# Integrating SNMP into a Mobile Agent Infrastructure

Paulo Simões, Luis Moura Silva, and Fernando Boavida Fernandes

University of Coimbra, CISUC - Dep. Eng. Informática
Pólo II, Pinhal de Marrocos
P-3030 Coimbra, Portugal
{psimoes,luis,boavida}@dei.uc.pt

**Abstract.** Mobile Code is an emerging paradigm that is gaining momentum in several fields of application. Network Management is a potential area for the use of this technology, provided it will be able to interoperate with well established solutions for Network Management. This paper presents the integration a classic NM protocol, like SNMP, into a platform of Mobile Agents. Our platform, called JAMES, has been developed in the context of an Eureka Project ($\Sigma$!1921) where the project partners are University of Coimbra, Siemens SA and Siemens AG. Since the main target of the platform is network management, it includes a set of SNMP services allowing mobile agents to easily interface with SNMP agents, as well as with legacy SNMP-based management applications. In the paper we present a brief overview of the general architecture of the platform and we describe in some detail the framework we used to provide for integration between mobile agent applications and SNMP.

## 1 Introduction

Typically, existing Network Management (NM) applications are based on one of two protocols established almost a decade ago to address the problem of interoperability in heterogeneous environments: SNMP [1], for data networks, and CMIP [2], for telecommunication networks. These protocols are based on static and centralized client/server solutions, where every element of the network sends all the data to a central location that processes the whole data and provides the interface to the user operator. By this reason, the management applications are not flexible, have problems of scalability and produce too much traffic in the network.

It is now widely recognized that the use of decentralization in this kind of applications potentially solves most of the problems that exist in centralized client/server solutions [3, 4]. The applications can be more scalable, more robust, can be easily upgraded or customized and they reduce the traffic in the network.

Several approaches have been followed in this quest for decentralized management, starting with early work on Management by Delegation [5] and including Active Networks [6], management environments based on CORBA [7], Web-based network management [8], Intelligent Agents [4] and Mobile Agents

[9]. For an extensive typology of distributed network management paradigms, please refer to [10].

Of these approaches, Mobile Agent technology is one of the most promising but still requires some detailed study. A Mobile Agent corresponds to a small program that is able to migrate to some remote machine, where it can execute some function or collect some relevant data and then migrate to other machines in order to accomplish some other tasks. 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.

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 leading companies, including IBM, Oracle, Digital and Microsoft [11]. 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 [12–14].

Several commercial implementations of mobile agents have been presented in the market, including Aglets from IBM [15], Concordia from Mitsubishi [16], Odyssey from General Magic [17], Voyager from ObjectSpace [18] and Jumping Beans from AdAstra [19]. However, although these software products have some interesting features, we believe they are too much general-purpose for most NM applications.

In the `JAMES` project [20], 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 network management. Although the discussion of the issues we take into account is beyond the scope of this paper, we can point to aspects like efficient code migration; fault-tolerance and robustness; flexible code distribution and easy upgrading; mechanisms for resource control; disconnected operation; portability; and interoperability with existing management technologies.

This issue of interoperability with existing management technologies, like SNMP, CORBA and CMIP, is crucial for the success of mobile agents in the network management area. In order to insure the deployment of applications for the management of large and heterogeneous systems, mobile agent systems need to provide interoperability. The reasons are twofold.

First, those technologies provide access to management information and services. Mobile agents are a very attractive approach to incorporate mobile code into the existing local management services, in order to perform intelligent tasks closer to management data. However, they will not entirely replace the protocols used by classic management applications to interface with management services in heterogeneous environments. Instead, they will complement them with powerful programming constructs allowing more efficient solutions for network management. Nevertheless, some management protocols are still necessary to retrieve and process the management information. This is the case when some of the managed Network Elements (NEs) are unable to host mobile agents,

when the management services of NEs are not directly available to the hosted mobile agents or when the direct interfaces with management services are non-standardized.

Second, management applications based on mobile agents often need to coexist with legacy management systems. Mobile agents are particularly well suited to develop and deploy new network management services, but it seems much more attractive to use these services from installed management applications than to develop separate specialized applications. If these services provided using mobile agents include SNMP or CORBA interfaces, for instance, they may integrate into the legacy management systems at use. In this way, mobile agents can be introduced, in an incremental and integrated fashion, to solve specific problems for larger management frameworks still based on classic paradigms.

CORBA has been one of the first technologies to be addressed by mobile agent systems, which is not surprising given its usefulness in a wide variety of application fields. The most visible initiative in this area is the MASIF standard (Mobile Agent System Interoperability Specification [21]). Although MASIF is mainly a CORBA interface for interoperability between agent systems, its concepts that can also be used for interoperability with CORBA-based management applications and CORBA services. Right now only Grasshopper [22], from IKV++, claims to be MASIF compliant, but other platforms that already provide CORBA access to and from external objects are expected to follow soon.

Integration between mobile agents and classic NM paradigms, like SNMP, was not given the same level of attention. Few platforms provide some degree of support for their usage, and even fewer include a well-defined framework for interoperability with classic management architectures.

One can argue that integration of classic 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.

The SNMP support that is offered by the `JAMES` platform might be used whenever it is necessary to interact with SNMP agents (local or remote) or to provide SNMP services to legacy applications. It includes support for agent mobility and it can be dynamically installed or removed from the platform. Furthermore, it is transparent to SNMP devices and SNMP Managers, not requiring any kind of specific adaptation from them.

In this paper, we will describe how we have done the integration of SNMP into our system of mobile agents, in order to provide this support for interoperability. The rest of the paper is organized as follows: Section 2 discusses related work and some of the design premises that lead to the `JAMES` design. Section 3 presents a brief overview of the `JAMES` Platform and Section 4 describes the design and functionality of the SNMP modules of `JAMES`. Section 5 concludes the paper.

## 2 Related Work

SNMP is not directly supported by any of the commercial implementations of mobile agents. However, there are a few ongoing research projects mixing SNMP with mobile agents.

The INCA Architecture [23], from NEC CCRLE-Berlin, provides access to SNMP devices based on a common Java SNMP library. Unfortunately, the lack of available documentation restricted us from an accurate analysis of this proposal.

The Discovery Platform [24], from the University of Maryland, proposes the use of the SNMP information model to represent the internal knowledge base of the platform. In this way, the host management information and the platform management information are unified in a single MIB tree, accessible directly from the mobile agents platform. This approach claims a few advantages: the agent information base and the host management information can be accessed using the same mechanisms and it is possible to add any object with arbitrary complexity to the tree. However, it requires specific support from the native SNMP agent residing at the local host and assumes that SNMP is always the most adequate protocol to access local management information. It also ties the agent management information with the SNMP functional limitations. Furthermore, it does not address the problem of accessing external SNMP agents, which continues to require a separate SNMP engine.

The Astrolog/MAGENTA platform [25], from IRISA, employs mobile agents to support the mobility of the network operator. However, the core NM system is based on a hierarchy of stationary agents. The mobile agents execution environments, designated as *lieus*, include a built-in SNMP agent available directly to local mobile agents as well as to the remote applications that use SNMP. Apparently, these SNMP agents do not support dynamic expansion, which means offered services are monolithic. This approach also affects the system portability: since SNMP agents are internal to *lieus*, it is necessary to adapt the *lieu* implementation for each type of managed NE, reproducing functionality that was, quite probably, already provided by native SNMP agents.

The Perpetuum Mobile Procura (PMP) Project [26, 27], from the Carleton University, proposes the use of DPI (SNMP Distributed Protocol Interface [28]) as a means to talk with local SNMP agents whenever there is not a more sophisticated interface with local resources. DPI is also used to extend SNMP agents through the download of Java classes. This is one of the few projects where SNMP is used both to access local management services and to interface with legacy management applications. However, the need for DPI support from the native SNMP agent affects the system portability.

All these projects provide interesting features. However, none of them is completely satisfactory. INCA doesnt seem to address interoperability with SNMP applications. Discovery and Astrolog do provide higher levels of interoperability but their design is too committed with SNMP and requires explicit support from SNMP devices, which affects portability. The PMP architecture is more independent (SNMP is seen as an optional feature) but usage of SNMP depends on DPI, which also affects portability.

In the JAMES project we considered a novel approach for SNMP support, with three key goals in mind.

First, we also want to provide operability with both SNMP devices and applications, in order to keep a large set of potential application fields.

Second, we see SNMP just as another tool to be added to the Swiss Army knife that mobile agents may use to solve management problems. Some problems require SNMP, others need to use CORBA or proprietary interfaces to interface with NEs, while for others it is sufficient to access relational databases (using JDBC) or even raw text files. This means that solutions where SNMP would affect the platform portability or functionality were not acceptable.

Third, we want SNMP support to be transparent to existing SNMP devices and applications. Any requirement of changes to SNMP devices or applications would jeopardize one of the main reasons to use SNMP: the fact that it is immediately available almost anywhere.

These premises led to the design of a new framework for SNMP integration that:

- provides maximum interoperability, covering three different service ranges: interaction with local and remote SNMP-agents; interaction between SNMP-managers and mobile agents; and infrastructure management using SNMP,
- supports agents mobility - usage of SNMP does not restrict the migration of mobile agents,
- is an optional feature of JAMES, not imposing additional overheads to the platform when turned-off,
- is non-intrusive to the SNMP architecture, in the sense that no intervention is required on existing SNMP devices and SNMP Managers. This preserves the overall portability.

## 3 The General 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 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 provides 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 mobile agents will be used.

Every NE runs a Java Virtual Machine and executes a JAMES Agency that enables the execution of mobile agents. JAMES agents will migrate through these machines in the network to access some data, execute some tasks and produce reports that will be sent back to the JAMES Manager. There is a mechanism for 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

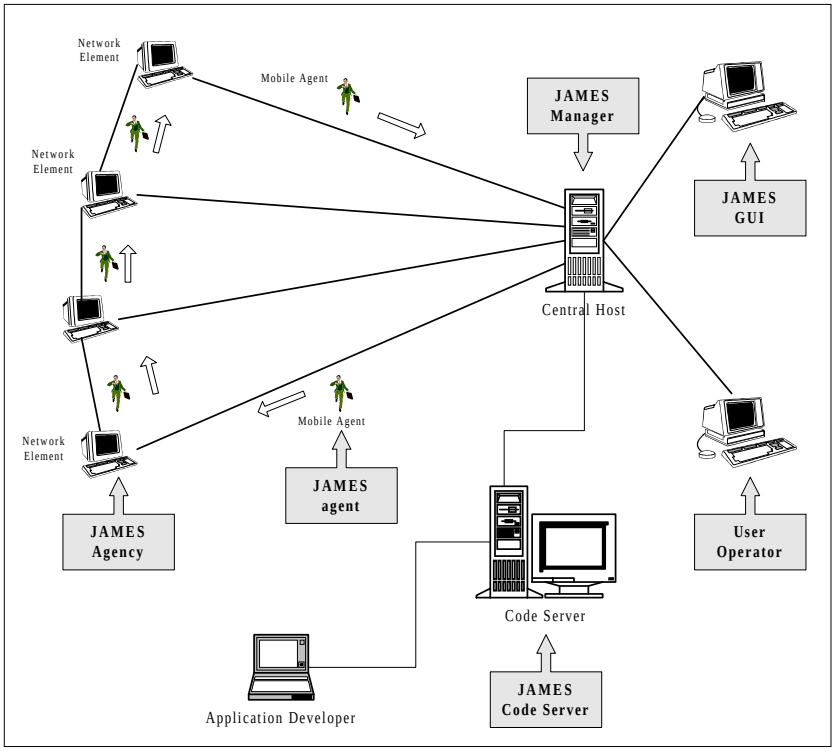


**Fig. 1.** An Overview of the JAMES Platform

developed to transfer the agents across the machines in a robust way and is atomic 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 (Java Archive) 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 JAR files available and the mapping to the corresponding mobile agents. The communication between the client program that is used by the application developer and the Code Server is done by the HTTP protocol. The Code Server interacts with the `JAMES` Manager through a dedicated stream socket.

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. Management applications using mobile agents can bypass this end-user interface using a Remote API. This interface includes

tools to manage all the Agents and Agencies in the system, and it is available in two different communication methods: Java RMI and CORBA.

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,
- Remote Access to the platform through a Java RMI interface,
- Remote Access to the platform through a CORBA interface,
- CORBA access to and from external objects,
- Integration of Java-based SNMP services into the platform,
- Inter-agent communication (through JavaSpaces [29]),
- Multi-paradigms for agent execution (simple agent, migratory agents and Master/Worker model).

The explanation of all these mechanisms is out of scope of this paper, but we will give some emphasis to a few of the issues considered important for our domain of applications: remote software upgrading, reliability and robustness.

### 3.1 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.

Two modules compose the JAMES Agency: a small `rexec` daemon and the Agency itself. The `rexec` 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 `rexec` daemon (JREXEC) 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.

### 3.2 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 is 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 will be 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 has 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 will make 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 insures 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 either 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 a migration protocol or 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 up and running.

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

### 3.3 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 clearly more robust and this is a crucial step if we want to use it in production codes.

## 4 Integration of SNMP into the JAMES Platform

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. In order to preserve the platform portability, these services have been written in Java. As represented in Fig. 2, three basic SNMP services are 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.

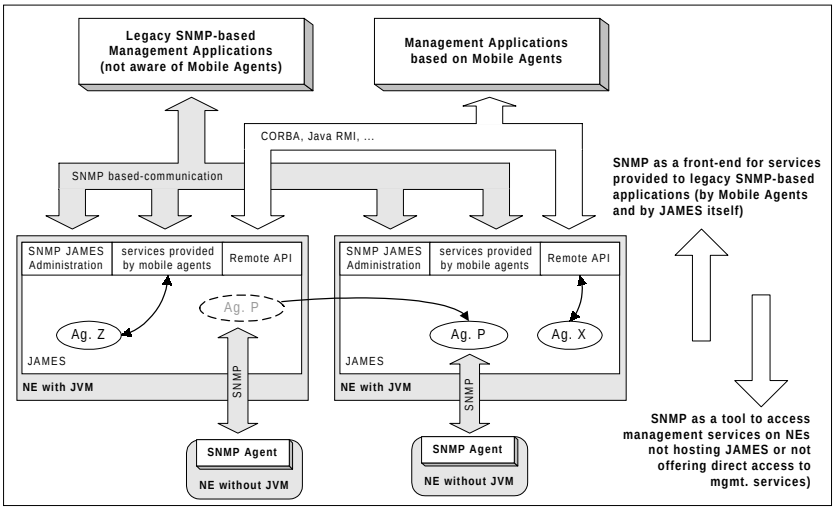


**Fig. 2.** Proposed Integration Framework

### 4.1 Design of SNMP Services

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

- SNMP support must be optional, not increasing resource usage when turned-off.
- SNMP support may not increase the complexity of the platform.
- The 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 issues resulted in a modular design (Fig. 3) 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 in the `JAMES` platform.

### 4.2 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

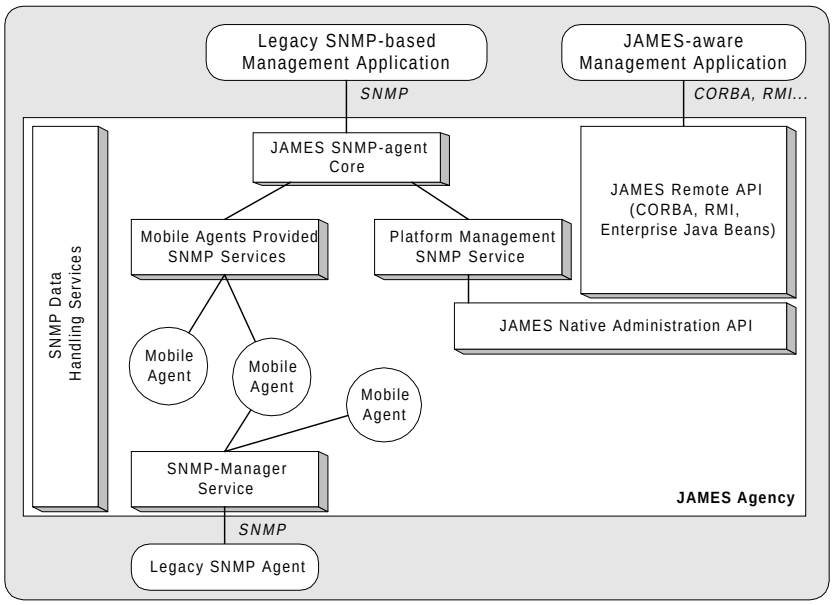


**Fig. 3.** High-Level Structure of SNMP Services of JAMES

representation of SNMP data types and PDUs. A set of Java-based ASN.1/BER [30] encoding methods is also available to be used by other SNMP Services. Mobile agents impose no particular requirements to this service, meaning general-purpose Java-based SNMP tools, like [31], could have been used without prior adaptation.

Presentation of the internal information of the platform, as suggested by [24], was not considered because the SNMP information model was considered too poor for that purpose.

### 4.3 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 found in most high-level SNMP stacks (sessions or contexts, request operations and event handlers), with protocol details being transparently handled. This Service, located

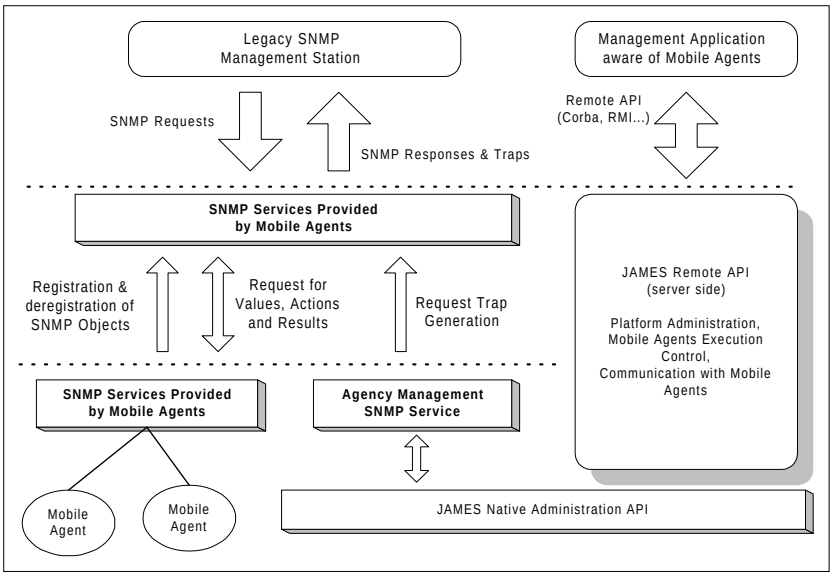


**Fig. 4.** JAMES Services for Integration with Management Applications

in Service Agents, might be replaced with a third-party classic SNMP stack integrated in the Agents code, trading-off mobility support.

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

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 agents 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.

### 4.4 Services for Interoperability with Legacy SNMP Managers

While the Service described in the previous section covers communication between mobile agents and SNMP agents, there are three other services providing interoperability with legacy SNMP Managers (Fig. 4). 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 adaptation of standard protocols for expansible SNMP agents like DPI is not excluded.

The JAMES SNMP-agent is independent of eventually existing native SNMP-agents although integration, as proposed by [32], would be more in the spirit of SNMP[2]. Since native agents are either monolithic or based on a wide diversity of agent-expansion mechanisms, like SMUX [33], DPI or AgentX [34], 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 Java Dynamic Management Toolkit (JDMK) [35], a commercial product from Sun, is not based on mobile agents but shares the vision of a flexible scheme to build new management services. JDMK is meant for fast development and deployment of Java-based management services, including a complete set of tools to create and remotely install these services using Java and push/pull techniques[3]. It provides a sophisticated set of development tools and includes support for SNMP, CORBA (IIOP), HTTP and Java RMI. The PMP Project 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 4). The intention is not to use SNMP to fully administer JAMES but to provide a basic set of SNMP-based management services.

It should be stressed that the option of separating the SNMP services provided by the JAMES mobile agents from the native SNMP-agents, although jus-

[2] i.e., it is preferred to maintain two separate agents answering in different ports than to try integration of both agents into a single one

[3] This means JDMK management services do not have support for full mobility across different hosts.

tified for sake of portability, imposes a small constraint when compared to more integrated solutions, like the one of the PMP project. It is not possible, for instance, to extend a MIB of the native SNMP-agent. This is not problematic since the most usual use of mobile agents, in this context, will be the provision of new services and not a very localized expansion of existing SNMP Services. That is considered outside our scope and more appropriate for products like JDMK.

## 5 Conclusions

The JAMES Project exploits the paradigm of mobile agents in the field of network management. 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 support for CORBA and SNMP. Within this project we expect to show that Mobile Agents can overcome some of the problems that exist with traditional Client/Server solutions.

During this project, the platform will be used in software products in the area of network management. Our industrial partners (Siemens S.A.) have been developing prototype applications in the area of performance management that use our platform of mobile agents. These prototypes are already finished and we are now conducting a benchmarking study to compare the use of mobile agents over traditional client/server solutions to see if we corroborate some of the advantages of this new paradigm in the field of distributed computing.

In this paper we discussed the provision of explicit SNMP support for management applications based on mobile agents. This kind of support is important whenever SNMP is the only available interface to access management information. Another reason to provide SNMP support is the possibility of using mobile agent technology to develop new management services to be used by legacy management applications. However, integration of SNMP into mobile agent systems has not received as much attention as CORBA-based interoperability and the number of mobile agent platforms providing such integration is still very reduced.

The framework in the JAMES project for the integration of SNMP differs from previous work because a great deal of attention is given to keep it transparent for SNMP devices and applications. Another distinctive issue of our approach is the fact that SNMP support is dynamically installable and removable, not affecting the platforms functionality or complexity when not being used.

## Acknowledgements

This project was accepted in the European Eureka Program ($\Sigma$!1921) and it is partially supported by ADI (*Agência de Inovacão*) and FCT (*Fundacão para a Ciência e Tecnologia*). Special thanks to Rodrigo Reis, for the help in the development of several SNMP services, and to the rest of the project team from University of Coimbra and Siemens.

## References


1. Rose M.: The Simple Book - An Introduction to Management of TCP/IP-based Internets, 2nd Edition. Prentice-Hall International Inc. (1994)
2. ISO/IEC: ISO/IEC 9595: Information technology - Open Systems Interconnection - Common Management Information Service Definition. International Organization for Standardization, International Electrotechnical Commission (1990)
3. Goldszmidt, G., Yemini, Y.: Decentralizing Control and Intelligence in Network Management. Proceedings of the 4 th International Symposium on Integrated Network Management, Santa Barbara (1995)
4. Magedanz, T., Rothermel, K., Krause, S.: Intelligent Agents: An Emerging Technology for Next Generation Telecommunications. Proceedings of INFOCOM96, San Francisco, CA (1996)
5. Yemini, Y., Goldszmidt, G., Yemini, S.: Network Management by Delegation. Proceedings of IFIP 2 nd International Symposium on Integrated Network Management, Washington (1991)
6. Tennenhouse, D., Smith, J., Sincoskie, W., Wetherall, D., Minden, G.: A Survey of Active Network Research. IEEE Communications Magazine (1997)
7. OMG: The Common Object Request Broker Architecture and Specification. (1995)
8. Wellens, C., Auerbach, K.: Towards Useful Management. The Simple Times, Volume 4, Number 3 (1996)
9. Bieszcad, A., Pagurek, B., White, T.: Mobile Agents for Network Management. IEEE Communications Surveys, 4Q (1998)
10. Martin-Flatin, J., Znaty, S.: Annotated Typology of Distributed Network Management Paradigms. Technical Report SSC/1997/008, cole Polytechnique Fdrale de Lausanne (1997)
11. Agent Product and Research Activities. `http://www.agent.org/pub/activity.html`
12. Intelligent Agents. Communications of the ACM, Vol. 37, No. 7 (1994)
13. Hermans, B.: Intelligent Software Agents on the Internet. `http://www.hermans.org/agents/index.html`
14. Pham, V., Karmouch. A.: Mobile Software Agents: An Overview. IEEE Communications Magazine, pp. 26-37 July (1998)
15. IBM Aglets Workbench. `http://www.trl.ibm.co.jp/aglets/`
16. Concordia. `http://www.meitca.com/HSL/Projects/Concordia/`
17. General Magic Odyssey. `http://www.genmagic.com/agents/`
18. Voyager. `http://www.objectspace.com/voyager/`
19. Jumping Beans. `http://www.JumpingBeans.com/`
20. Silva, L., Simes, P., Soares, G., Martins, P., Batista, V., Renato, C., Almeida, L., Stohr, N.: JAMES: A Platform of Mobile Agents for the Management of Telecommunication Networks. Proceedings of IATA'99 (3rd International Workshop on Intelligent Agents for Telecommunication Applications), Stockholm (1999)
21. Mobile Agent System Interoperability Facilities Specification. OMG TC Document orbos/97-10-05 (1998)
22. Grasshopper. `http://www.ikv.de/products/grasshopper/`
23. Nicklish, J., Quittek, J., Kind, A., Arao, S.: INCA: an Agent-based Network Control Architecture. Proceedings of IATA'98 (2nd International Workshop on Intelligent Agents for Telecommunication Applications), Paris (1998)
24. 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)

25. Sahai, A., Morin, C.: Enabling a Mobile Network manager (MNM) Through Mobile Agents. Proceedings of Mobile Agents, Second International Workshop MA98, Stuttgart, Germany (1998)
26. Perpetuum Mobile Procura Project. Carlton University, http://www.sce.carleton.ca/netmanage/perpetum.shtml
27. 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)
28. Wijnen, B., Carpenter, G., Curran, K., Sehgal A., Waters, G.: Simple Network Management Protocol Distributed Protocol Interface Version 2.0, RFC 1592 (1994)
29. JavaSpaces. http://java.sun.com/products/javaspaces
30. Information Processing, Open Systems Interconnection: Specification of Basic Encoding Rules for Abstract Syntax Notation One. ISO (1987)
31. AdventNet SNMP. http://www.adventnet.com/products/snmpbeans
32. 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)
33. Rose, M.: SNMP MUX protocol and MIB. RFC 1227 (1991)
34. Daniele, M., Wijnen, B., Francisco, D.: Agent Extensibility (AgentX) Protocol Version 1, RFC 2257 (1998)
35. Java Dynamic Management Kit. http//www.sun.com/software/java-dynamic