Comparison and Performance Evaluation of Mobile Agent Platforms

Raquel Trillo, Sergio Ilarri and Eduardo Mena
IIS Department, University of Zaragoza (Spain)
Email: {raqueltl, silarri, emena}@unizar.es

***Abstract*— Mobile agents are considered a very interesting technology to develop applications for mobile, pervasive, and distributed computing. Thus, they present a combination of unique features, such as their autonomy and capability to move to remote computers to process data there and save remote communications.**

Many mobile agent platforms have been developed since the late nineties. While some of them have been abandoned, others continue releasing new versions that fix bugs detected or offer new interesting features. Moreover, other new platforms have appeared in the last few years. So, a common problem when one wants to benefit from mobile agent technology to develop distributed applications is the decision about which platform to use. In this paper, we provide an up-to-date evaluation of existing mobile agent platforms. We compare them qualitatively and evaluate their performance in a variety of settings with an extensive set of experiments.

I. Introduction

Mobile agents [1] have been proposed as a useful technology for building distributed applications. They present interesting advantages, such as autonomy, flexibility, and effective usage of network bandwidth [2]. Due to these features, they have also been considered as an enabling technology for mobile, wireless and pervasive computing [3]. However, the initial hype concerning mobile agent technology was followed by a much more moderate period. Nowadays, mobile agents are still an important focus of attention [4], but some doubts arise, for example, about their applicability and performance [5].

Undoubtedly, there is still much work to do to allow the developers of distributed applications to take full advantage of mobile agents. Indeed, despite the high number of mobile agent platforms developed along time, new platforms have still been developed in the last few years. Similarly, new versions of popular platforms have also been released. So, choosing the right or most suitable platform is a challenge for the developer.

In this paper, we compare the features of existing mobile agent platforms. We also perform an extensive experimental evaluation in order to analyze the performance and reliability of such platforms in different scenarios. In the following, we present in Section II the motivation of this work. In Section III, we compare qualitatively some popular mobile agent platforms. In Section IV, we perform several experiments to compare the performance of the most popular platforms under different conditions. Finally, we summarize our conclusions in Section V.

II. Motivation

In this section, we highlight the importance of this work and justify the choice of the mobile agent platforms that we have considered for comparison: *Aglets*, *Voyager*, *Grasshopper*, *Tryllian*, *JADE*, *Tracy*, and *SPRINGS*.

According to the survey in [6], Aglets, Voyager and Grasshopper were among the four best mobile agent platforms; their ranking was: 1) Grasshopper, 2) Jumping Beans, 3) Aglets and 4) Voyager. Similarly, a hierarchical framework for benchmarking mobile agent platforms at different levels of detail was proposed in [7], and the authors considered representative for their evaluation the platforms Aglets, Voyager, Grasshopper and Concordia. The previous studies were published several years ago (in 2001). Since then, new platforms and improved versions have appeared and others are today not so relevant. Thus, we also consider relevant today the platforms Tryllian, JADE, Tracy and SPRINGS. The last version of Tryllian has been released recently as open source, JADE is a very popular platform for the development of agent-based systems, the development of Tracy has led to the publication of the most recent book on mobile agents (since some years ago), and SPRINGS has been developed very recently and offers a promising scalability. In this work we do not to evaluate Concordia or Jumping Beans due to the following reasons: Concordia is not available since 1st of December 2003 and, as far as we know, Jumping Beans has always been a commercial product not available for free. Although there are more platforms that have contributed to different aspects of current mobile agent technology, we consider that the platforms chosen for evaluation are a good representative of the current state in the field.

Other papers that include some comparisons among different platforms are: [8] (1999, they compare qualitatively Aglets, Concordia, Grasshopper, Odyssey and Voyager, and also show some performance results), [9] (1999, they present the conclusions of some experimental results comparing platforms such as Aglets, Concordia, Voyager, Odyssey, Jumping Beans, Swarm and Grasshopper), [10] (2001, they compare experimentally Aglets and Concordia), [11] (2001, they compare qualitatively mobile agent platforms such as Telescript, Aglets, D'Agents and AgentSpace), [12] (2004, they evaluate experimentally Tryllian and Aglets, but not their support for mobile agents), [13] (2004, they compare qualitatively agent platforms such as Tryllian, Grasshopper and JADE, although

not with mobile agents), [3] (2004, they present experiments with Aglets and Voyager in the context of wireless access to distributed databases), and [14] (2005, they compare qualitatively agent toolkits such as JADE and Aglets, and measure some performance results, but they do not consider mobile agents).

In general, the different works where some mobile agent platforms are compared are partially outdated today, as new platforms and versions have appeared since the publication of such works. For example, as far as we know no other survey evaluates JADE and Tryllian in the context of mobile agents. As developers of distributed systems based on mobile agents, we think that a work with an updated comparison should be beneficial to the community.

III. Qualitative Comparison

In this section, we review the different mobile agent platforms considered.

A. Aglets

Aglets (http://aglets.sourceforge.net/), initially developed by *IBM* in 1997 and maintained by the open source community since 2001 (last version: 2.02, February 2002), is probably the most popular mobile agent platform developed so far. One of its strongest points is that it follows the *MASIF* specification [1]. Aglets is built around a *single-thread* model for agents and a communication infrastructure based on *message passing*. Both *synchronous and asynchronous* messages are supported. Agents in Aglets use *proxies* (similar to the *stubs* in *RMI*) as a convenient abstraction to refer to remote agents (e.g., to send them messages).

While Aglets has contributed significantly to the field of mobile agents, the level of activity associated to this platform nowadays seems to be quite low. An important disadvantage of the platform is that the proxies it provides are not dynamic proxies (i.e., they cannot be used after the agent they point to moves to another *place*): therefore, the programmer must obtain himself/herself an updated proxy, if needed, before using it. As every agent is assigned a single thread, the programmer must avoid the execution of long-running tasks: otherwise, that would prevent agent events (such as incoming messages) from being considered. This single-thread model, for example, could lead to deadlocks if two agents send each other a synchronous message at the same time (each of the agents is unable to process the incoming message until the other agent has received its message). The platform does not support remote calls to agents or assigning them user-friendly identifiers. Finally, it is not possible to specify *callback* methods for travel operations (a predefined method is always executed on arrival to the target *place*).

B. Voyager

Voyager (http://www.recursionsw.com/), developed initially by *ObjectSpace* in 1997 and currently by *Recursion Software* (last version: *Voyager Edge* 6.0.1, August 2006), is a distributed computing middleware focused towards simplifying the management of remote communications of traditional CORBA and RMI protocols. It offers facilities such as the dynamic generation of CORBA proxies, mobile code, and mobile agents. Agents communicate via traditional remote method invocations using proxies. According to [15], Voyager provides location transparency through forwarding chains of proxies. Voyager is an interesting platform, with many functionalities, which eases the development of distributed systems.

A key disadvantage of Voyager is that it is a commercial product not available for free, which could prevent many researchers from using it in favor of other alternatives available. The forwarding chain mechanism used to track mobile agents could also be inefficient (the whole chain must be traversed in order to locate an agent) and weak (a single broken link makes the agent unreachable).

C. Grasshopper

Grasshopper [16] was developed by *IKV++* in 1999 (last version: 2.2.4, January 2003), then became part of the commercial *Enago Mobile*, and today its development has probably been abandoned. It is an easy-to-use platform for mobile agents, compliant with the standards *MASIF* [1] and *FIPA* (http://www.fipa.org/). A Grasshopper system can be composed of different *regions*. It provides agent developers with interesting features, including a graphical user interface to manage agents, agencies, and regions. By defining regions, the developer can benefit from dynamic proxies.

The main disadvantage of Grasshopper is that it is not available anymore and new versions will not appear in the future. The *region server* could become a bottleneck, as it must update every proxy right before using it. A disconcerting feature of Grasshopper (stated in the manual) is that a call to an agent that is moving can end up executing on the *copy* of the agent at origin (which will be removed once the agent arrives at its destination). Finally, as in Aglets, the same predefined method is always executed after an agent's trip.

D. Tryllian

Tryllian (http://www.tryllian.org), developed by the homonym company in 2001 (last version: 3.2.0, released as *open source* in November 2005), is based on a *sensing-reasoning-action* mechanism. It allows programmers to define a reactive (based on incoming messages) and proactive (based on *heartbeats*) behavior of agents. Tryllian proposes a *task-based* programming model and communication among agents is achieved through *message passing* and in accordance with the *FIPA* standard. It also provides a *persistency* service.

The main disadvantage of Tryllian is that it does not offer location transparency (the current location of the target agent of a message must be known in advance). In addition, its task-based and asynchronous model could be difficult to use, due to its differences with the classical procedural programming. The use of a single thread per agent could be inefficient and a limitation for the programmer. Tryllian provides a large set of configuration options, which could be overwhelming. Finally,

it does not offer facilities for synchronous communication or conventional method invocation.

E. JADE

JADE (http://jade.tilab.com/), developed by *Telecom Italia Lab* since July 1998, was released as *open source* in February 2000 (last version: JADE 3.4.1, November 2006). It is a very popular *FIPA-compliant* agent platform. An agent is composed of different concurrent (and non-preemptive) *behaviors*, which can be added dynamically. Among the benefits, we could indicate that there is a wide variety of tools provided (e.g., for remote management and monitoring of agents, and to track interchanged messages) and it can be integrated with different software such as *Jess*[1] (a rule engine which allows JADE agents to "reason" using knowledge provided in the form of declarative rules). Finally, it is also worth mentioning its support for the development of ontologies to represent the knowledge of agents.

Probably, the main disadvantage is that mobility is not a key element in JADE. Thus, it focuses on other functionalities relevant to the development of multiagent systems. The JADE built-in *Agent Mobility Service* supports mobility among containers within the same *JADE platform* (similar to the idea of *region* in Grasshopper and SPRINGS), and researchers at the *Autonomous University of Barcelona* provide an *Inter-Platform Mobility Service* (https://tao.uab.cat/ipmp/). Proxies do not exist; instead, an agent searches the current location of its target by querying the *AMS* (*Agent Management System*), according to the FIPA specifications.

F. Tracy

Tracy (http://www.mobile-agents.org/), developed at the *University of Jena* in Germany (last version: 1.0.1-52, April 2005), has a *plugin-oriented* architecture. Plugins are software components that can be added dynamically to a running *agency* (the context where agents execute), if required, in order to provide high-level services (e.g., inter-agent communication, migration, security, etc.). Therefore, Tracy agencies are lightweight (a service is not loaded if it is not required) and extensible (new plugins can be developed and added to support new services). The platform offers several *migration strategies* for agents.

A key disadvantage of Tracy is that it does not support remote communications between agents: an agent must travel to the agency where another agent is running in order to communicate with it. The platform was probably developed mainly as a test environment where different migration and class loading strategies could be evaluated, and in that aspect the authors have performed a meritorious and interesting research work. Compared with other platforms (such as Voyager, Grasshopper or JADE), there is no much documentation available, which makes it difficult to use (even with the associated book [17], which covers aspects of mobile agents in general). Despite the novelty of the platform, its level of activity is very low (e.g., the mailing list is hardly used and the *blog* has been inactive since its creation).

[1]http://herzberg.ca.sandia.gov/jess/

G. SPRINGS

SPRINGS (http://sid.cps.unizar.es/SPRINGS/), developed recently by the *Distributed Information Systems Group* at the *University of Zaragoza* in Spain (in continuous development and available under request), focuses on scalability and reliability in scenarios with a moderate and high number of mobile agents. Its development has been inspired by the features of other popular platforms, such as Voyager and Grasshopper. Similarly to other platforms (e.g., Grasshopper and JADE), it proposes a hierarchical architecture based on *regions*. It provides full *location transparency*: 1) for movements (the programmer does not need to specify network addresses but just the name of the destination (the mapping is dynamic and does not rely on configuration files); and 2) for calls, through the use of *dynamic proxies*. Moreover, it attempts to minimize the *livelock* problem that can arise when agents move quickly. The experimental results in [18] show that SPRINGS outperforms other platforms in scenarios with a high number of very dynamic mobile agents.

The main disadvantage of SPRINGS is perhaps that it does not support agent communication using the standard FIPA. Also, it does not provide sophisticated security mechanisms. Despite it is easy to use, it does not offer any graphical tool to the user. Finally, as it is a new platform, there is little documentation available about it.

H. Summary of Main Features of Mobile Agent Platforms

A summary of some features of the platforms compared is shown in Table I. In the table, we evaluate the following features for each platform: 1) its philosophy for programming mobile agents; 2) the main components in the platform; 3) whether it supports the concept of proxy or not; 4) whether the proxies continue being valid when agents move; 5) whether it supports synchronous communications and/or 6) asynchronous communications; 7) whether the agents can communicate by passing messages among themselves[2]; 8) whether remote calls (RMI-like) are supported; 9) whether an agent can specify a *callback* method to execute on arrival at its destination (or, on the contrary, the same predefined method is always executed); 10) whether it is possible to indicate the target of a message/call by specifying a user-friendly name; 11) whether it allows to indicate the target of a movement by specifying a user-friendly name; 12) whether it is freely available for download; 13) whether it is shipped with graphical tools; 14) the level of activity associated with the platform (new versions released, updates to its web page, mailing lists, etc.); 15) whether it offers security mechanisms; and, finally, 16) some other features that also characterize the platform. Notice that some of the features shown in the table are not inherently positive or negative. For example, some programmers will find a traditional procedural approach more convenient to build

[2]We do not use the term *message* here in the object-oriented sense (as Voyager does); for that, we use the term *remote call*.

TABLE I

QUALITATIVE COMPARISON AMONG MOBILE AGENT PLATFORMS

Feature	Aglets	Voyager	Grasshopper	Tryllian	JADE	Tracy	SPRINGS
1) Model	Events	Procedural	Procedural	Tasks	Behaviors	Procedural	Procedural
2) Elements	-Contexts -Agents (aglets) -Tahiti	-Servers -Agents	-Places -Regions -Agents	-AFC -ADK -Habitats -Agents	-Containers -Main container -Platforms -Agents -DF, AMS, MTS	-Agencies -Agents - Plugins	-Places -Regions (RNSs) -Agents
3) Proxies	Yes	Yes	Yes	No	No	No	Yes
4) Dynamic proxies	No	Yes (forwarding)	Yes (via region server)	No	No	No	Yes (location updates)
5) Synchronous communications	Yes (deadlock)	Yes	Yes	Yes (SendAndReceiveTask)	No	No	Yes
6) Asynchronous communications	Yes	Yes	Yes	Yes	Yes	Yes (within same agency)	Yes
7) Messages	Yes	No	Yes (FIPA)	Yes (FIPA)	Yes (FIPA)	Yes (within same agency)	Yes
8) Remote calls	No	Yes	Yes	No	No	No	Yes
9) Callbacks after movements	No	Yes	No	No	No	No	Yes
10) Call/messages by name	No	No	No	No	Yes (AID)	Yes (within same agency)	Yes
11) Movements by name	No	No	No	Yes (via configuration files)	Yes (via AMS)	Yes (via TNS plugin)	Yes (dynamically)
12) Available for download	IBM Public License	Not free (evaluation version)	Not anymore	LGPL	LGPL	Yes (binaries)	Yes (binaries)
13) GUI Tools	Some	No	Yes	Yes	Yes	No	No
14) Level of activity	Very low	High	None	Medium	Very high	Very low	High
15) Security Mechanism	Basic	Yes (security managers, etc.)	Basic	Yes (signed agents, etc.)	Yes (JAAS, etc.)	Yes (Bouncy Castle)	Basic (policies)
16) Some other features	-ATP -Itinerary	-Multicast -Publish/subscribe -Dynamic aggregation	-MASIF -FIPA -Multicast	-FIPA -DNA	-FIPA -Jess, JADEX, etc. -Ontology support	-Lightweight, extensible -Kalong mobility model -Several migration strategies	-No livelock -Schedule -Reliable, efficient

their agents, while others will prefer an event-based or a task-based approach. The same can be said regarding whether a communication model based on message passing or based on remote calls is supported. Similarly, some agent developers (e.g., those working in fields related to Artificial Intelligence) will consider the compliance with FIPA a core feature, while others will just consider important the support of good communication mechanisms to build distributed applications.

IV. EXPERIMENTAL EVALUATION

In this section, we perform two types of experiments. First, we consider a scenario with a number of agents calling among themselves and moving from one computer to another; in such a scenario, we can easily compare the platforms under different stress levels. Then, with the platforms that obtained the best results in the first set of tests, we perform an experiment where the agents carry out a cooperative task. Each test is performed 10 times and average values are reported. We benefit from the functionalities of *regions* whenever they are available in the platform evaluated (i.e., in SPRINGS and Grasshopper).

A. Experimental Comparison with a Sample Test

The scenario that we consider in this experiment is the following. We deploy 5 *places* on 5 different computers (Pentium IV 3.6 GHz with Linux and 2 GB RAM) and we launch a certain number of agents which will continuously move from its current *place* to another *place* at random. Unless it is specified otherwise in the following, every agent will call a certain agent after moving randomly to another *place*; for communication, each agent is randomly assigned a *peer agent* at the beginning of the test. So, an agent will be constantly either moving or calling its peer. Movements and calls will be retried, if they fail, as many times as necessary. Notice that, with a high number of agents, this is a challenging scenario that could overload a mobile agent platform. For performance evaluation, every agent keeps a *log* of its operations (movements and calls) and the delays experienced in such operations. After an agent performs 50 iterations (i.e., 50 movements and calls), it is considered that the agent has finished its tasks and the recording of new data in the agent's log finishes. However, to keep the overloading, the agent will continue iterating until all the agents finish their tasks.

In the scenario described, we compare the following mobile agent platforms: Aglets, Voyager, Grasshopper, Tryllian, JADE and SPRINGS. Tracy is not considered because it does not support remote communications (see Section III-F). We benefit from dynamic proxies whenever they are available in the evaluated platform (i.e., in Voyager, Grasshopper and SPRINGS). Concerning Aglets, proxies to agents are not automatically updated when they point to roaming agents; therefore, an *aglet* in our test searches for its peer in the existing *places* when the proxy becomes invalid due to a movement of the peer. In JADE, a query about the current location of the target agent needs to be issued to the *AMS* (see Section III-E). Finally, Tryllian requires the specification of the target habitat when sending a message, so the target agent is searched first every time a call occurs. In the following, we describe three tests that we have carried out to compare the platforms.

In our first test, we evaluate how the platforms perform in the scenario described with 100 agents. In Figure 1, we show how each test evolves over time, in terms of the total number of iterations performed so far and the number of agents that have finished their 50 iterations. In this test, we can observe that:

- Only SPRINGS is able to finish the test (in 40 seconds).
- The second best platform is Voyager, where 58 out of the 100 agents are able to finish. At time instant 2:05, agents in Voyager have performed 3576 iterations in total and they are not able to progress anymore.
- The third best platform is Grasshopper, with 53 agents finished and 3334 iterations performed. From time instant

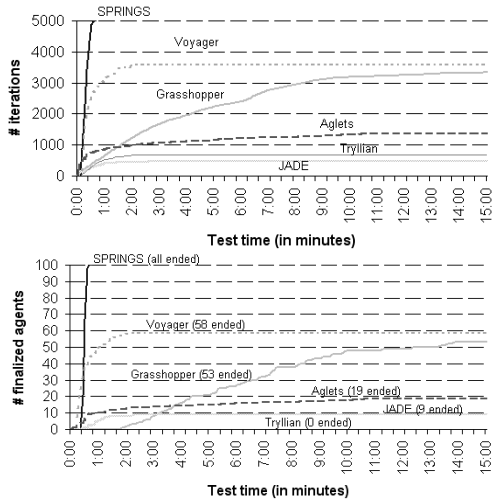

Fig. 1. Comparison with 100 agents: number of iterations (top) and number of agents that finish the test (bottom) along time

14:00 no more agents are able to finish the test. We cannot ensure that the test would not be able to finish if given much more time, but the rate of iterations performed at the end of the test is extremely low.

- Next in performance is Aglets, with 19 agents finished and 1357 iterations performed.
- JADE (460 iterations, 9 agents finished, unable to progress since time instant 3:00) and Tryllian (66 iterations, 0 agents finished, unable to progress since time instant 2:00) exhibit the worst performance results in this test.

In our second experiment, we slightly modify the conditions of the previous test by removing the calls among the agents (i.e., we assume that the agents do not cooperate with their peers). First, and in order to determine if the problems observed previously (agents that cannot finish their tasks) were due to a "high" number of agents, we consider a scenario with only 2 agents. The performance results regarding the number of iterations along time are shown in Figure 2. In this case, all the platforms are able to finish the test. Aglets, Voyager, JADE and SPRINGS offer the best performance and the differences are not significant (notice the small scale of the X-axis). We can highlight that Voyager and SPRINGS are also in this test among the best platforms, and that Tryllian still shows a comparatively low performance. JADE, which did not perform well in the previous test, is among the best platforms according to this experiment.

In our last experiment, we consider again a scenario with 100 agents but, as in the previous test, with no calls among them. In Figure 3, we can see that SPRINGS, Tryllian, Aglets and JADE are able to finish the test. Voyager (with 87 finished agents) was performing again very well until time instant 1:05, when it stopped progressing. Notice also the good performance of Tryllian in this test. We must also indicate that, as in our first test, the rate of iterations performed at the end of the test with Grasshopper is very low (2950 iterations are

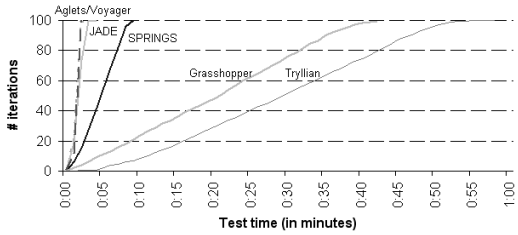

Fig. 2. Comparison with 2 agents that only move (no calls)

performed in total). This experiment reveals that the reliability and performance problems of some platforms arise only when agents move and communicate among themselves (as seen in Figure 1): agents that only move usually lead to less problems (as Figures 2 and 3 show).

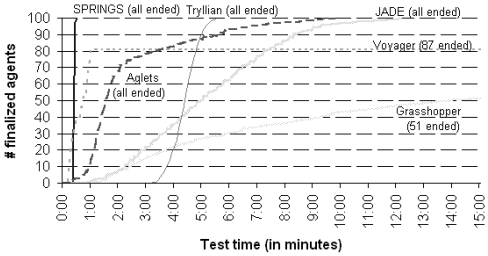

Fig. 3. Comparison with 100 agents that only move (no calls)

According to these experiments, SPRINGS and Voyager could be considered the platforms that offer the best performance. Although Voyager does not finish some tests, it could be due to a problem which we expect to be solved in future releases of the platform. Thus, in our tests with Voyager we only retry a movement if a mobility exception is thrown. Although this should be the only indicator of a failed movement, we have detected that sometimes other type of exception may be thrown, which is some cases prevents and *in others do not* the success of the trip; we have decided not to retry the trip in case a non-mobility exception occurs because, if the agent has actually arrived successfully in the target *place*, we would end up with two copies of the same agent roaming the network.

B. Performance Comparison Executing a Parallel Algorithm

In this test we compare SPRINGS and Voyager, the platforms that offered the best performance in our previous experiments, implementing a parallel algorithm to multiply two real matrices in a distributed environment. While there exist several algorithms for parallel matrix multiplication, we simply choose one that is simple to implement (although not optimal). We consider two square *n-by-n* matrices A and B, and we want to compute the product of both matrices (matrix C). We create n agents to perform the multiplication. Initially, each agent i holds the row i of matrix A and the column i of matrix B. This allows the agent to compute $C(i, i)$. After that, the agent sends the column i of matrix B to agent $i + 1$ and receives the column $i - 1$ of matrix B from agent $i - 1$. With

row i of matrix A and column $i-1$ of matrix B, it computes $C(i, i-1)$. The process repeats similarly until every agent i has computed entirely the row i of C.

We take advantage of the mobility of the agents in the following way. Every agent monitors periodically the CPU overload of its current computer, and decides to move to another computer when it exceeds a certain threshold. In this way, we can expect that the agents will execute, most of the time, on computers that are not highly overloaded. To make sure that the CPU overload varies frequently enough, we have performed these experiments in a *computer cluster* that several research groups at our university use to carry out CPU-intensive computations; the nodes in the cluster are Pentium IV 2.8 GHz with Linux and 2 GB RAM.

In Figure 4, we show the percentage of the resulting matrix computed along time with SPRINGS and Voyager, for a scenario with 500 agents (i.e., two 500x500 input matrices). We can see how SPRINGS also outperforms Voyager in this scenario[3]; notice that we are interested in the comparison, not in the total time needed to perform the computation (which depends on the overload of the computers and the fine-tuning of the algorithm). For example, we have not analyzed the threshold value that leads to the best performance, as our goal is just to make the agents move.

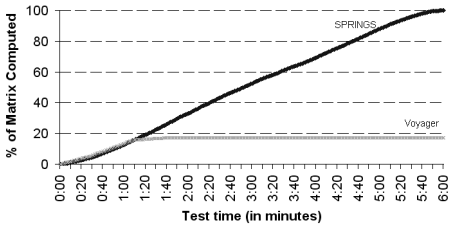

Fig. 4. Comparison of SPRINGS and Voyager multiplying matrices

V. Conclusions and Future Work

In this paper, we have presented a comparison and experimental evaluation of current mobile agent platforms. The qualitative comparison intends to provide an overview of the different alternatives. The experiments have been performed considering different scenarios in order to evaluate the platforms under varied conditions. We believe that all the platforms compared in the paper are interesting, and that our comparison could help a developer to decide which platform suites his/her needs better.

With this work, we expect to contribute to cover the need of an updated review of mobile agent platforms and also to encourage future work in the field.

Acknowledgment

This work was supported by the CICYT project TIN2004-07999-C02-02. We would also like to thank the support of the Spanish Excellence Network of Agents (CICYCT TIN2005-25869-E). Last but not least, we are grateful to Raúl Marzo for helping us with some of the experiments.

[3]We observed a strange behavior in Voyager when an agent was communicated a new column while it was traveling; we solved the problem refusing remote calls in those situations.

References

[1] D. Milojicic, F. Douglis, and R. Wheeler, *Mobility: processes, computers, and agents*. Addison-Wesley Professional, April 1999.

[2] D. B. Lange and M. Oshima, "Seven good reasons for mobile agents," *Communications of the ACM*, vol. 42, no. 3, pp. 88–89, March 1999.

[3] C. Spyrou, G. Samaras, E. Pitoura, and P. Evripidou, "Mobile agents for wireless computing: the convergence of wireless computational models with mobile agent technologies," *Mobile Networks and Applications (MONET)*, vol. 9, no. 5, pp. 517–528, October 2004.

[4] G. Samaras, "Mobile agents: What about them? did they deliver what they promised? are they here to stay?" in *Mobile Data Management (MDM'04), Berkeley, California, USA*, January 2004, pp. 294–295.

[5] G. Vigna, "Mobile agents: Ten reasons for failure," in *Mobile Data Management (MDM'04), Berkeley, California, USA*, January 2004, pp. 298–299.

[6] K. Ludwig, A. Josef, W. E. Edgar, S. Wolfgang, and G. Franz, "Using mobile agents in real world: A survey and evaluation of agent platforms," in *Second International Workshop on Infrastructure for Agents, Multi-Agent Systems, and Scalable Multi-Agent Systems, at the 5th International Conference on Autonomous Agents, Montreal, Canada*. ACM Press, May 2001.

[7] M. Dikaiakos and G. Samaras, "A performance analysis framework for mobile-agent systems," in *Second International Workshop on Infrastructure for Agents, Multi-Agent Systems, and Scalable Multi-Agent Systems at the 4th International Conference on Autonomous Agents, Lecture Notes in Computer Science, Barcelona, Spain, June 3-7, 2000, Revised Papers*, vol. 1887. Springer-Verlag, June 2001, pp. 180–187.

[8] M. K. Perdikeas, F. G. Chatzipapadopoulos, I. S. Venieris, and G. Marino, "Mobile agent standards and available platforms," *Computer Networks*, vol. 31, no. 19, pp. 1999–2016, August 1999.

[9] L. Silva, G. Soares, P. Martins, V. Batista, and L. Santos, "The performance of mobile agent platforms," in *First International Symposium on Agent Systems and Applications / Third International Symposium on Mobile Agents (ASA/MA'99), Palm Springs, California, USA*. IEEE Computer Society, October 1999, pp. 270–271.

[10] M. D. Dikaiakos and G. Samaras, "Performance evaluation of mobile agents: Issues and approaches, lecture notes in computer science," in *Performance Engineering*, vol. 2047. Springer, May 2001, pp. 148–166.

[11] A. R. Silva, A. Romäo, D. Deugo, and M. M. Silva, "Towards a reference model for surveying mobile agent systems," *Autonomous Agents and Multi-Agent Systems*, vol. 4, no. 3, pp. 187–231, September 2001.

[12] K. Burbeck, D. Garpe, and S. Nadjm-Tehrani, "Scale-up and performance studies of three agent platforms," in *International Conference on Performance, Computing, and Communications (IPCCC'04), Phoenix, Arizona, USA*. IEEE Computer Society, April 2004, pp. 857– 863.

[13] R. Leszczyna, "Evaluation of agent platforms," European Commission, Joint Research Centre, Institute for the Protection and Security of the Citizen, Ispra, Italy, Tech. Rep., June 2004.

[14] E. Shakshuki, "A methodology for evaluating agent toolkits," in *International Conference on Information Technology: Coding and Computing (ITCC'05), Las Vegas, Nevada, USA*, vol. 1. IEEE Computer Society, April 2005, pp. 391–396.

[15] L. Moreau, "A fault-tolerant directory service for mobile agents based on forwarding pointers," in *ACM symposium on Applied computing, Madrid, Spain*. ACM Press, March 2002, pp. 93–100.

[16] C. Bäumer and T. Magedanz, "Grasshopper - a mobile agent platform for active telecommunication," in *Third International Workshop Intelligent Agents for Telecommunication Applications (IATA'99), Stockholm, Sweden*, August 1999, pp. 19–32.

[17] P. Braun and W. R. Rossak, *Mobile Agents-Basic Concept, Mobility Models, and the Tracy Toolkit*. Morgan Kaufmann Publishers, December 2005.

[18] S. Ilarri, R. Trillo, and E. Mena, "SPRINGS: A scalable platform for highly mobile agents in distributed computing environments," in *Fourth International WoWMoM 2006 Workshop on Mobile Distributed Computing (MDC'06), Niagara Falls/Buffalo, New York, USA*. IEEE Computer Society, June 2006.