best currently existing agent platforms, JADE version 3.1 [13, 14] and "stress-tested" in five scenarios, that can be divided into two groups: two of them are focused on message exchanging capabilities (results reported in the next section) and three of them concentrated primarily on agent creation and/or migration performance (results reported in Section 3).

2. Message exchange performance

In this section we present results of tests that were aimed at testing the messaging capabilities of JADE. These tests differ from, but follow in spirit these reported in [12]. The main rationale behind them is as that in agent-based systems functionality is divided into agents (each agent is responsible for a particular function of the system e.g. search agent, query agent, database wrapper agent etc.) [3]. These agents coordinate their actions and/or communicate by exchanging messages. Assuming that a large number of agents are to be used, a large number of messages are to be expected. We therefore try to find out what is the message-load efficiency of JADE agent platform.

2.1 Spamming Test

The first test is very simple and is designated to flood the system with *Agent Communication Language* (ACL) messages [16]. Here *spammer* agents send to *user* agents a large number of messages. The general scheme of interaction between three *spammer* agents and three *user* agents is illustrated in Figure 1.

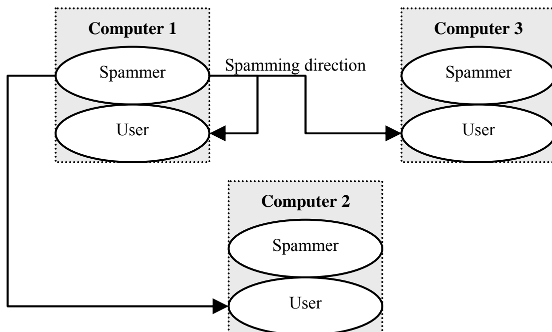

Figure 1. Spamming scheme

On each computer participating in the test a pair consisting of a *user* agent and a *spammer* agent is generated. At a given time all *spammer* agents starts sending messages to all *user* agents (including these existing on their own machines). For example, in Figure 1, *spammer* agent from Computer 1 sends messages to *user* agents residing on Computers 1, 2 and 3. Similarly *spammer* agent residing on Computer 2 sends messages to *user* agents residing on computers 1, 2 and 3 etc. In the JADE platform all posted messages are put in a receiver message queue [14, 15] and then it is processed by the receiver (see Figure 2).

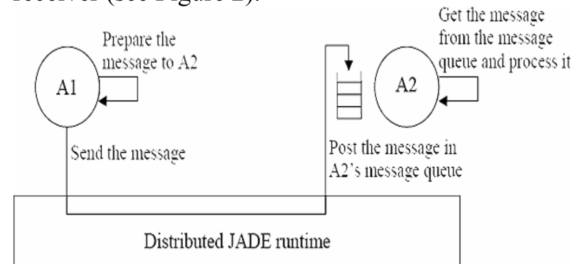

Figure 2. JADE message processing scheme

To measure the performance we utilize a *starter* agent which initiates the spamming process and measures time. During the execution each *spammer* agent broadcasts a certain number of messages and the total time of this broadcast is measured. Separately the time of processing of all messages flooding the system is also measured.

Our tests were performed on 8 Sun workstations each with an UltraSparc III processor running at 300 MHz and 192 Mb RAM. All these machines were Internet-connected through a Cisco switch with full backplane 100 Mbits/s transmission rate. We have used ACL messages with content consisting of 300 ASCI characters. A total of 5000 messages were sent by each *spamming* agent to each *user* agent.

Experimental results are summarized in Table 1 and Figure 3.

Agent pairs	Spamming time [ms]	Receiving time [ms]
2	40034	87053
3	24440	141778
4	25128	217501
5	25217	313625
6	28843	448181
7	35164	634847
8	40624	821341

Table 1. Message sending and receiving times

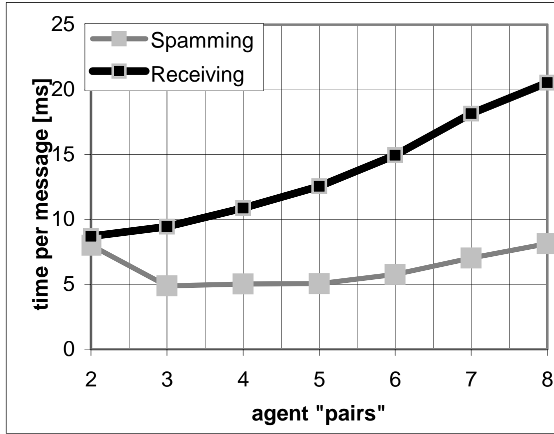

Figure 3. Average message sending and receiving times for 2-8 machines (and thus spammer-user pairs); calculated by dividing total times by a number of sent messages.

A number of observations can be made. (1) For the (relatively small) number of computers used the total "spamming time" practically does not depend on the number of recipients. This is related to the broadcast command used. It is only after the total number of spamming agents becomes larger than 5 when the spamming time increases. (2) As the number of agent pairs increases, the receiving time starts to increase immediately. While we were not able to confirm this, the message receiving time seems to be increasing slightly faster than linearly. This probably because in the case of receiving a message, it is first put in a message queue and only then processed by an agent (see Figure 2). (3) When each of 8 *spammers* sends 5,000*8 = 40,000 messages; resulting in a total of 320,000 messages flooding the system (with each message being more than 0.3 Kbytes – message and its ACL wrapper – totaling approximately 100 Mbytes of data), *user* agents are capable to process them in no more than 14 minutes.

2.2 Processing messages with database access

The second series of tests involved messaging and database access. In this scenario it was assumed that a number of *user* agents (*taskSender* in Figure 3) generate tasks to be executed (in this case these tasks consist of inserting information into the database). These tasks are stored in a list contained in a *list* agent. *User* agents do not communicate with the *list* agent directly, but they send their requests to an intermediate agent (agent *B*). Similarly, *worker* agents (*SQLAgent* in Figure 3) obtain tasks from the *list* agent through an intermediate agent *A*. *Worker* agents insert data into a database (*DB* in Figure 3). All communication between agents involves ACL messages. While this scenario is not based in any particular application, it allows us to observe message processing involved in a relatively complicated flow pattern which is depicted in Figure 3. It also involves a real-life constraint of an access to a database.

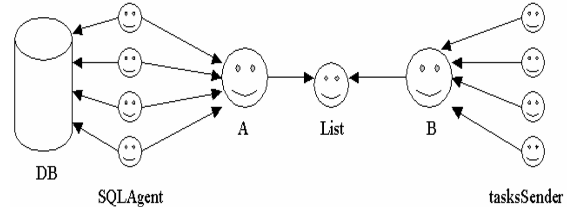

Figure 3. Communication pattern for messaging and database access test.

We have tested the following two situations:

- tasks originate from one *tasksSender* agent while 1 to 4 *SQLAgent* agents are inserting information into a database
- tasks originate from four *tasksSender* agents while 1 to 4 *SQLAgent* agents are inserting information into a database (DB)

Furthermore, each *SQLAgent* agent is located on a different computer and each *tasksSender* agent is located on a different computer as well. Agents *A*, *list* and *B* are all located on the same computer. *TasksSender* agents sent to agent *B* messages consisting of a name of that agent and an SQL statement describing what to insert into the database; a total of approximately 40 ASCI characters. The total number of messages sent was 1000, 10,000 and, attempted, 1,000,000. *List* agent receives messages from agent *B* and appends them into the end of the queue. When agent *A* requests the next task, agent *list* removes the task from the front of the queue and sends it to agent *A*. If there are no tasks stored in the queue one will be send to *A* when one is received from agent *B*. The queue is implemented using the standard java class ***list*** and wrapped in an agent (there is no practical way to persist an "independent" list structure in an agent system). Unemployed *SQLAgent* agents request tasks from agent *A* and start to "work" as soon as they receive a response from agent *A*. Agent *A* requests and receives tasks from the *list* agent.

In our tests we have utilized a PostgreSQL database running on a 2 processor Linux server with 1GHz Intel P3 processors and 1GB of RAM. We have also used the same 8 Sun computers as in Section 2.1. Results of testing flow of 1000 and 10000 messages are presented in Table 2.

Number of *SQLAgent*'(s)	Received by *SQLAgent*'(s) when *tasksSender* ends sending	All received by all *SQLAgent*'(s)
1000 messages; 1 *tasksSender* agent		
1	40	1000
2	68	1000
4	95	1000
1000 messages each; 4 *tasksSender* agents		
1	42	4000
2	69	4000
4	98	4000
10000 messages; 1 *tasksSender* agent		
1	220	10000
2	370	10000
4	574	10000
10000 messages each; 4 *tasksSender* agents		
1	215	40000
2	365	40000
4	558	40000

Table 2. Message flow through a number of intermediate agents

It can be observed that the message flow through the system is rather slow. In the best case, when 40,000 messages have been send, only 558 of them have been processed. This means that at this stage there are still about 39500 messages waiting to be processed. Moreover, these messages are likely to be waiting to be processed by one of the three "central" agents of the system: *A, list* or *B.* Finally, when the number of *SQLAgent* agents increases, the total number of processed messages increases as well. We will investigate this fact in the next series of experiments.

As noted above we have also tried to push 1,000,000 messages through the system. We were not able to do so. In the case of 1 *SQLAgent* agent, agent *B* received its messages too fast and was not able to put them into the *list* agent. It stopped working after receiving about 150,000 messages and returned an error "out of memory." This caused *tasksSender* agent to stop after sending about 200,000 messages, also with an "out of memory" error. Finally, *SQLAgent* agent completed about 19,000 insertions and stopped because the queue was empty. This would indicate that agent *B* was able to send about 19,000 messages to the *list* agent. In the case when two *SQLAgent* agents were used, the same scenario repeated somewhat later. Agent *B* stopped (out of memory) when after receiving approximately 175,000 messages. The *SQLAgent* agents inserted about 17,000 messages each (a total of 34,000 messages processed) and the queue became empty. Agent *tasksSender* stopped working (out of memory) after sending about 230,000. Finally, when four *SQLAgent* agents were used, agent *B* died after receiving about 195,000 messages; *SQLAgent* agents were able to insert approcimately 16000 messages each (total of 58,000) and *taskSender* agent died after sending approximately 240,000 messages. These results indicate that the system in the proposed setup, with the bottleneck at the three "central agents" (*A, list* and *B*) is capable to resist a flood of up to almost 200,000 messages. This total number of messages is somewhat smaller than that reported in the spamming scenario, however in that case the distributed in nature data processing did not have a clearly defined bottleneck.

Results reported thus far indicate that adding *SQLAgent* agents into the system improves its message processing capabilities. We have decided to further test this hypothesis. We have therefore increased the number of *SQLAgent* agents from 1 to 5. The results for processing 1,000 messages are presented in Figure 4, while the results for 10,000 messages are presented in Figure 5.

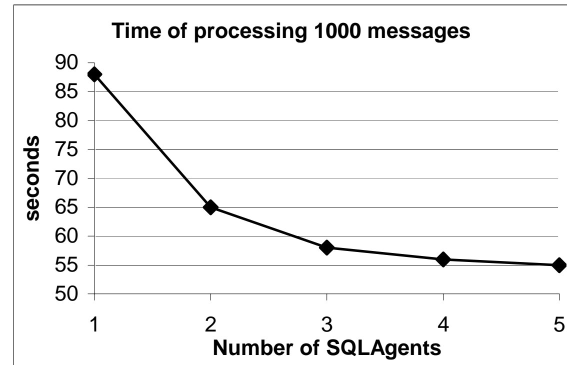

Figure 4. Processing times for increasing number of SQLAgent agents and 1000 messages.

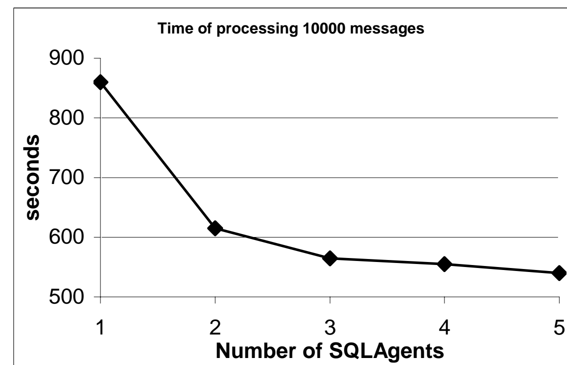

Figure 5. Processing times for increasing number of SQLAgent agents and 10000 messages

The results indicate a substantial performance difference between one and two *SQLAgent* processing tasks. The performance gain can be observed up to four *SQLAgent* agents, but the performance curve is clearly flattening. It can be conjectured that in the current setup adding more than five *SQLAgent* agents will not help. It is worthy noting that processing 10 times as many messages (10,000 vs. 1,000) requires 10 times as much time and that this ratio remains constant regardless of the number of *SQLAgent* agents used (~900 vs. ~90 and 550 vs. ~55 seconds) . This indicates that the processing time of a single task (requesting it and completing it) remains constant regardless of the total number of tasks to be processed.

3. Agent creation and migration performance

The second group of tests was designated to test the ability of JADE to create and migrate a large number of agents. This is in response to the perceived potential of utilization of agents in implementation of large software systems. Additionally, it is often claimed that agent mobility is one of the important factors that make agent systems attractive.

3.1 Agent migration

The first experiment was focused on pure agent migration and was mimicking a relay-race. A fixed number of containers was placed on separate computers. Each container constituted a "place" where agent runners exchange batons. The "race" starts in JADE's "Main-Container" and leas agents to the standard "Container-1" (on a different computer). There agents pass the relay baton by exchanging ACL messages. Second group of agents proceeds to the "Container-2" (on the next computer) and the process repeats. The total race consists of 5 laps and ends in the "Main-Container." Two tests were performed. First, four runner teams were running around an increasing number of containers/computers. Second, an increasing number of runner teams moved around four homogeneous computers/containers.

For the homogeneous setup each container was placed on the Sun workstation (as described above). We have also used a PC with a Pentium 120 MHz processor and 48 Mb of RAM to create a heterogeneous configuration. In Figures 6 and 7 we depict the total migration time for four agent groups and increasing number of containers. Figures 8 and 9 present the results of increasing the number of agent-teams while keeping the number of containers constant.

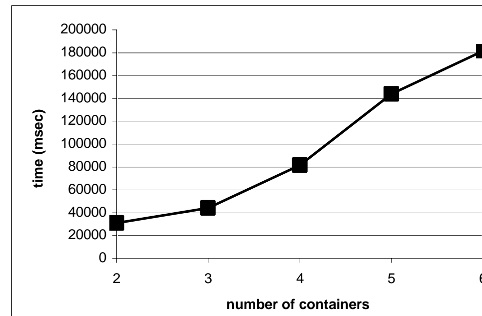

Figure 6. Total migration time; heterogeneous environment; increasing number of containers

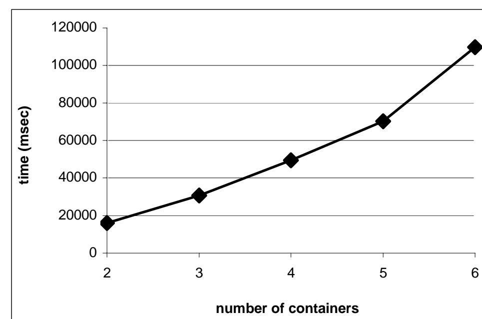

Figure 7. Total migration time; homogeneous environment; increasing number of containers

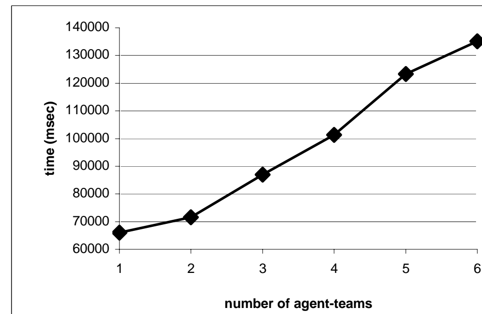

Figure 8. Total migration time; heterogeneous environment; increasing number of agent-teams

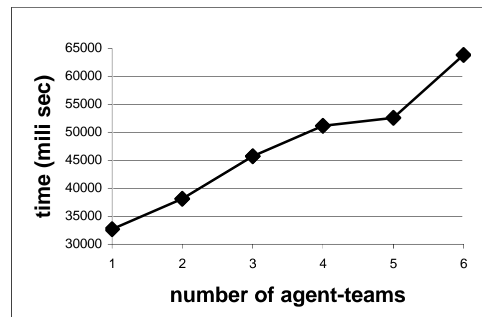

Figure 9. Total migration time; homogeneous environment; increasing number of agent-teams

In all cases the results are practically linear. Some jumps in the total time in the heterogeneous environment can be attributed to the usage of a substantially slower computer. To confirm the linearity we have decided to test the performance for a very large number of runner teams. The results are summarized in Figure 10.

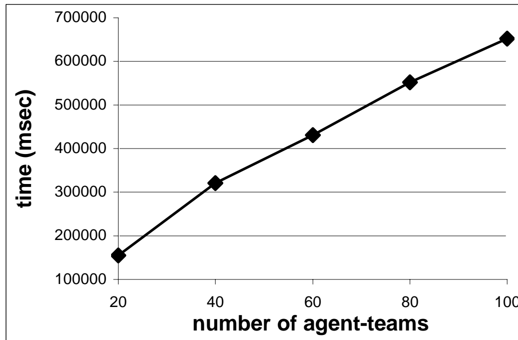

Figure 10. Total migration time; homogeneous environment; large and increasing number of agent-teams

The results are very similar to these in the case of a small number of agents. As the number of agent teams increases, time increases linearly. Note that JADE did not collapse even when a total of four hundreds agents were residing and migrating in four containers on four computers with a relatively small amount of available memory (196 Mbytes).

3.2 Shop performance - agent flooding

In this experiment we wanted to test the performance of JADE when one of its nodes (container) was flooded by a large number of agents from other machines from the platform.

As an example we have used a scenario that can occur when an e-shop is implemented using agents. Here one container was a metaphor of a "store" and held the *MotherShop* agent. This agent was a "store manager" and its main task was to create (and later remove) *Seller* agents for each *Client* which came to the store and requested to be served. We had created also *MotherClient* agents which produced *Client* agents and send them to the store.

The complete scenario of the test is as follows. *MotherClient* agents create *Client* agents that migrate to the store. Here for each visiting *Client* agent the *MotherShop* agent generates a *Seller* agent. Then the *Seller* and the *Client* agents briefly negotiate, via. ACL messages, about goods and prices (in our case the "conversation" was: "do you have beer?," "yes," "please give me one," "here you are," "thank you"). When the "conversation" ends *Seller* confirms transaction with the *MotherShop* agent while the *Client* agent moves back to the node/container in which it was created and reports to its *MotherClient*. Here, *MotherClient* agents generate a given number of *Client* agents and send all of them to flood the shop and then just simply wait for their return. This process is depicted in Figure 11.

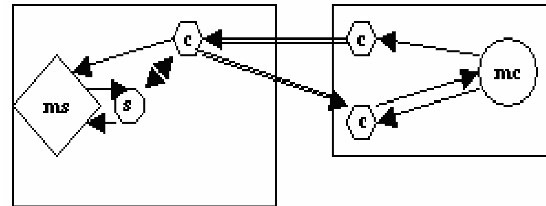

Figure 11. Agent shop experiment; ms – *MotherShop* agent, s – *Seller* agent, mc – *MotherClient* agent, c – *Client* agent.

In our experiments we study the system when (1) we increase the number of *MotherClient* agents, and (2) increase the number of *Client* agents generated in each node/container. We have, again, experimented on the same network of Sun workstations. Each *MotherClient* agent was located on a different machine (in a separate container), while the shop was also located in a separate container on a separate computer. We have also experimented with generating larger number of *Client* agents on a smaller number of host computers and on a smaller number of containers and the results differed only up to 10%. The experimental results (in milliseconds) of system processing 30, 40, 50 and 120 agents generated by 1, 2, …, 5 *MotherClient* agents (in the largest case a total of 600 agents flooding the system) are depicted in Figure 12.

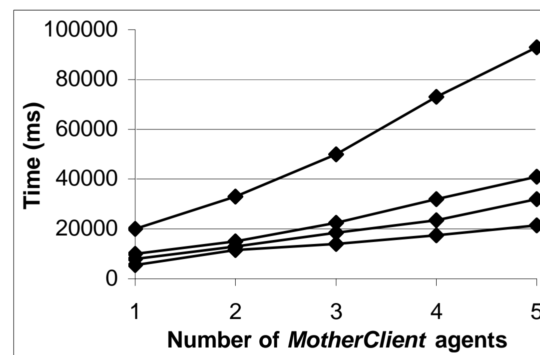

Figure 12. Agent shop experiment; 30, 40, 50 and 120 agents generated by each MotherClient agent

As previously, the processing time is almost linear. We have experimented also with large number of agents in the system and we have found

that the processing time was still almost linear e.g. for 520 *Client* agents the processing time was 75 seconds, while for 1020 *Client* agents 136 seconds. However, we have also found that at approximately 1430 *Client* agents Java generated an "out of memory" exception. To verify the connection of an amount of RAM and numbers of agents we have made a test on a weaker configuration which consisted of PC's with 4 times smaller amount of available RAM. In this test the exception appeared after generating approximately 370 agents; again an almost linear relationship.

3.3 Agent creation and migration within two containers and/or two computers

The results of experiments reported above indicated that JADE is capable of handling a large number of agents, however the question remains open: how many agents can JADE handle? While this question may sound a bit like a proverbial "how many angels can dance on the head of a pin?" it seems relatively important in the context of a relationship between the number of agents and the total available memory. We have therefore implemented a very simple scenario to find an answer:

Start of the process:

AgentA1, which resides in the Main-Container

Actions:

1. *AgentA1* creates AgentB
2. *AgentB* moves to Container-1
3. *AgentB* registers at the Directory Facilitator (DF)
4. *AgentB* sends a message to *AgentA1* confirming arrival at Container-1
5. *AgentB c*lones itself
6. *AgentB* clone repeats steps 3 and 4 above
7. *AgentA1* counts only messages received from "original" agents (not clones) but replies to all of them
8. When *AgentB* and its clones receive the response from *AgentA1* they deregister and terminate
9. *AgentA1* creates as many *AgentB* agents as their number in the previous step plus an additional 50 agents
10. Steps 2 to 7 are repeated for every newly created agent *AgentB*

This process involves a sizable migration between containers as well as an increasing number of agents that are created and exist at the same time in the system. We have experimented with this scenario on a number of machines: the Sun workstation and the PC described above as well as PC with an AMD Athlon processor running at 1.4 GHz, and a PC with AMD Duron processor running at 800 MHz (both these PC's had 256 Mbytes of RAM). We have experimented two situations: (1) when both the Main-Container and the Container-1 were located on the same computer and (2) when they were located on separate computers. The main observations can be summarized as follows:

1. With both containers on a slow PC – the system died after creation of 151 agents, during the migration.
2. With both containers on a Sun workstation – system died after creation of 451 agents, during the migration.
3. With both containers on the Athlon PC – system died during creation and migration of approximately 300 agents.
4. With both containers on different Sun workstations – system died during creation and migration of 501 agents.
5. With the Main-Container on the Athlon PC and Container-1 on the Duron PC – system died during creation and migration of 351 agents.
6. With the Main-Container on the slow PC and Container-1 on the Sun workstation – system lived with over 600 agents, and likely would live longer, but the whole process lasted over an hour and we decided to terminate it

The results are inconsistent. Obviously, when the amount of available memory increases, the number of agents that can be processed increases as well, but then one could ask, why the Athlon PC with 256 Mbytes of memory was not able to handle more agents than the Sun with 196 Mbytes of RAM (in both cases we have used default setups for the JVM)? We have tried to answer this question in additional experiments and the results were rather peculiar, indicating complicated interactions between JVM and JADE, and prevented us from being able to provide a definite answer here.

Even more surprising may seem to be that the PC with 48 Mbytes of memory is able to exist in a system that generates 600+ agents. An explanation seems to be in the typical error generated here: *Error creating agent on destination container. Abort transfer. An RMI error occurred [nested Connection refused to host: 62.21.17.200; nested exception is: java.net.ConnectException: Connection refused: connect]*. The error occurred during agent transfer and involved the RMI port. This means that when a slow PC is generating and sending agents to the Sun workstation "slowly" the Sun can accept them and the RMI port is not

overrun. At the same time in other cases, the creation and migration of agents occurs much too fast for the receiving container to be able to accept them and the RMI port generates an exception.

5. Concluding remarks

The aim of our work was to follow and expand the experimental research outlined in [12] (using JADE 2.5). Here, we have used the most recent JADE 3.1 and in addition to messaging performed experiments related to agent creation and migration. Our main goal was to establish if JADE can be used to follow the research program put forward by Nwana and Ndumu in [11] and be used while developing implement large software systems.

Our tests indicate that JADE is quite an efficient environment limited mostly by the standard limitations of Java programming language, which is interpreted and executed in a Virtual Machine: processor speed, amount of available memory and speed of network connection. The environment itself does not introduce substantial overhead. Executing JADE on a relatively antiquated hardware (PC's with Pentium II processors running at 120 MHz with 48 Mbytes of RAM and workstations with UltraSparc III processors running at 300 MHz with 192 Mbytes of RAM) we were able to run experiments with thousands of agents effectively migrating among eight machines and communicating by exchanging tens of thousands of ACL messages. Furthermore, an increase in the number of agents results typically in a linear increase of processing time.

It has to be stressed that it does not really matter here how realistic or unrealistic our experimental scenarios were. Even if one would like to believe that they are completely unrealistic, they still show how efficient JADE is and that there is no excuse for agent researchers, but to start designing and implementing large software systems, consisting of hundreds of agents and study their behavior. We can do it already today and there is no reason to stop with demonstrator systems consisting of only a few agents. And we believe that this is very good news for the future of agent research.

6. References

[1] J. Hendler, Is There an Intelligent Agent in Your Future?, Nature, http://www.nature.com, March 11th, 1999.

[2] M. L. Griss, My Agent Will Call Your Agent … But Will It Respond?, Technical Report, Hewlett Packard, 1999, http://www.hpl.hp.com/techreports/1999/HPL-1999-159.pdf.

[3] N. R. Jennings, An agent-based approach for building complex software systems, *Communications of the ACM*, 44 (4), 2001, 35-41

[4] T. Berners-Lee, J. Hendler, O. Lassila, The Semantic Web, *Scientific American*, May, 2001, http://www.sciam.com/article.cfm?articleID=00048144-10D2-1C70-84A9809EC588EF21

[5] P. Maes, Agents that Reduce Work and Information Overload, *Communications of the ACM*, 37(7), 1994, 31-40

[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] N. Wijngaards, M. van Steen, F. Brazier, "On MAS Scalability", Proc.2nd Int'l Workshop on Infrastructure for Agents, MAS and Scalable MAS. May 2001.

[8] 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.

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

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

[11] H. Nwana, D. Ndumu, A perspective on software agents research, *The Knowledge Engineering Review*, 14 (2), 1999, 1-18

[12] G. Vitaglione, F. Quarta, E. Cortese, Scalability and Performance of JADE Message Transport System, presented at AAMAS Workshop on AgentCities, Bologna, 16th July, 2002, http://sharon.cselt.it/projects/jade/papers/Final-ScalPerfMessJADE.pdf

[13] S. Rahimi, J. Bjursell, D. Ali, M. Cobb, M. Paprzycki, Preliminary Performance Evaluation Geospatial Data Conflation System, Proceedings of The IEEE International Agent Technology (IEEE-IAT 2003), Halifax, Canada, 2003, 550-553

[14] JADE: http://sharon.cselt.it/projects/jade/

[15] Giovanni Caire, JADE Tutorial: JADE Programming for Beginners, http://sharon.cselt.it/projects/jade/

[16] Agent Communication Language Specification, http://www.fipa.org/repository/aclspecs.html