ELSEVIER

Computer Communications 23 (2000) 769–778

computer communications

www.elsevier.com/locate/comcom

Comparing the performance of mobile agent systems: a study of benchmarking

L.M. Silva*, G. Soares, P. Martins, V. Batista, L. Santos

Departamento Engenharia Informática, Universidade de Coimbra, Polo II, Vila Franca, 3030—Coimbra, Portugal

Abstract

In the past few years there has been an enthusiastic interest in mobile agent technology and several platforms have been developed. Some of them have only been used for research purposes while others have been deployed as commercial products. The community is now looking for applications where these platforms can be effectively used. Some comparisons about the functionality of some mobile agent systems have been presented in the literature. However, to the best of our knowledge, there has been no reported study that compares the real performance of the platforms.

In this paper, we present the first results of an experimental study that compares the performance of eight Java-based Mobile Agent systems: Aglets, Concordia, Voyager, Odyssey, Jumping Beans, Grasshopper, Swarm and James. This study presents some insights about the performance and the robustness of each platform. © 2000 Elsevier Science B.V. All rights reserved.

Keywords: Mobile agents; Performance; Benchmarking

1. Introduction

Mobile agents is an emerging technology that is gaining momentum in the field of distributed computing. The use of mobile agents can bring some interesting advantages when compared with traditional client/server solutions [1]: it can reduce the traffic in the network, it can provide more scalability, it allows the use of disconnected computing and it provides more flexibility in the development and maintenance of the applications.

In the latest years, several commercial implementations of mobile agent systems have been presented in the market, including Aglets from IBM, Concordia from Mitsubishi, Voyager from ObjectSpace, Odyssey from General Magic, Jumping Beans from AdAstra and Grasshopper from IKV.

We have also been developing a Java-based platform, called James, in a cooperating project between the University of Coimbra and Siemens (Eureka Project Σ!1921) [2]. This platform is mainly oriented for telecommunications and network management. In this target field the performance of the applications plays a very important role, together with fault-tolerance, resource-control and security. The platform has been developed from scratch and, together with our industrial partners (Siemens), we are now working on the implementation of three agent-based applications: one in the area of performance management, other for system integration in mobile networks and a third one for data network management.

However, some assertive questions have been made by our project partners: what is the performance of the existing mobile agent platforms? What is their level of dependability? How robust will be the applications that use this technology? What are the benefits of the mechanisms that have been introduced in the James platform? How this platform compares with the other ones?

To answer some of these questions we have conducted an experimental study of benchmarking. In this paper, we will present some results that compare the performance of eight Java-based mobile agent systems and we will take some conclusions about its run-time behaviour. To the best of our knowledge this is the first study that has been reported about the performance of mobile agent systems. Although performance is not the main issue to decide about a particular platform it has its relevance when companies decide to deploy some production codes that make use of this technology.

The rest of the paper is organized as follows. Section 2 presents a brief overview of the platforms that have been selected for our study. Section 3 describes the methodology of our benchmarking study, while Section 4 presents some of the most relevant results. Section 5 gives some conclusions

* Corresponding author. Tel: + 351-39-790090; fax: + 351-39-701266.
E-mail addresses: luis@dei.uc.pt (L.M. Silva), fsoares@dei.uc.pt (G. Soares), pmartins@dei.uc.pt (P. Martins), vbatista@dei.uc.pt (V. Batista), lsantos@dei.uc.pt (L. Santos).

0140-3664/00/$ - see front matter © 2000 Elsevier Science B.V. All rights reserved.
PII: S0140-3664(99)00237-6

about the obtained results. Section 6 presents some final remarks.

2. Brief description of the platforms

In our study we have selected eight different mobile agent platforms. All of them were written in Java. There are several reports in the literature about the functionality of some of these platforms. In Ref. [3] is presented an extensive comparison between the features of Voyager, Odyssey, Aglets and Concordia. Kiniry and Zimmerman [4] have presented a direct comparison between Odyssey, Aglets and Voyager. In Ref. [5] is presented another comparison of agent system features that includes Aglets, Voyager, Odyssey and Kafka. In Ref. [6] is presented a comprehensive review about three platforms: Aglets, Voyager and Odyssey. Another extensive evaluation has been presented in Ref. [7]. This one included more platforms: D'Agents, April, Aglets, Grasshopper, Odyssey and Voyager.

These reports only focus in the list of features of each platform and present some conclusions about the overall functionality of the platforms. However, no performance results have been reported so far. Next, we will present a short description about each platform.

2.1. Aglets SDK

The Aglets Software Developer Kit (ASDK) was developed at the IBM Research Laboratory in Japan. The first version was released in 1996. ASDK requires the JDK 1.1 or higher to be installed. The migration of Aglets is based on a proprietary Agent Transfer Protocol (ATP). The ASDK run-time consists of the Aglets server and a visual agent manager, called Tahiti. There is an additional module of software, called Fiji, that allows the installation of an Aglets server on a HTTP-browser. The ASDK provides a modular structure and an easy-to-use API for the programming of Aglets. This platform has extensive support for security and agent communication and provides an excellent package of documentation. In our experiments we have used ASDK 1.0.3. More details about Aglets SDK can be obtained in: http://www.trl.ibm.co.jp/aglets/

2.2. Concordia

Concordia has been developed by Mitsubishi Electric. It requires the JDK 1.1 or higher to be installed. This platform provides a rich set of features, like support for security, reliable transmission of agents, access to legacy applications, inter-agent communication, support for disconnected computing, remote administration and agent debugging. This system also provides good documentation. In our experiments we have used version 1.1.2 of Concordia. More details about Concordia can be obtained in: http://www.meitca.com/HSL/Projects/Concordia/

2.3. Voyager

Voyager is an object request broker with support for mobile objects and autonomous agents. It was developed by ObjectSpace. It requires the JDK 1.1 or higher to be installed. The agent transport and communication is based on a proprietary ORB on top of TCP/IP. Voyager has a comprehensive set of features, including support for agent communication and agent security. Voyager provides support for Corba and RMI. Due to its dynamic proxy generation these technologies can be used without the need for stub generators. Thereby, Voyager objects can be used as Corba objects. In our experiments we have used version 3.0 beta of Voyager. More details about Voyager can be obtained in: http://www.objectspace.com/products/voyager/

2.4. Odyssey

Odyssey is a Java-based mobile agent system from General Magic. It requires JDK1.1 or higher to execute. The platform has a transport-independent API that work with Java RMI, IIOP and DCOM. Odyssey provides the basic functionality and a small set of features. Currently, it is not clear if General Magic will continue the efforts in this platform. In our experiments we have used version 1.0 beta 2 of Odyssey. More details about Odyssey can be obtained in: http://www.genmagic.com/technology/odyssey.html

2.5. Jumping Beans

The Jumping Beans platform is commercially distributed by AdAstra, a Silicon Valley company. It requires JDK 1.1.2 or higher to execute. The main strengths of this platform include the support for security, agent management, easy integration with existing environments and a small footprint. However, this platform uses a client/server approach for the agent migration: if an agent wants to migrate between two Agencies it has to go first to the Agent Manager. This approach may represent a point of bottleneck in large-scale applications. In our experiments we have used version 1.0.4 of Jumping Beans. More details about this platform can be obtained in: http://www.JumpingBeans.com/

2.6. Grasshopper

Grasshopper is the first mobile agent platform that is MASIF-compliant. It is distributed commercially by IKV++, a company from Berlin. The entire platform is implemented in Java and requires JDK 1.1 or higher to be installed. Grasshopper supports several transport protocols by the use of an internal ORB: a proprietary protocol based on TCP/IP, Java RMI, Corba IIOP, MAF IIOP, TCP/IP with SSL and RMI with SSL. The platform support comprehensive support for security, agent communication and agent persistency. In our experiments we have used the release 1.2

of the light edition of Grasshopper. More details about this platform can be obtained in: http://www.ikv.de/products/grasshopper/

2.7. Swarm

The Swarm platform is being developed by one research centre of Siemens A.G. (ZT SW 2, Munich). It is based on version alpha 1.0 of the Mole platform, from the University of Stuttgart, Germany. Swarm is not a commercial product; it is being mainly used by the ACTS AMASE Project to provide a middleware for mobile applications in wireless networks. Swarm provides an extensive support for inter-agent communication and agent management. In our experiments we have used version 1.0 of Swarm. More details about this platform can be obtained in Ref. [8].

2.8. James

The James platform has been developed by the University of Coimbra (Portugal) in cooperation with Siemens S.A. This platform is mainly oriented for telecommunications and network management. James is not a commercial product. It requires JDK 1.1.1 or higher to execute. The main strengths of this platform include the following features: efficient code migration, support for fault-tolerance, integration with SNMP, mechanisms for resource control, flexible code upgrading, disconnected computing and agent management. The James platform has been enhanced with a set of mechanisms to optimize the migration of mobile agents, including caching techniques, code prefetching, protocol optimizations, recycling of threads and sockets. More details about these techniques can be found in Ref. [9]. In our experiments we have used version 1.0.3 of James. More details about our platform can be obtained in Ref. [2] or in the following web site: http://james.dei.uc.pt/james

In the next section we present the methodology of the benchmarking study, a description of the test environment, the test parameters and the benchmark application.

3. Conditions for the benchmarking study

3.1. Test environment

In our experiments we have used a dedicated cluster of six machines connected through a 10 Mb/s switched Ethernet. All the results were taken when the machines were fully dedicated. Every machine has a Pentium II (300 MHz) processor and 128 Mb of RAM. These machines were running Microsoft Windows NT 4.0 and all the mobile agent platforms have used JDK 1.1.6 with the JIT option.

3.2. Application benchmark

In all the experiments that will be presented in this paper we have used eight variations of the same benchmark application. The benchmark application is composed by a single agent. The agent’s mission is to run across the network through a closed itinerary, which we call a “lap”, to produce a report about the current memory usage of each machine in the network.

In all the experiments that will be presented in this paper we have used a simple benchmark application, composed by a migratory agent that roams the network to get a report about the current memory usage of each machine. This application has been written in eight different versions for all those platforms.

3.3. Test parameters

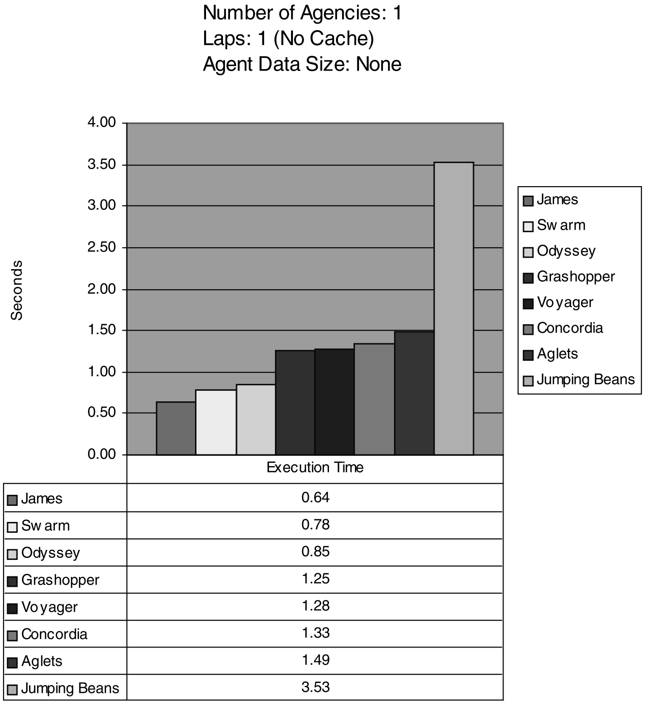
In our experiments we have changed some of the application and platform parameters, namely the number of Agencies, the number of laps performed by the agent, the agent size and the use of caching and prefetching techniques. This way, we have made tests with 1, 3 and 5 Agencies. The number of itinerary laps performed by the agent has been changed between 1, 10 and 100. The size of additional data that was carried by the mobile agent has been set to none, 100 Kb and 1 Mb. The size of the serialised object with no additional data was around 1 Kb. When the agent’s code was unknown at the destination it must be downloaded from the code server of the platform. The `jar` file with all the agent code was around 3.66 Kb.

3.4. Methodology of the benchmarking

All the platforms have been tested in the same conditions, using the same application, the same test parameters, the same agent itinerary and the same configuration. Before every set of tests all the machines of the cluster were rebooted for operating system rejuvenation. The Agencies were also restarted before each experiment, except for those cases where we wanted to measure the effect of code caching. We tried to make all the tests with the Agent Manager running uninterruptedly. Some platforms were not able to survive to some situations of stress-testing and the Agent Manager had to be restarted when failed. All the experiments were repeated at least four times and the standard deviation was within 5% of the average values.

4. Experimental results

The benchmark application was executed in all those eight platforms by changing all the test parameters (number of Agencies; number of laps; agent data size; caching mode). We have measured two main metrics: performance of the application and network traffic. For lack of space we will only present the most relevant results, corresponding to 12 experiments. During this study we were also able to evaluate some robustness level of the platforms in some situations of stress testing. We will present some insights about this issue in section of conclusions.



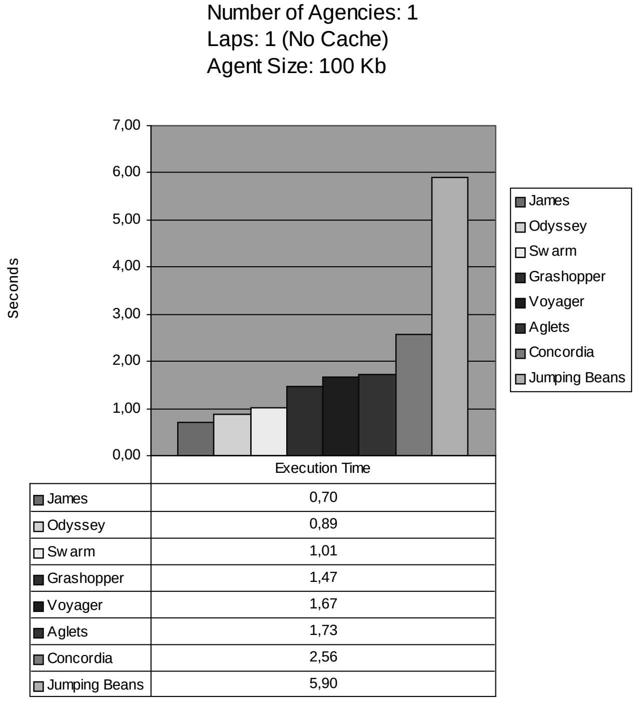
	Execution Time
James	0.64
Swarm	0.78
Odyssey	0.85
Grashopper	1.25
Voyager	1.28
Concordia	1.33
Aglets	1.49
Jumping Beans	3.53

Fig. 1. Execution time with 1 Agency, 1 lap, no data size, and no caching.

4.1. Results of performance

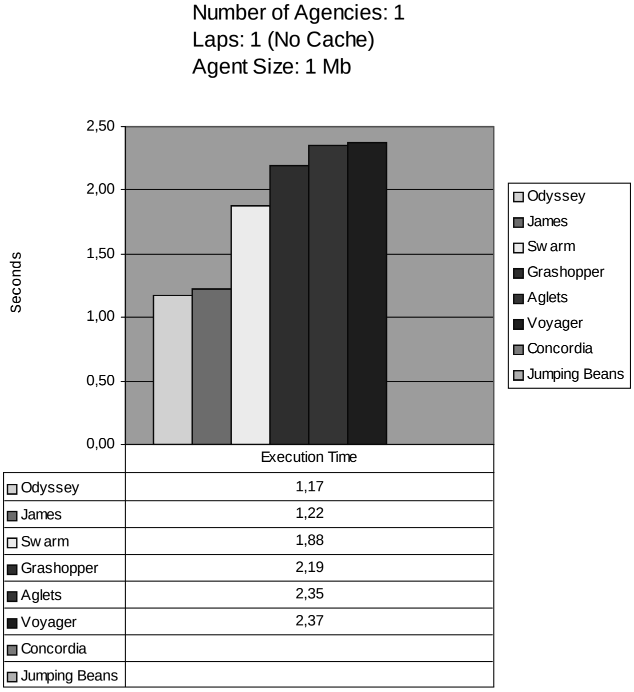
4.1.1. Experiment #1

In this experiment we have measured the execution time of all the eight platforms when using the following parameters for the system and the application: (1 Agency; 1 lap; data size = none; no caching). The results are presented in Fig. 1. James, Odyssey and Swarm present the best results for this small size agent. Jumping Beans is the slower platform: for instance, in this case it executed 5 times slower than James. When using the caching mechanism the difference was even higher: Jumping Beans was 48



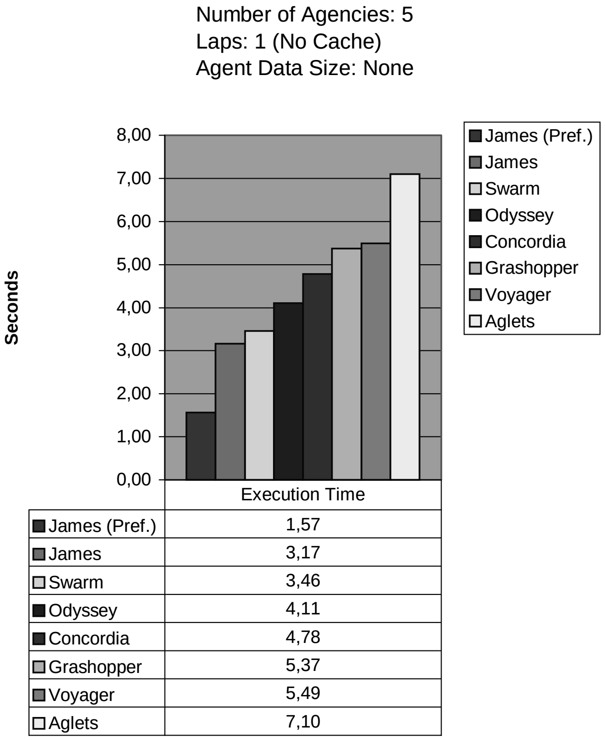
	Execution Time
James	0,70
Odyssey	0,89
Swarm	1,01
Grashopper	1,47
Voyager	1,67
Aglets	1,73
Concordia	2,56
Jumping Beans	5,90

Fig. 2. Execution time with 1 Agency, 1 lap, data size = 100 Kb, and no caching.



	Execution Time
Odyssey	1,17
James	1,22
Swarm	1,88
Grashopper	2,19
Aglets	2,35
Voyager	2,37
Concordia	
Jumping Beans	

Fig. 3. Execution time with 1 Agency, 1 lap, data size = 1 Mb, and no caching.



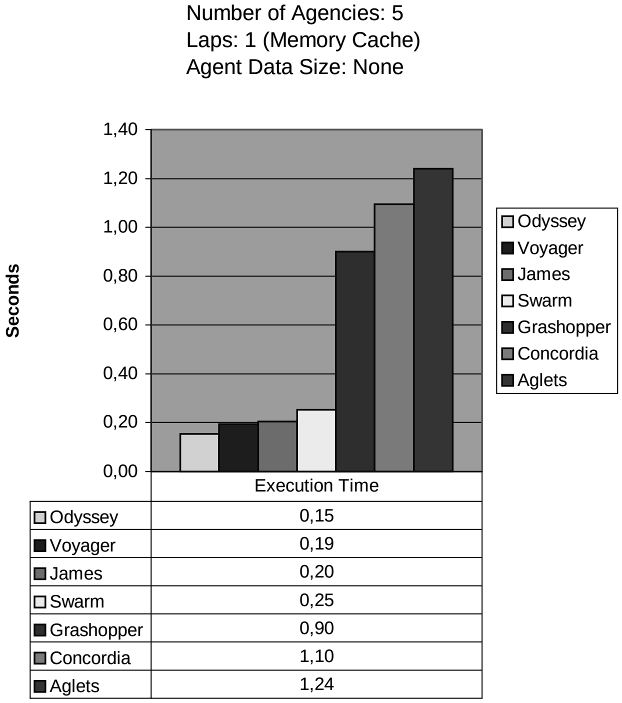
	Execution Time
James (Pref.)	1,57
James	3,17
Swarm	3,46
Odyssey	4,11
Concordia	4,78
Grashopper	5,37
Voyager	5,49
Aglets	7,10

Fig. 4. Execution time with 5 Agencies, 1 lap, no data, and no caching.

times slower than James. The results with caching are not presented in Fig. 1.

4.1.2. Experiment #2

In this experiment we have increased the size of the agent to 100 Kb. The results are presented in Fig. 2. The results were quite similar: James, Odyssey and Swarm present the best results while Jumping Beans presented the worst results, being 8 times slower than James.



	Execution Time
Odyssey	0,15
Voyager	0,19
James	0,20
Swarm	0,25
Grashopper	0,90
Concordia	1,10
Aglets	1,24

Fig. 5. Execution time with 5 Agencies, 1 lap, no data, but using caching.

4.1.3. Experiment #3

In this third experiment we have increased the size of the agent to 1 Mb. The results are presented in Fig. 3. In this case, it was interesting to observe that Odyssey and James presented the best results. However, the most important result was the fact that two of the platforms crashed in this test: Jumping Beans and Concordia.

4.1.4. Experiment #4

In this experiment we have used 5 Agencies and we only have results for seven platforms since the evaluation copy we had from Jumping Beans only executed in three platform servers. This test was done without using the caching mechanisms of the platforms and measured the impact of using the code prefetching techniques that we have implemented in the James platform. The results are presented in Fig. 4.

As can be seen in Fig. 4, the version that uses code prefetching achieved the best results: it was 2 times faster than the version of James without prefetching and it was 4 times faster than the Aglets SDK.

4.1.5. Experiment #5

In this experiment we executed the benchmark application in 5 Agencies of the dedicated network. We exploited the caching mechanisms of the platforms by running previously the application into those Agencies. The results are presented in Fig. 5. Odyssey, Voyager, James and Swarm were faster than the other three platforms.

4.1.6. Experiment #6

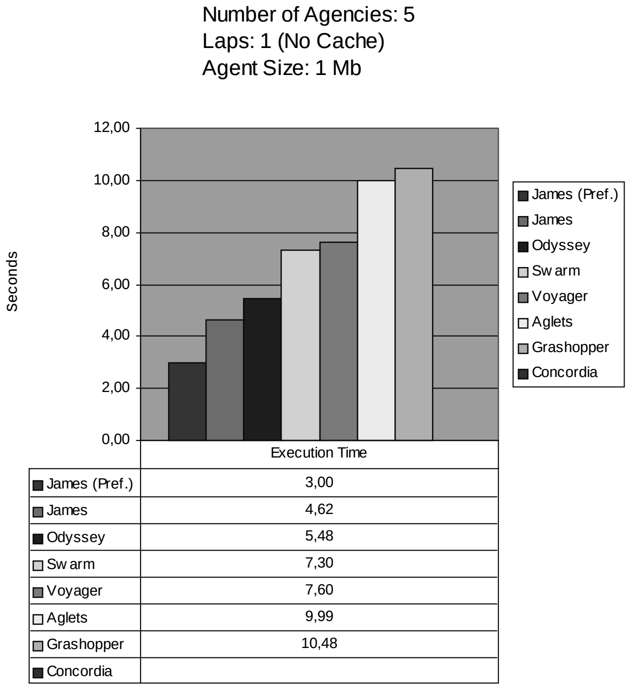
In this experiment we did not use caching and the size of the agent was increased by 1 Mb. With this agent size the Concordia system always crashed. The best results were achieved with the James platform and using the code prefetching scheme. As can be seen in Fig. 6, this version was 3 times faster than Aglets SDK and the Grasshopper platform.

4.1.7. Experiment #7

This experiment was similar to the previous, but this time we exploited the use of memory caching by the platforms. When there is caching the use of code prefetching makes no sense. Once again, the Concordia system was not able to execute the application with a mobile agent of (~) 1 Mb. The results are presented in Fig. 7 and show that Odyssey and James achieved the best results. In this experiment, Grasshopper was the slowest platform.

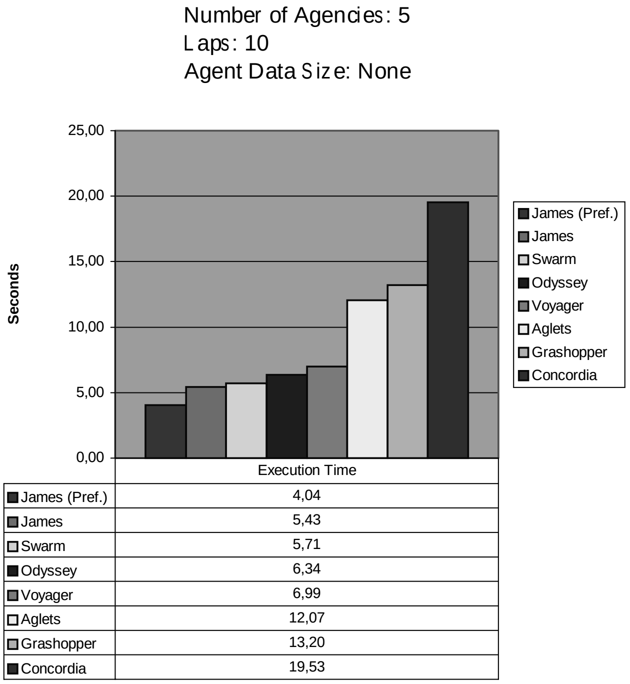
4.1.8. Experiment #8

This experiment departs from the previous ones: this time



	Execution Time
■ James (Pref.)	3,00
■ James	4,62
■ Odyssey	5,48
□ Swarm	7,30
■ Voyager	7,60
□ Aglets	9,99
■ Grashopper	10,48
■ Concordia	

Fig. 6. Execution time with 5 Agencies, 1 lap, data size = 1 Mb, and no caching.

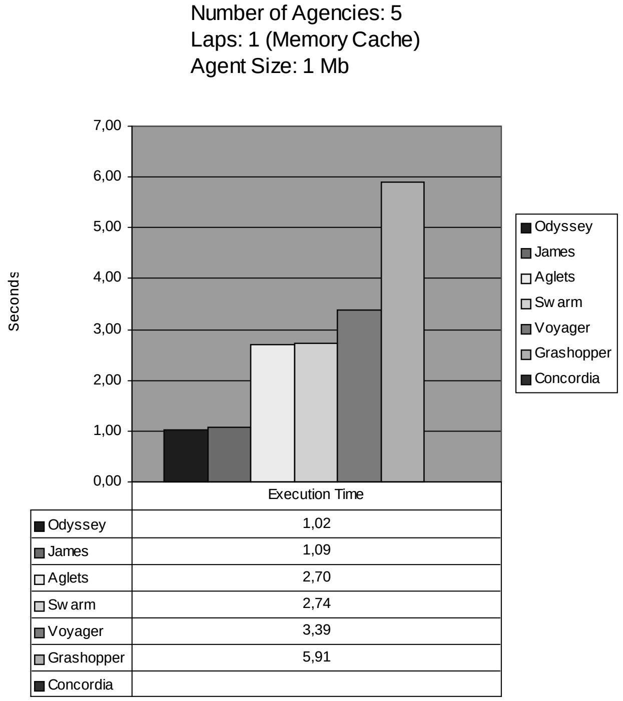


	Execution Time
■ James (Pref.)	4,04
■ James	5,43
□ Swarm	5,71
■ Odyssey	6,34
■ Voyager	6,99
□ Aglets	12,07
■ Grashopper	13,20
■ Concordia	19,53

Fig. 8. Execution time with 5 Agencies, 10 laps, no additional data.

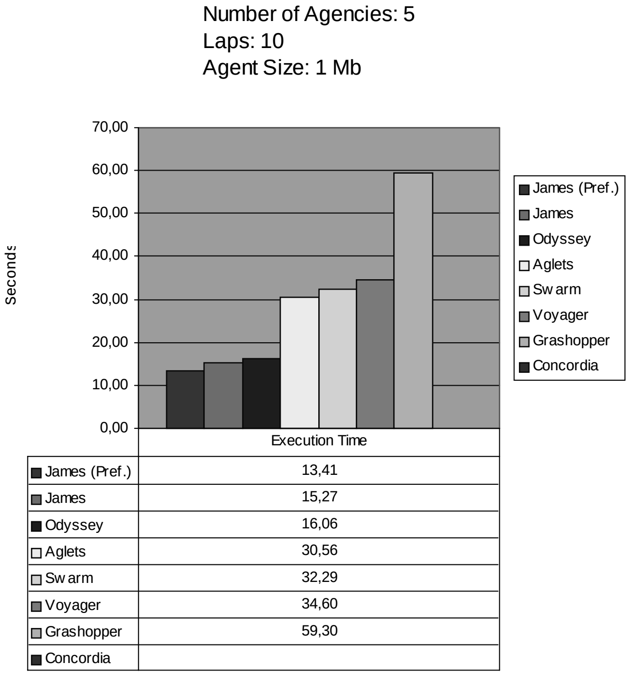
the agent had to execute 10 laps in the itinerary of 5 Agencies. Increasing the number of laps allowed us to observe the behaviour of the caching mechanisms and the way the platforms recycle the communication channels that are used by the mobility sub-system. We started by using an agent with small size. The results are presented in Fig. 8.

The performance of James was still the best one, although with minimal differences to Swarm, Odyssey and Voyager.



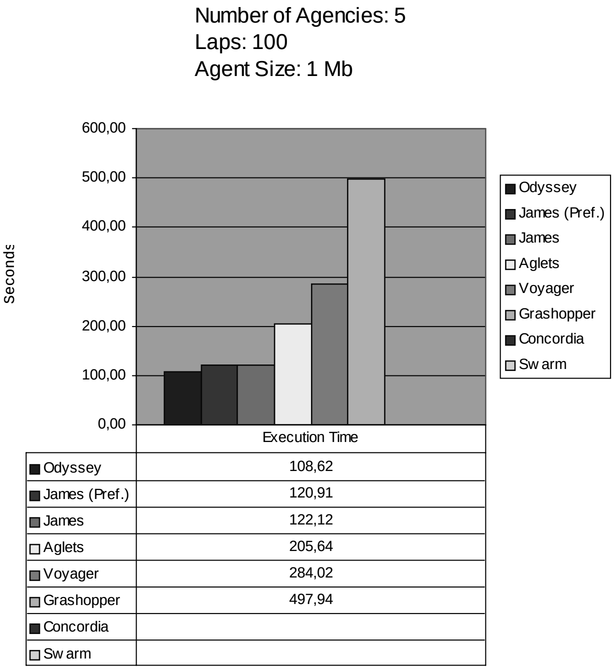
	Execution Time
■ Odyssey	1,02
■ James	1,09
□ Aglets	2,70
□ Swarm	2,74
■ Voyager	3,39
■ Grashopper	5,91
■ Concordia	

Fig. 7. Execution time with 5 Agencies, 1 lap, data size = 1 Mb, but using caching.



	Execution Time
■ James (Pref.)	13,41
■ James	15,27
■ Odyssey	16,06
□ Aglets	30,56
□ Swarm	32,29
■ Voyager	34,60
■ Grashopper	59,30
■ Concordia	

Fig. 9. Execution time with 5 Agencies, 10 laps, data size = 1 Mb.



	Execution Time
■ Odyssey	108,62
■ James (Pref.)	120,91
■ James	122,12
□ Aglets	205,64
■ Voyager	284,02
■ Grashopper	497,94
■ Concordia	
□ Sw arm	

Fig. 10. Execution time with 5 Agencies, 100 laps, data size = 1 Mb.

Nevertheless, when we used the prefetching scheme James was 5 times faster than Concordia.

4.1.9. Experiment #9

This experiment was similar to the previous one, but this time we increased the agent size by 1 Mb. Once again the Concordia system was not able to execute the application without crashing. James and Odyssey were reasonably faster than the other platforms. Once again, Grasshopper was the slowest platform. The results are shown in Fig. 9.

4.1.10. Experiment #10

In this final experiment for the execution time we have done some stress testing of the platforms, by using a mobile agent of about 1 Mb and running it 100 laps over the itinerary of 5 Agencies. The results are presented in Fig. 10.

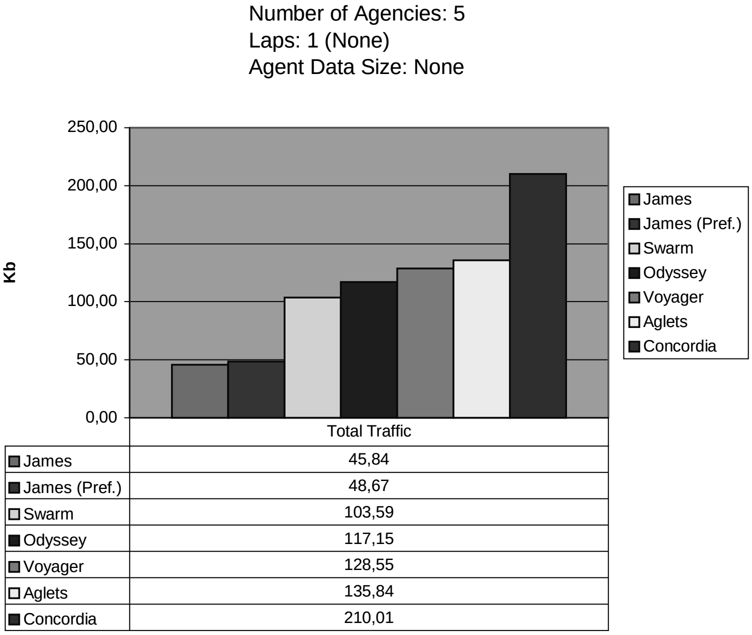
Two of the platforms, Concordia and Swarm, were not able to execute this agent without crashing. Odyssey and James were the fastest platforms, while Grasshopper was the slowest one. It was 5 times slower than the Odyssey system.

4.2. Measuring the network traffic

4.2.1. Experiment #11

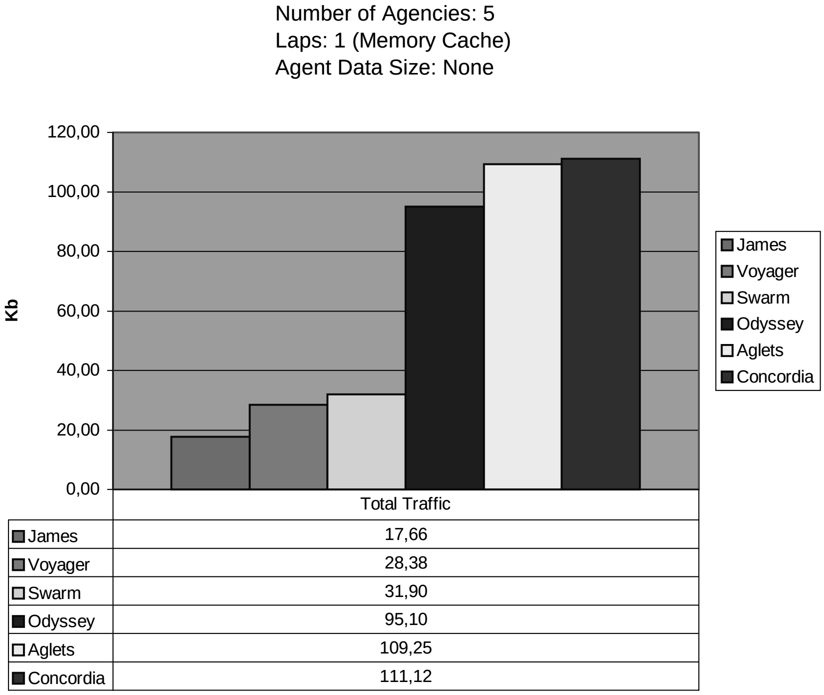
In this experiment we measured the whole traffic in the network that is imposed by the application and the platform protocols. These results were collected by using a network sniffer (Sniffer Pro). This metric is useful to evaluate the degree of optimization that was introduced in the mobility sub-system and the network overhead that is introduced by the protocols. Fig. 11 presents the first results that were taken with 5 Agencies. The agent had no additional data and executed 1 lap in its itinerary without making use of caching. The network traffic is represented in Kbytes.

As can be seen the James platform introduces a small amount traffic in the network, when compared with the other platforms. This shows some benefits from the optimisations we have in the platform protocols. The version of



	Total Traffic
■ James	45,84
■ James (Pref.)	48,67
□ Swarm	103,59
■ Odyssey	117,15
■ Voyager	128,55
□ Aglets	135,84
■ Concordia	210,01

Fig. 11. Network traffic (in Kb) with 5 Agencies, 1 lap, and no additional data.



	Total Traffic
▨ James	17,66
▨ Voyager	28,38
□ Swarm	31,90
■ Odyssey	95,10
□ Aglets	109,25
■ Concordia	111,12

Fig. 12. Network traffic (in Kb) 5 Agencies, 1 lap, but making use of caching.

James that used code prefetching imposed more traffic than the other version due to the additional messages that are necessary to implement that scheme. Concordia was the platform that introduced more traffic in the network.

4.2.2. Experiment #12

In the second experiment of this series, we have also used 5 Agencies, a small size agent that runs one lap of its itinerary. However, we activated the use of caching in all the platforms, to reduce some of the network traffic due to the distribution of code. The results are shown in Fig. 12. In this situation, James was still the platform that introduced the smallest amount of traffic in the network. Voyager and Swarm presented similar results. Odyssey, Aglets and Concordia were the platforms that introduced more traffic.

5. Conclusions about the results

In this section, we will present some conclusions about the behaviour of each platform in this benchmarking study. We will also relate some facts that have been observed during the execution of the tests. All the following opinions should not be understood as free criticism, but rather as some feedback to the platform developers to improve some weak points of their systems. The normal reader should be also interested to know about the real performance, the robustness, the good points and the weak points of these platforms.

5.1. Aglets

Aglets SDK is probably the most famous platform of mobile agents. The results show that it is quite a robust platform and it has passed all the tests without crashing. The performance is not so good when compared with other platforms. For instance, the James platform is 2–14 times faster than Aglets, depending on the test cases. The caching mechanisms seem to be not so efficient. We have done some profiling experiments where we detected there is some garbage left in the memory of the Agencies. This memory leak can lead to a deterioration of the performance of the application over time.

5.2. Concordia

Concordia is another well-known platform. Unfortunately, the results show that this platform is not very robust in situations of stress testing. We could not run the benchmark with a big size agent (~1 Mb) since it always gave an *OutOfMemory* error. Although the Agency did not hang up at the first time the GUI interface crashed every time we tried to create a second agent of that size. The garbage collection within the platform is also not done in appropriate way and there is a big deterioration in the execution of agents when we perform some consecutive experiments. Performance is another weak point of Concordia, as can be seen by the results presented. This platform is also the one that generated more network traffic.

5.3. Voyager

Voyager is a commercial platform with hundreds of users. The performance results are not brilliant, and we can say this platform is in the middle of the table. However, there are some issues related with some lack of robustness of this platform. Some times we got the *OutOfMemory* error and the platform crashed completely. There were some test cases that could not be done at any time. It is important to notice that this situation happened with big size agents (~1 Mb) but also with the small size agents. The platform produces a big amount of network traffic when not making use of the caching mechanism.

5.4. Odyssey

Odyssey is the Java-based successor of Telescript. The results have shown that this platform is very robust: it did not crash in any test we have made. The performance is also very good and it presented the best execution times, together with James. The only drawbacks we found was some lack of functionality and the absence of graphical interface for the management of the application and the launching of mobile agents.

5.5. Jumping Beans

The evaluation copy we had from Jumping Beans only allows the execution with three machines. The number of tests we could perform was therefore quite limited. However, those tests were enough to conclude that this platform has really a poor performance. In some cases, it was 40 times slower than the other platforms. The reason for this poor performance is simple: every time a mobile agent wants to migrate from machine A to machine B it has to go first to the Agent Manager. This Manager is a point of bottleneck and the platform is not scalable. The platform is also not very robust in situations of stress testing: for instance, it was not possible to execute the big size agent (~1 Mb) without giving an *OutOfMemory* error.

5.6. Grasshopper

This platform has a very user-friendly graphical interface and a comprehensive set of features. In fact, this is the platform that presents the higher functionality. However, the performance of Grasshopper is not very good: it was 2–5 times slower than the James platform. Version 1.2 of the platform had some minor bugs and some problems of robustness. However, most of them have corrected in a recent release (v1.2.2).

5.7. Swarm

This platform presented some problems of stability, although it can have some good performance results. The platform seems to open a channel between all the Agencies. If some of these channels are not well established at the beginning of the execution, the agent's migration can not be done properly and the application hangs up. This situation has happened several times, showing that there are some problems to be solved in this system. The GUI interface is a bit confusing and crashes periodically. The platform also had some problems when using big size agents: it crashed very often and had to be completely rebooted.

5.8. James

The James platform was devised and implemented with performance and robustness in mind. Several mechanisms have been introduced to optimize the migration of mobile agents (see Ref. [9]) and the platform has been enhanced with comprehensive support for fault-tolerance and resource-control. It seems these techniques have introduced clear benefits. In most of the test cases, James was the platform with the best level of performance and it presented a very good level of robustness. The resource-control mechanisms have been quite useful to increase the stability of the applications. However, James is still not a commercial platform and some of the programming features should still be improved to simplify the life of the application programmers.

6. Final remarks

It is clear that performance, network traffic and robustness are not the only metrics that should be taken into account. The list of features and the overall functionality of each platform also play a very important role. To get a complete picture about the best platforms the interested reader should still take a look to the programming support of each system and the comparison of features that have been presented elsewhere [3–7]. In this paper, we were only evaluating the performance and the robustness of James and to see how it compares with the other similar Java-based mobile agent systems. The results are very promising and James has a very competitive position.

We are now interested to evaluate the performance of the mobile agent paradigm against traditional client/server solutions. Some preliminary, but quite interesting, results have been reported in Ref. [10]. Our main concerns go now to improve the programming support of the platform and to help in the deployment of agent-based applications that make use of James.

References

[1] D. Lange, M. Oshima, Seven good reasons for mobile agents, Communications of the ACM 42 (3) (1999) 88–89.

[2] L.M. Silva, P. Simoes, G. Soares, P. Martins, V. Batista, C. Renato, L.

Almeida, N. Stohr, James: a platform of mobile agents for the management of telecommunication networks, Proc. IATA'99, Intelligent Agents for Telecommunication Applications, Stockholm, Sweden, August 1999.

[3] Voyager and agent platforms comparison, Technical Report available at: http://www.objectspace.com/products/voyager/.

[4] J. Kiniry, D. Zimmerman, A hands-on look at Java mobile agents, IEEE Internet Computing July–August (1997) 21–30.

[5] T. Ugai, M. Bursell, Comparison of autonomous mobile agent technologies, Internal Report, FollowMe Project, APM, Cambridge, UK, October 1997.

[6] M. Corkery. A review of state of the art in mobile agent systems, Technical Report, Department of Computer Science, National University of Ireland Maynooth, Ireland, 1998.

[7] A. Guther, M. Zell, Platform enhancement requirements, Internal Report, Project MIAMI (ACTS Program AC338), 1998, URL of the MIAMI Project: http: //www.fokus.gmd.de/research/cc/ima/miami.

[8] E. Kovacs, K. Rohrle, M. Reich, Integrating mobile agents into the mobile middleware, Proc. Second Int. Workshop on Mobile Agents, MA'98, Stuttgart, Germany, September 1998, pp. 124–135.

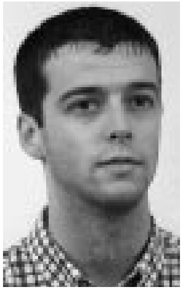
[9] G. Soares, L.M. Silva, Optimizing the migration of mobile agents, Proc. MATA'99, Mobile Agents for Telecommunication Applications, Ottawa, Canada, October 1999.

[10] L.M.Silva, G.Soares, Comparing the performance of mobile agents with client/server solutions, Technical Report, James Project, May 1999.

Luis M. Silva is an Assistant Professor at the Department of Computer Science in the University of Coimbra, Portugal. He has a degree of Computer Science from the University of Coimbra (1990), a MSc from the Technical University of Lisbon (1993) and a PhD in Computer Science from the University of Coimbra (1997). His research interests include distributed and parallel computing, mobile agent systems and fault-tolerance.

Guilherme Soares is a Masters student at University of Coimbra, Portugal. His research interests include distributed computing, web technologies and mobile agent systems. He is currently working on the development of the James platform.

Paulo Martins is a Masters student at University of Coimbra, Portugal. His research interests include distributed computing, resource management and mobile agent systems. He is currently working on the development of the James platform.



Victor Batista is a Masters student at University of Coimbra, Portugal. His research interests include distributed computing, fault-tolerance and mobile agent systems. He is currently working on the development of the James platform.

Luis Santos is a Masters student at University of Coimbra, Portugal. His research interests include mobile agent systems, software deployment, distributed computing and network management. He is also a teaching assistant at the Technical Institute of Engineering in Coimbra.