JAMES: A Platform of Mobile Agents for the Management of Telecommunication Networks

Luis Moura Silva[1], Paulo Simões[1], Guilherme Soares[1], Paulo Martins[1], Victor Batista[1], Carlos Renato[2], Leonor Almeida[2], and Norbert Stohr[2]

[1] CISUC - Dep. Eng. Informatica, University of Coimbra
P-3030 Coimbra, Portugal
luis@dei.uc.pt
[2] Siemens S.A., Rua Irmãos Siemens, N. 1
P-2720-093 Amadora, Portugal

Abstract. This paper presents an overview of JAMES, a Java-based platform of mobile agents that is mainly oriented for the management of data and telecommunication networks. This platform has been developed on behalf of a Eureka Project (Σ!1921) and the project partners are Siemens SA, University of Coimbra and Siemens AG. We describe the main architecture of the platform giving more emphasis to the most important features. To show the effectiveness of some of the techniques that have been implemented we will present some performance results that compare the JAMES platform with the Aglets Workbench.
The main target of our platform is network management and telecommunication applications. In this line, we have done a Java-based implementation of SNMP that has been integrated within the platform. The industrial partners of our project (i.e. Siemens S.A.) have developed a prototype application for TMN performance management. Although it is still a prototype it is being used to validate the technological advantages of using mobile agents in the management of telecommunication networks.

1 Introduction

The main goal of the JAMES project is to develop an infrastructure of Mobile Agents with enhanced support for network management and try to exploit the use of this new technology in some telecommunications software products. A Mobile Agent corresponds to a small program that is able to migrate to some remote machine, where it executes some function or collects some relevant data and then migrates to other machines in order to accomplish another task. The basic idea of this paradigm is to distribute the processing throughout the network: that is, send the code to the data instead of bringing the data to the code.

The existing applications in the management of telecommunication networks are usually based on static and centralized client/server solutions, where every element of the network sends all the data to a central location that executes the

whole processing over that data and provides the interface to the user operator. By consequence, they are not flexible, they have problems of scalability and they produce too much traffic in the network.

The use of Mobile Agents in this kind of applications represents a novel approach and potentially solves most of the problems that exist in centralized client/server solutions. The applications can be more scalable, more robust, can be easily upgraded or customized and can reduce the traffic in the network.

The `JAMES` project will try to exploit all these technological advantages and see how the mobile agents technology can be used in software products that are been developed by Siemens SA. The project involves the development of a Java-based software infrastructure for the execution of mobile agents. The use of Java was motivated for reasons of code portability.

In the last few years the use of Mobile Agent technology has received an extraordinary attention from several Universities and research institutes and a notable investment from several companies [1]. Mobile agents have been applied in several areas, like mobile computing, electronic commerce, Internet applications, information retrieval, workflow and cooperative work, network management and telecommunications [2–5]. Several commercial implementations of mobile agents have been presented in the market, including Aglets from IBM [6], Concordia from Mitsubishi [7], Odyssey from General Magic [8], Voyager from ObjectSpace [9] and Jumping Beans from AdAstra [10]. Although these software products have some very interesting features they are too much general-purpose and do not provide any special support for network management.

In our project, we are developing from scratch a new Mobile Agent infrastructure that is being tuned and customized for the applications we have in mind in the area of telecommunications and data network management. For this reason we decided to develop another platform from scratch that would take into account the following list of issues:

- high-performance and efficient code migration,
- fault-tolerance and robustness,
- support for network management,
- flexible code distribution and easy upgrading,
- mechanisms for resource control,
- disconnected operation,
- easy-to-use programming interface,
- 100% pure Java implementation,
- support for CORBA [11].

These are the main goals of our platform. Some of them have been already achieved in the first release, while the others are scheduled for future versions.

During this project, the platform will be used in two software products: one in the area of telecommunication and other for data network management. Our industrial partners (Siemens S.A.) have been developing a prototype application in the area of performance management by using our platform of mobile agents. This prototype is already finished and we are now conducting a benchmarking study to compare the use of mobile agents over traditional client/server solutions

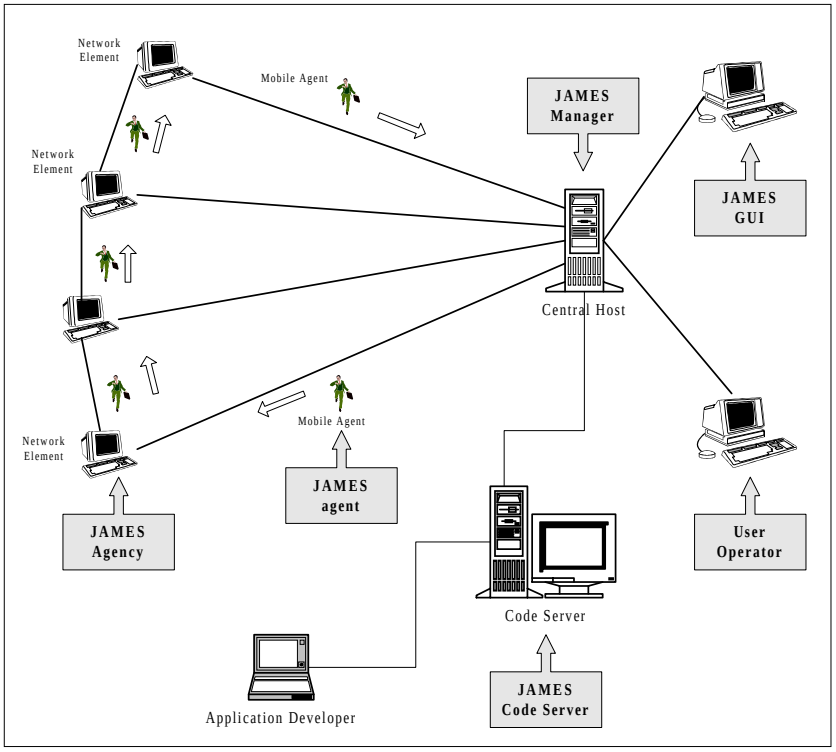

Fig. 1. An Overview of the JAMES Platform

to see if we corroborate some of the advantages of this new paradigm in the field of distributed computing.

The rest of the paper is organized as follows: section 2 presents an overview of the JAMES Platform, giving more emphasis to the most important features of the system. Section 3 presents some experimental results where we compare the performance of JAMES against the Aglets system. Section 4 describes the work we have done in the integration of SNMP in our platform, while section 5 presents a brief overview of the prototype application that has been implemented by Siemens S.A. Section 6 concludes the paper.

2 The Architecture of the JAMES Platform

The JAMES Platform provides the running environment for mobile agents. There is a distinction between the software environment that runs in the manager host and the software that executes in the Network Elements (NEs): the central host executes the JAMES Manager while the nodes in the network run a JAMES Agency. The agents are written by application programmers and will execute on top of that platform. The JAMES system will provide a programming interface that

allows the full manipulation of Mobile Agents. Fig. 1 shows a global snapshot of the system, with a special description of a possible scenario where the mobile agents will be used.

Every NE runs a Java Virtual Machine and executes a JAMES Agency that enables the execution of the mobile agents. The agents will migrate through the machines of the network in order to access some data, execute some tasks and to produce reports that will be sent back to the JAMES Manager. There is a mechanism of authentication in the JAMES Agencies to control the execution of agents and to avoid the intrusion of non-official agents. The communication between the different machines is done through stream sockets. A special protocol was developed to transfer the agents across the machines in a robust way and to provide atomicity to the occurrence of failures.

The application developer writes the applications that are based on a set of mobile agents. These applications are written in Java and should use the JAMES API for the control of mobility. After writing an application the programmer should create a JAR with all the classes that make part of the mobile agent. This JAR file is placed in a JAMES Code Server. This server can be a different machine or in the same machine where the JAMES Manager is executing. In both cases, it maintains a code directory with all the available JAR files and provides a mapping to the corresponding mobile agents. The Code Store can be replicated if we want to increase the availability of the code.

The host machine that runs the JAMES manager is responsible for the whole management of the mobile agent system. It provides the interface to the end-user, together with a Graphical User for the remote control and monitoring of agents, places and applications. A snapshot of this interface is presented in Fig. 2. The JAMES GUI is the main tool for management and administration of the platform. With this interface, the user can manage all the Agents and Agencies in the system.

The JAMES platform is still in its first version but the main features of the platform can be summarized in the following list:

- Kernel of the JAMES Manager and JAMES Agency,
- Service for remote updating of Agents and Agencies,
- Flexible code distribution (caching and prefetching schemes),
- Atomic migration protocol,
- Support for fault-tolerance through checkpoint-and-restart,
- Reconfigurable itinerary,
- Support for disconnected computing,
- Watchdog scheme and system monitoring,
- Mechanisms for resource control,
- Logging and profiling of agent activity,
- GUI interface to allow the remote control of agents,
- Interface with CORBA,
- Integration of a Java-based SNMP stack into the platform,
- Inter-agent communication (through JavaSpaces [12]),
- Multi-paradigms for agent execution (simple agent, migratory agents and Master/Worker model).

Fig. 2. The Graphical User Interface

The explanation of all these mechanisms is out of scope of this paper, but we will give some emphasis to some of the issues that have been considered as more important for our domain of applications: high-performance, flexible code distribution, remote software upgrading, reliability and robustness.

2.1 Flexible Code Distribution

Each Mobile Agent has a specific `Itinerary` with a set of tasks (`Missions`) to be executed across the `JAMES` Agencies. The agent is usually created and dispatched by the `JAMES` Manager.

The code classes of the `JAMES` agents are grouped in `JAR` files - there is one unique JAR file per agent. The first step is to register the agents in the platform. All the JAR files that have been registered are stored in the Code Store.

To enable a more efficient migration of the agents we have implemented a flexible distribution of their code. The Agencies have their own local disk repository of agent JAR files (disk cache) and there is also a local memory cache of agent classes per Agency. The memory cache is further separated between the different agents running on the same Agency. The disk and the memory cache make use of a LRU policy to replace the JAR files when the cache is full. The cache size can be customized by the platform administrator.

Every time an agent arrives at an Agency there is a hierarchical progressive scheme to fetch the classes of the agent: first, the agent classes are searched in the memory cache; then, in the local disk cache. If the JAR file, associated to the agent, is not in the local cache the Agency tries to get it from the previous Agency of the Itinerary. If the classes could not be fetched from this machine then the Agency contacts the central Code Store.

This hierarchical search scheme provides great flexibility in the code distribution. There is always a central point where all the code is stored but there are multiple choices that can be used to improve performance and scalability.

2.2 Code Prefetching for High-Performance

The caching of JAR files tries to exploit the locality of code of the mobile agents. However, we have improved even further the performance of the agent system by optimizing the inner parts of the migration protocol and by implementing a code prefetching technique to speed up the execution of migratory agents.

When an agent is created and launched to the `JAMES` platform it has its own Itinerary of Agencies. There is always an overhead in the startup time when the classes of the agent are not in the local caches of the Agencies. This startup time corresponds to the time that is spent to fetch the classes from the local disk repository, or from another Agency or from the Code Store.

We can reduce this stratup time by informing all the Agencies that some particular agent is going to execute there. The `JAMES` Manager sends some information to all the Agencies (except the first one) of the Itinerary. With this information the remaining Agencies can load in advance the class files while the agent is still executing in the first Agency of the Itinerary.

The `JAMES` Manager maintains a global information about the caching information of the Agencies and it can even send the JAR file in advance to some Agency, if it that file does not exist in the remote cache.

2.3 Remote Software Upgrading

The `JAMES` platform provides a flexible mechanism for the remote upgrading of mobile agents, as well as Agencies. Each Agency is seen as a stationary agent: it cannot move around the network once installed in a machine, but it should be easy to upgrade, customize and install by a central host.

The `JAMES` Agency is composed by two modules: a small `jrexec` daemon and the Agency itself. The `jrexec` daemon is a static piece of software; once installed it does not need to be constantly upgraded. The Agency itself is a more dynamic module, since it can be changed whenever required.

The Java `jrexec` daemon implements an instance of the *Class Loader* and receives some network commands regarding the installation and control of the `JAMES` Agency. This daemon will be instantiated every time the machine is booted. The daemon can receive a JAR file containing a `JAMES` Agency and it will perform its local installation. After this first step, the `JAMES` Manager can send some remote commands to the `jrexec` Daemon:

- Refresh the local memory by calling the Java garbage collector;
- Kill the local Agency;
- Install a new Agency on the local machine;
- Upgrade the local Agency with a new set of classes.

This scheme will be useful in dynamic environments since it provides a flexible way to upgrade remote software.

2.4 Fault-Tolerance

The JAMES platform has some special support for fault-tolerance. The first version includes a checkpoint-and-restart mechanism, a failure detection scheme, an atomic migration protocol and some support for fault-management. The platform should be able to tolerate any failure of a mobile agent, a JAMES Agency or the JAMES Manager.

Fault-Tolerance at the Agencies. Periodically, the internal state of the JAMES Manager and Agencies are saved as a checkpoint in persistent storage. The internal state consists of all the internal objects that keep all the relevant state about the platform and the execution of the agents. If any of the servers (Agency or Manager) fails or is simply shut down the system should have enough information to recover the server to a previous consistent state. This state is retrieved from persistent storage and all the internal state can be reconstructed. The checkpointing mechanism makes use of the Java object serialization facility and is completely transparent to the application programmer.

Fault-Tolerance at the Mobile Agents. If there is a communication or node failure that affects the execution of the agent the system should have some way to assure a forward progress of the mobile agent. This is also achieved through a checkpointing mechanism. When a mobile agent finishes a task in a JAMES Agency its internal state is saved to stable storage before being transmitted to the next destination. The agent is migrated to another host but its data will remain in stable storage until it has been successfully restarted in the next Agency. When it is restarted in the new place the system takes a new checkpoint of the agent and the previous place is informed. The previous checkpoint of the agent can then be removed from stable storage. This checkpointing mechanism is transparent to the application developer and is incorporated in the migration protocol to assure the atomicity of the agent transfer. This means that or the agent is completely migrated to its destination or whenever is a failure the agent is not lost and the system is able to recover the agent in the previous Agency. We have used a conventional two-phase commit protocol to achieve the exactly-once property in the migration of the agents.

When there is a failure in the migration protocol or if one of the Agencies in the Itinerary is not available the agent can execute one of the three following procedures:

1. go back to the JAMES Manager;
2. jump to the next available Agency in the itinerary;
3. or just wait until the destination Agency is back alive.

The procedure to follow by a mobile agent in the occurrence of a failure can be customized by the application programmer.

Resource Control. One important feature in a platform of mobile agents is a good set of mechanisms for resource control. In the JAMES platform we have included some schemes to control the use of some important resources of the underlying operating system, namely: the use of threads, sockets, memory, disk space and CPU load.

These schemes have proved to be very effective when we were doing some stress testing. In some situations when the Agencies are running almost out of any of those resources it was still possible to maintain the platform up and running. Without such mechanisms the Agencies would normally hang up. With resource control the platform has become much more robust and this is a crucial step if we want to use it in production codes.

3 Some Performance Results

In this small section we would like to show the effectiveness of some of the techniques that we have implemented in the JAMES platform.

We decided to compare the performance of our platform with the Aglets Workbench from IBM Tokyo [6]. We used a simple mobile agent that roams a network of five computers to get a report about the free memory of each machine. These machines were running Windows NT and JDK 1.1.5 and were connected through a dedicated 10 Mbit/sec Ethernet network. Each machine was executing a JAMES Agency, while the JAMES Manager was installed in a separated computer.

In Fig. 3 we present the results that were taken when executing by the first time the mobile agent with the JAMES platform and with the Aglets system. The results were taken with two versions of JAMES: one that makes use of the prefetching techniques (JAMES-Pref) and another one that uses the normal code distribution procedure. We increased the internal size of the agent to see its relevance in the migration time. We used three different sizes: 1 Kb, 100 kb and 1 Mbytes.

In both cases the JAMES platform presented a better performance than the Aglets system. When using the prefetching technique we were able to improve the performance against the normal version of JAMES by a factor of 1.8. Also, the JAMES platform with the prefetching technique was 4 times faster than the Aglets system. It can also be seen that the performance gap between JAMES and Aglets increases directly when we augment the size of the mobile agent.

JAMES is being developed with high-performance in mind since this is an important issue in our target domain of applications while the Aglets Workbench is mainly oriented for Enterprise computing and Internet applications. In these two areas performance is not a so crucial aspect.

In the previous experiment we were executing the mobile agent by the first time in the network. The experiment was useful to measure the importance of the prefetching technique but the application did not make any use of the caching scheme. In the next experiment, whose results are presented in Fig. 4, we present the second execution of the mobile agent using JAMES and Aglets. This time the class files of the agent were residing in the local caches of the Agencies in the

Fig. 3. Comparing the performance of JAMES with Aglets Workbench (the use of prefetching)

network. The `JAMES` platform was in average 4.3 times faster than the Aglets system for the second execution of the agent. The interesting aspect is the fact that Aglets also has a caching mechanism inside its platform, but apparently it was not so efficient as the scheme implemented on `JAMES`.

Although this section did not present a comprehensive study, the performance results of the `JAMES` platform seem to be quite promising when compared with a general-purpose system of mobile agents.

4 Integration of SNMP

In this section we describe in some detail the support we have included in `JAMES` platform for SNMP-based network management.

4.1 SNMP and Mobile Agents

Classic management applications use protocols like SNMP [13] and CMIP [14] to interface with management services in heterogeneous environments. Mobile Agents will not replace these protocols. Instead, they will complement them with powerful programming metaphors allowing more efficient solutions for network management.

Mobile agents provide a very attractive approach to incorporate mobile code into the existing local management services, in order to perform intelligent tasks closer to management data [15]. However, some management protocols are still necessary to retrieve and process the management information. This is the case when some of the managed NEs are unable to host mobile agents, the management services of NEs are not directly available to the hosted mobile agents or the interfaces with management services are non-standardised.

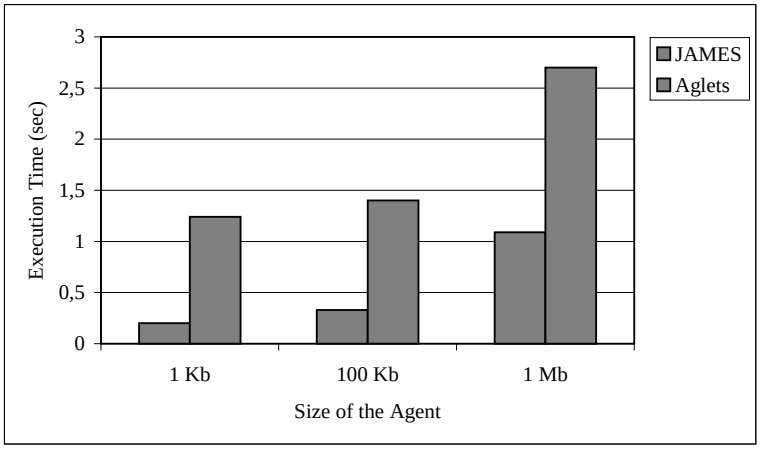

Fig. 4. Comparing the performance of JAMES with Aglets Workbench (the use of caching)

Integration of management protocols with mobile agents can be relegated to the applications' developer, eventually using the same "general-purpose" libraries used by static management systems. However, code mobility, security constraints and resource usage control imposed on mobile agents' applications seriously limit the usage of these protocols without explicit support from the underlying infrastructure. This is why `JAMES` includes explicit support for interoperability with SNMP devices and applications.

There are now several worthwhile projects mixing SNMP with mobile agents, like the Perpetuum Mobile Procura Project [16, 17], the Discovery platform [18], the Astrolog/Magenta platform [19] and the INCA Architecture [20]. Some of these projects relegate SNMP support to applications' developers whilst others integrate SNMP within the mobile agents' infrastructure. The `JAMES` platform also integrates SNMP within the platform, but departs from these projects for the following reasons:

- it provides maximum interoperability, since it covers three different service ranges: interaction with local and remote SNMP-agents; interaction between SNMP-managers and mobile agents; and infrastructure management using SNMP;
- it provides full support for agents mobility;
- it is a completely optional feature of `JAMES`, imposing no additional overheads when turned-off;
- it does not require any intervention on existing SNMP devices and SNMP Managers, preserving the overall portability.

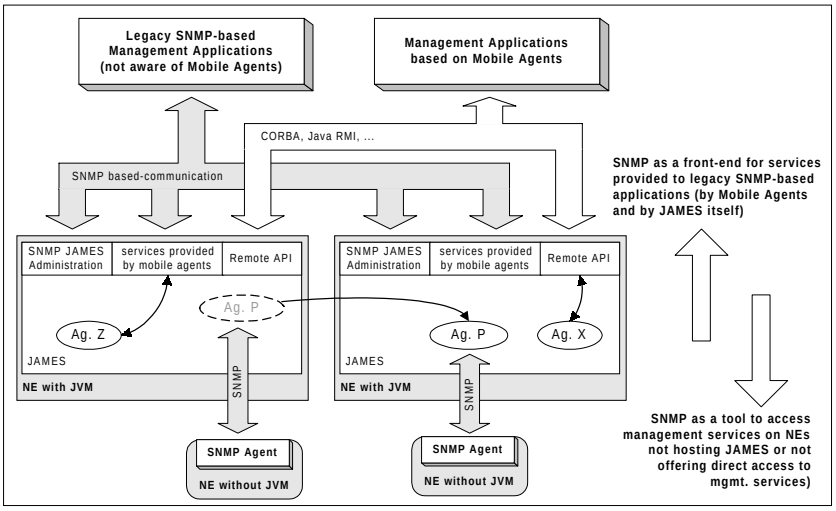

Fig. 5. Proposed Integration Framework

4.2 SNMP Services Provided by JAMES

JAMES includes a framework of full-fledged SNMP services already integrated and available to the NM-application developer, resulting in broader application fields and reduced development costs. As represented in Fig. 5, three nuclear SNMP services were considered:

- a service allowing mobile agents to interact with SNMP-agents, acting as SNMP-managers;
- support for communication between SNMP-managers and mobile (or stationary[1]) agents;
- a management service allowing legacy management platforms to administer the JAMES infrastructure itself using SNMP.

These services provide the following features:

- management of NEs not supporting JAMES Agencies but equipped with SNMP-agents;
- management of NEs supporting JAMES Agencies but restricting direct access to management information for security or architecture reasons;
- management of the JAMES infrastructure itself as an SNMP-service;
- usage of mobile agents to deploy intermediary management services layered between NEs (SNMP capable or not) and legacy SNMP-managers. These could be new services or just management information processing closer to the NEs;
- usage of mobile or stationary agents for fast development and deployment of SNMP services.

[1] JAMES supports "stationary agents" in the sense that agents can make little or no use at all of their migration capability.

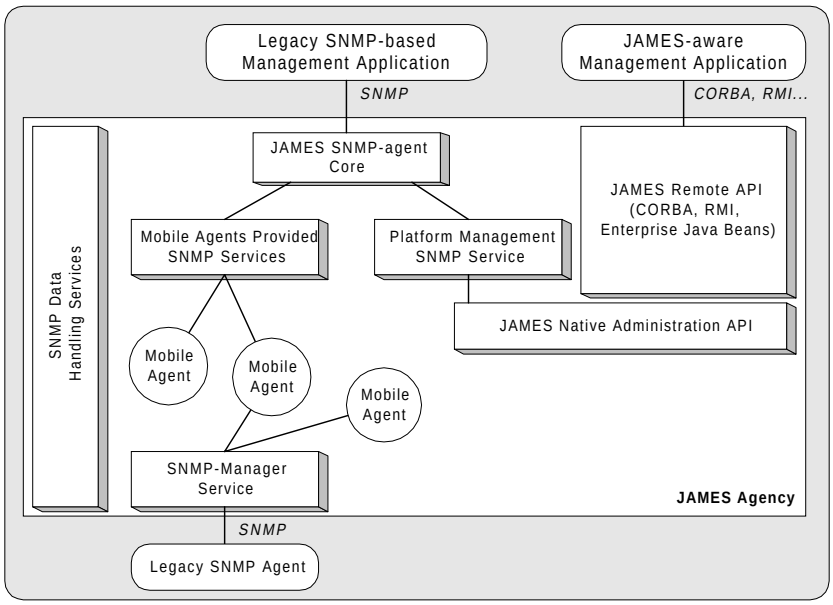

Fig. 6. High-Level Structure of SNMP Services of JAMES

4.3 Design of SNMP Services

Since SNMP is just one of several protocols to be used by network management applications, the following design constraints were defined:

- SNMP support must be optional, not increasing resource usage when turned-off;
- SNMP support should not increase the complexity of the platform;
- design of SNMP support may not compromise the platform scalability and functionality;
- SNMP support must be portable across different hosts without conflicting with SNMP services already installed in the hosts, like native SNMP agents.

These resulted in a modular design Fig. 6 where SNMP services are placed outside the platform core and can be dynamically installed and removed, without imposing a permanent overhead in the `JAMES` infrastructure. Most services consist themselves of mobile agents (the "Service Agents", granted with exceptional permissions to access necessary resources) providing services to common agents through inter-agent communication. The Agency offers a directory service where common mobile agents can locate the Service Agents (or implicitly require their installation). This solution provides an elegant lightweight framework to support specific services. In the future, new kinds of services can be easily integrated into the `JAMES` platform.

SNMP Data Handling Services. These services include all the tools needed to handle SNMP data and SNMP Protocol Data Units. These tools are available as a set of Java classes for high level representation of SNMP data types and PDUs, and for ASN.1/BER encoding [21]. Mobile agents impose no particular requirements to this Service, meaning general purpose Java-based SNMP tools, like [22], could have been used without prior adaptation.

SNMP Manager Service. This service allows mobile agents to interact with SNMP Agents using a manager-API, to query SNMP-agents, and a Trap Listener that receives SNMP Traps and redirects them to the interested mobile agents. When compared with similar Services integrated in classical management applications, this Service presents two key differences: support for mobility - mobile agents receive SNMP Traps independently of their present location and migrate without abandoning ongoing SNMP queries - and the service location within the platform - based on the already mentioned "Service Agents".

The SNMP Manager-API is based on the traditional concepts in most high-level SNMP stacks (*sessions* or *contexts*, *request* operations and event handlers), with protocol details being handled in a transparent way. This Service, located in "Service Agents", might be replaced with a third-party "classic" SNMP stack integrated in the Agent's code[2], trading-off mobility support.

The Trap Listener also uses traditional concepts found on other Trap Multiplexers. Mobile agents register their interest on the reception of certain SNMP Traps. Registrations may be valid just while the agent remains at the Agency, for a pre-defined period of time or for the agent's entire lifetime (in the last two options, the Trap Listener may have to forward the Trap Arrival Notification to the new location of the agent). The arriving SNMP Traps produce Trap Arrival Notifications.

Services for Interoperability with Legacy SNMP Managers. While the Service described in the previous section covers communication between mobile agents and SNMP agents, there three other services providing interoperability with legacy SNMP Managers (Fig. 7).

The SNMP-agent Core maintains an SNMP-agent with data being supplied by the two underlying services. The present interface used to register new variables or groups at the MIB-table, to issue Traps and to reply to SNMP requests is proprietary, although future use of standard protocols for expansible SNMP agents like DPI [23], as proposed by [24] is not excluded.

The `JAMES` SNMP-agent is independent of eventually existing native SNMP Agents, although integration would be more in the spirit of SNMP. Since native

[2] This is based assumption that since `JAMES` mobile agents implicitly control their migration, they can delay migration whenever completion of on-going SNMP transactions is crucial. This degrades performance and affects the programming model (agent migration has to become aware of SNMP transactions) but still allows for some mobility.

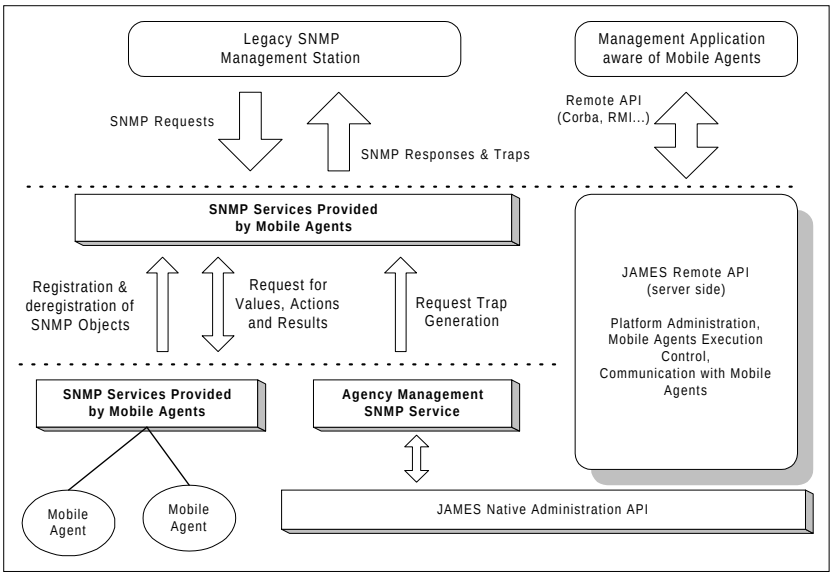

Fig. 7. JAMES Services for Integration with Management Applications

agents are either monolithic or based on a wide diversity of agent-expansion mechanisms, like SMUX [25] or DPI, there is no truly portable and non-intrusive integration method.

The JAMES SNMP-agent allows SNMP communication between legacy applications and mobile agents, opening the way for easy installation of new management services (corresponding to one or several mobile agents) available to legacy SNMP-based network management systems.

In such a scenario mobile agents can be used to pre-process data gathered from existing management services (thus offering higher level functionality), operate as SNMP proxies for NEs using proprietary management interfaces and dynamically install new management services.

The use of stationary or mobile agents for fast deployment of management services available to legacy applications is not new. The JDMK toolkit [26], although not based on mobile agents, shares the vision of a fast and flexible scheme to develop and deploy management services. It includes a complete set of tools to create and remotely install these services using Java and push/pull techniques. The Perpetuum Mobile Procura Project [16, 17] presents a framework where mobile agents use DPI to provide services to SNMP Managers.

The JAMES approach consists of a "service agent" where mobile agents interested on providing an SNMP interface must register their SNMP objects. Later on, SNMP requests from outside applications result in events passed to mobile agents. These events will then trigger predefined management actions resulting in SNMP responses.

The JAMES SNMP agent also allows SNMP-based management of the JAMES platform itself. Although a richer interface is available to dedicated applications using Java RMI and CORBA, a subset of management functions has been "translated" into an SNMP MIB that provides monitoring, fault-management and performance management. This is implemented using a "Service Agent" (the Agency SNMP Management Service) that acts as a translator between the SNMP-agent Core and the internal JAMES administration API (Figure 7). The intention is not to use SNMP to fully administer JAMES but to provide a simple SNMP-based management service.

5 Prototype Application for Performance Management

The software for TMN Performance Management is currently dominated by systems based on Client/Server technologies. In most cases, this approach results in monolithic, not scalable and hardly flexible solutions.

Typically, there are several network devices distributed across the network. In each device it is installed a static server (or agent) that is responsible for collecting raw data the local device, with little or even no pre-processing at all. At the manager host there is a client process that interacts with all the servers in the network devices, collects the information from them and provides the information required to the end-user. These servers (or agents) are static and proprietary processes. They are very difficult to upgrade or to customize. Some of the current solutions also suffer from an information bottleneck at the manager host due to its centralized nature. Some of the applications are very inefficient in the use of the network bandwidth, since most of the data has to be sent from the network devices to the manager site in order to be processed.

Additionally, raw data information generated in the TMN Network Elements has some "troublesome" characteristics, like:

- proprietary raw data formats;
- huge amount of raw data produced at a daily basis;
- variation of data formats depending on customer and/or software version installed.

Given this scenario, the use of Mobile Agents for TMN data collection provides several potential benefits. First of all, mobile agents allow some processing at the data sources which increases the scalability and reduces the traffic in the network. The robustness of the application is also improved through the use of autonomous mobile agents and the application provider will have a more flexible paradigm to deal with the diversity of network configurations. The application developer can create diverse collector agent versions, each one matching the proprietary raw data format involved, the software versions involved and the customer involved. Software upgrading is another important issue that comes from the use of mobile agents. The upgrade of an application requires only the coding of new agents or the automatic deployment of new agent versions over the

managed network. This way, the extension of the functionality or the installation of new software versions becomes more flexible and effortless.

In order to evaluate the use of mobile agents and the `JAMES` platform in real PM/TMN environments we have designed a prototype application to provide O&M Destinations Reporting, a component of TMN Performance Management Traffic measurements [27].

A commercial TMN application from Siemens S.A. already addresses this problem, making it possible to evaluate the advantages/disadvantages of Mobile Agents versus the use of the client/server paradigm in a real TMN environment.

5.1 Overview of the Application

The TMN application from Siemens collects and processes performance data from the NEs in order to produce a set of global reports about the performance of the network. This can be applied to the mobile and switched network. Right now, data collection and report building are two dissociated tasks: data is collected through file-transfer and is organized in a centralized relational database. This database is later queried to produce the performance reports. This two-step approach is based on a traditional client/server approach and presents some technical limitations: the data collection introduces too much traffic in the network and the overall system has some problems of scalability.

The benchmark application, designated as "EWSD Destination Reporting Application", is designed to reproduce a representative subset of the TMN application reports using mobile agents to collect and process the management data. This new version of the application is supposed to overcome some of the problems of the existing client/server version. It is structured into three different modules:

- an application GUI that handles the end-user requests and outputs the reports;
- a mobile agents handler, responsible for the control of the reporting features;
- the application specific mobile agents which are assigned to fulfil the required reports.

The benchmark application produces two different types of reports: *on-demand* and *predefined/scheduled* reports. *On-demand reports* correspond to requests to be immediately executed over the traffic destination raw data files stored in the remote NEs. Fig. 8 presents a snapshot from an *on-demand* report that shows the evolution of two combined traffic destination counters.

Predefined reports are a sort of report *templates* with a pre-defined behavior. This report templates can be used at a later time with user provided attributes like time window to be evaluated, NEs to be considered, type of traffic destination to be analyzed, etc.

5.2 Benchmark Study

The development of the benchmark application is already complete and a benchmarking study is being conducted, focusing on the following metrics:

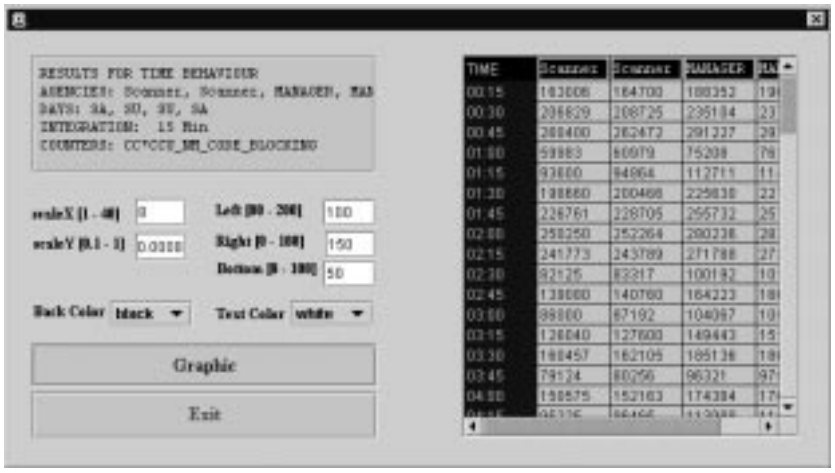

Fig. 8. Snapshot from the Prototype Application: a Time Behavior Report in Raw Table Format

- scalability and traffic bottlenecks;
- dynamic execution of mobile agents;
- performance and mobility;
- network traffic;
- persistency of the applications;
- robustness under stress testing;
- upgrading mechanisms for both agents and agencies;
- easiness to use and develop applications using the JAMES API;
- flexibility of customization.

The complete results of this benchmarking study should be complete within a couple of months.

6 Conclusions

The JAMES Project exploits the paradigm of mobile agents in the field of management of telecommunication networks: the use of Mobile Agents. We are developing a Java-based platform of mobile agents and we have some important goals in mind, like: high-performance, flexible code distribution, remote software upgrading, reliability, robustness and flexible support for SNMP and network management. The preliminary results about the performance of the platform seem to be quite promising.

The JAMES platform will be used in two software products: one in the area of TMN and another for data network management. Within this project we expect to show that Mobile Agents can overcome some of the problems that exist with traditional Client/Server solutions.

Acknowledgements

This project is partially supported by Agência de Inovacão and was accepted in the Eureka Program (Σ!1921). Special thanks to the rest of the team that is working in the project: Luis Santos, Fernando Bernardino and Rodrigo Reis (from the University of Coimbra), Jochen Reban, Patricia Monteiro and João Boavida (from Siemens S.A.) and Norbert Baumgart (from Siemens A.G.).

References

1. Agent Product and Research Activities. http://www.agent.org/pub/activity.html
2. Intelligent Agents. Communications of the ACM. Vol. 37, No. 7, July 1994
3. Hermans, B.: Intelligent Software Agents on the Internet. http://www.hermans.org/agents/index.html
4. Magedanz, T., Rothermel, K., Krause, S.: Intelligent Agents: An Emerging Technology for Next Generation Telecommunications. Proc. INFOCOM'96, San-Francisco (1996)
5. Pham, V.A., Karmouch, A.: Mobile Software Agents: An Overview. IEEE Communications Magazine, pp. 26-37, July 1998
6. IBM Aglets Workbench, http://www.trl.ibm.co.jp/aglets/
7. Concordia, http://www.meitca.com/HSL/Projects/Concordia/
8. General Magic Odyssey, http://www.genmagic.com/agents/
9. Voyager, http://www.objectspace.com/voyager/
10. Jumping Beans, http://www.JumpingBeans.com/
11. OMG: The Common Object Request Broker Architecture and Specification. (1995)
12. JavaSpaces, http://java.sun.com/products/javaspaces
13. Rose, M.: The Simple Book - An Introduction to Management of TCP/IP-based Internets, 2nd Edition. Prentice-Hall International Inc. (1994)
14. ISO/IEC 9595: Information technology - Open Systems Interconnection - Common management information Service definition. International Organization for Standardization, International Electrotechnical Commission (1990)
15. Goldzmith, G., Yemini, Y.: Decentralizing Control and Intelligence in Network Management. Proceedings of 4th International Symposium on Integrated Network Management, Santa Barbara (1995)
16. Perpetuum Mobile Procura Project, Carlton University, http://www.sce.carleton.ca/netmanage/perpetum.shtml
17. Bieszczad, A.: Advanced Network Management in the Network Management Perpetuum Mobile Procura Project. Technical Report SCE-97-07, Systems and Computer Engineering, Carleton University (1997)
18. Lazar, S., Sidhu, D.: Discovery, A Mobile Agent Framework for Distributed Application Development, Technical Report, Maryland Center for Telecommunications Research, University of Maryland Baltimore County (1997)
19. Sahai, A., Morin, C.: Enabling a Mobile Network manager (MNM) Through Mobile Agents. Proceedings of Mobile Agents, Second International Workshop MA'98, Stuttgart, Germany (1998)
20. Nicklish, Quittek, J., Kind, A., Arao, S.: INCA: an Agent-based Network Control Architecture. Proceedings of IATA'98, Paris (1998)
21. Information Processing, Open Systems Interconnection: Specification of Basic Encoding Rules for Abstract Syntax Notation One. ISO (1987)

22. AdventNet SNMP, http://www.adventnet.com/products/snmpbeans
23. Wijnen, B., Carpenter, G., Curran, K., Sehgal, A., Waters, G.: Simple Network Management Protocol Distributed Protocol Interface Version 2.0, RFC 1592 (1994)
24. Susilo, G., Bieszczad, A. and Pagurek, B.: Infrastructure for Advanced Network Management based on Mobile Code. Proceedings of the IEEE/IFIP Network Operations and Management Symposium NOMS'98, New Orleans (1998)
25. Rose, M.: SNMP MUX protocol and MIB. RFC 1227 (1991)
26. Java Dynamic Management Kit, http//www.sun.com/software/java-dynamic
27. ITU-T Recommendation M.3010: Principles for a Telecommunications Management Network. (1992)