Towards Manageable Mobile Agent Infrastructures

Paulo Simões, Paulo Marques, Luis Silva, João Silva, and Fernando Boavida

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

Abstract. This paper addresses the problem of managing distributed mobile agent infrastructures. First, the weaknesses of current mobile agent implementations will be discussed and identified from the manageability viewpoint. The solutions devised and experimented in order to alleviate these weaknesses in our own agent platform will then be presented. These solutions are generic and could easily be applied to the majority of existing mobile agent implementations. The paper will finish with the discussion of a new approach we are following in the `M&M` Project, based on a rather different architecture that significantly reduces the manageability requirements.

1 Introduction

A mobile agent (MA) is a software program that is able to migrate to some remote machine, where it is able to execute some function or collect some relevant data and then migrate 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. MA systems differ from other mobile code and agent-based technologies because increased code and state mobility allow for even more flexible and dynamic solutions. Telecommunication applications and network management are part of the broad range of application fields for MA systems [1].

Mobile agent implementations are based on some kind of distributed supporting infrastructure, that provides the execution environment for the agents in each location, controls the agent migration and lifecycle, and usually delivers additional services such as security and multi-agent coordination. This infrastructure typically consists of mobile agent platforms designed as extensions of the host's operating system. Each network node hosts one platform where agents from different applications and different users coexist and (hopefully) cooperate with each other.

One of the problems with current mobile agent systems is the considerable overhead required to properly install, configure and manage this large, distributed and remote infrastructure. This overhead often counterbalances the advantages of applying mobile agents in the first place, and seriously affects the usability of MA technology. For this reason, effective infrastructure management solutions are crucial for the success of mobile agent systems.

2 Agent Management and Infrastructure Administration

Manageability of mobile agent systems includes two distinct domains: the mobile agents themselves, and the supporting infrastructure.

Agent management is by now fairly well covered. Almost every known MA implementation allows adequate control of the agent lifecycle, the agent location, inter-agent coordination, migration and security. Although current implementations rely on proprietary and internal mechanisms, ongoing standardization initiatives - like MASIF [2] and, to some extension, FIPA [3] - are expected to provide the desired means of interoperability, as more implementations start to comply with these standards.

The management of the supporting distributed infrastructure, on the other way, is usually considered as a side-problem not directly related with MA technology and, therefore, not deserving the same level of attention. Although this might be so for small-scale prototypes, when one tries to deploy real world distributed MA platforms, the costs of installation and administration easily rise to unacceptable levels.

In the last couple of years we have developed the `JAMES` platform for mobile agents [4, 5], which was later used by our industrial partners to produce and deploy several MA-based applications for telecommunications and network management. This scenario provided us with a good perspective on infrastructure manageability. Three key problems were identified in current systems: (i) they are too much focused on mobile agents; (ii) they do not decouple the management functionality from the infrastructure itself; and (iii) they lack effective support for remote maintenance of geographically distributed platforms.

The excessive focus on mobile agents results in the assumption that applications are 100% based on mobile agents. However, most applications should be composed by a mix of "static" modules (user interfaces, databases, system specific modules, etc.) and, just where appropriate, specialized mobile agents. This excessive focus on the MA technology, instead of the overall MA-based application, leads to a general lack of clear and powerful interfaces between the mobile agent infrastructure, the agents and the "static" components of agent-based applications. This results either in complicated and inefficient ad-hoc interfaces between the agents and the applications and in unbalanced application design, with more and more functionality pushed into mobile agents just because there is no easy way to interface with external modules.

The lack of decoupling between the management functionality and the platform itself is the second major drawback with current systems. Although they already entail some kind of global infrastructure management, with basic monitoring and control, this functionality is hidden in the platform internals and, at most, only available through closely attached user interfaces. General-purpose management applications are thus unable to integrate the management of MA infrastructures into its global system management framework.

The remote management factor is also an important but overlooked issue. One of the key advantages of MA systems is the ability to dynamically upgrade remote services installed on mobile agents. However, this advantage is dependent on the robustness of the underlying agent platform, which demands support for remote operations like installation, upgrading, monitoring, rejuvenation and tuning. With few exceptions, current platforms do not provide such services and require either local installation and maintenance (with increased running costs) or the usage of general remote desktop applications like Microsoft's Systems Management Server or Intel's Landesk, with increased complexity and poor integration.

3 Platform Manageability in the JAMES Project

In order to tackle with these key issues, three simple but effective solutions were introduced in the `JAMES` platform: a low-level service for remote upgrade and control of MA platforms; a high-level API for external applications interfacing with the platform or the platform's mobile agents; and an SNMP service that allows legacy management applications to monitor and administer the MA infrastructure. Fig. 1 shows the integration of these three solutions into the `JAMES` framework. Their implementation uses very simple and pragmatic approaches, and other MA systems should also be able to support this management model without requiring major system redesign.

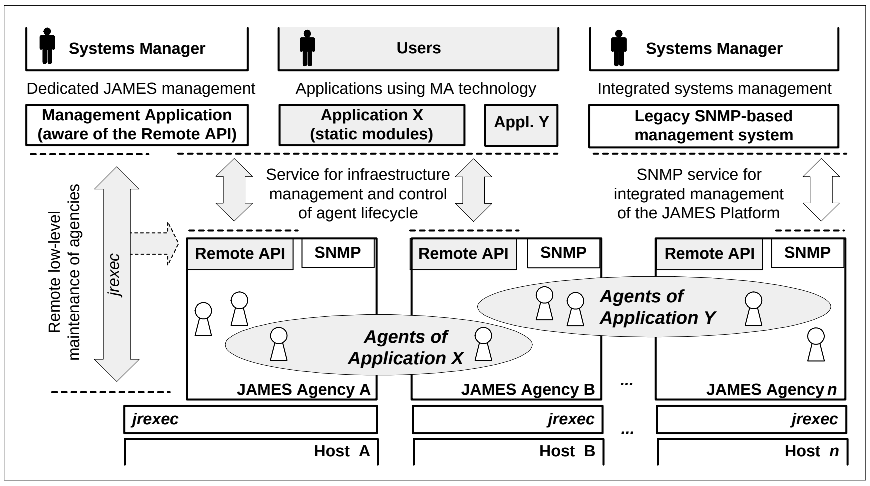
Systems Manager
Users
Systems Manager
Dedicated JAMES management
Applications using MA technology
Integrated systems management
Management Application (aware of the Remote API)
Application X (static modules)
Appl. Y
Legacy SNMP-based management system
Service for infraestructure management and control of agent lifecycle
SNMP service for integrated management of the JAMES Platform
Remote low-level maintenance of agencies
jrexec
Remote API
SNMP
Agents of Application X
Agents of Application Y
JAMES Agency A
JAMES Agency B
JAMES Agency n
...
jrexec
Host A
Host B
Host n

Fig. 1. `JAMES` Threefold Support for Platform Management

3.1 Low-level Management of Remote Agent Platforms

The lower level of management for remote agent platforms (i.e. agencies) is provided by *jrexec*, a small and very stable service that runs at the host operating system level and controls the execution of the MA platform providing several services from remote locations: agency start and stop, agency rejuvenation, agency monitoring and upgrade of the agency software. This service represents a hook that avoids expensive local interventions, even in the case of unexpected MA platform crashes.

The installation and operation of *jrexec* is dependent of the host's operating system. It might be installed, for instance, as an Windows NT service or a Unix daemon. However, the interface to the management services provided by the several versions of *jrexec* is homogeneous, resulting in uniform MA infrastructure management across heterogenous networks. Since it is even possible to run the agencies without *jrexec* (relying in general-purpose remote management tools or using local interventions whenever necessary) the portability of the `JAMES` platform is not affected.

3.2 Remote Interface for External Applications

`JAMES` includes a single unified interface for communication between the mobile agent system and external applications. The implications of this interface, that we name Remote API, are manifold:

– the core of the platform became much simpler. Several user interfaces for basic operations and maintenance tasks (system monitoring, agent lifecycle control, system maintenance), previously attached to the platform's core, were pushed to the outside and redesigned as standalone applications that interact with the platform using the Remote API. In fact the current version of the platform has no graphical user interface at all;

– interaction between mobile agents and its associated "static" modules (applications) was enhanced. Each application directly launches, controls and communicates with its agents in a straightforward fashion. This simplifies the application design and allows more efficient usage of agent technology;

– and MA technology can be totally hidden bellow MA-based applications that, through the Remote API, completely control and make use of `JAMES` without requiring direct contact between the user (i.e. the customer) and MA related technology. This opens the way for deployment of MA-based applications into mainstream markets.

The current implementation of the Remote API is based on Java RMI, and consists on the interface itself, on the side of platform, and a set of Java classes to be included on the external applications (Fig. 2). These classes provide to the application developer with a high-level interface to the agent infrastructure, either to control/access their own agents or to manage the whole infrastructure.

The administrative model of the Remote API considers several types of entities: Applications, Application Licenses, Users, Mobile Agents, Instances of

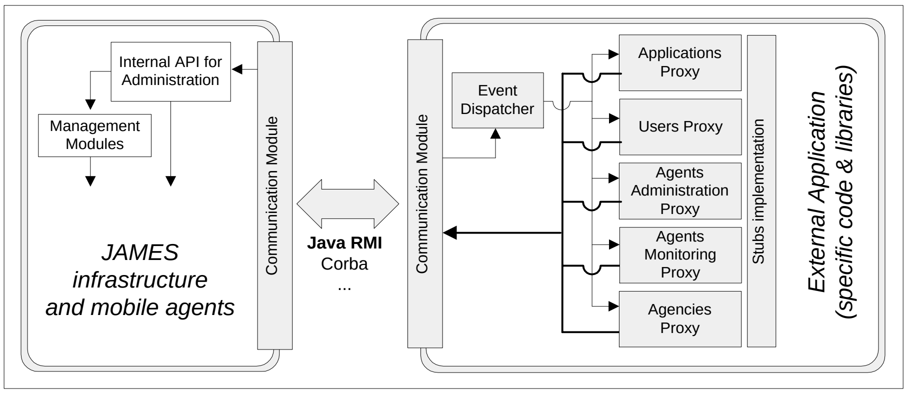

Fig. 2. Structure of the Remote API

Mobile Agents and Agencies (Fig. 3). With this model it is possible to support coexistence of multiple applications and multiple users in a flexible manner. A detailed security framework also allows the flexible definition of the permissions of each entity. It should be noted that the Remote API imposes no clear distinction between infrastructure management and common usage of mobile agents. The role played by each external application connected to the `JAMES` platform only depends of its security permissions and the context of each request. The

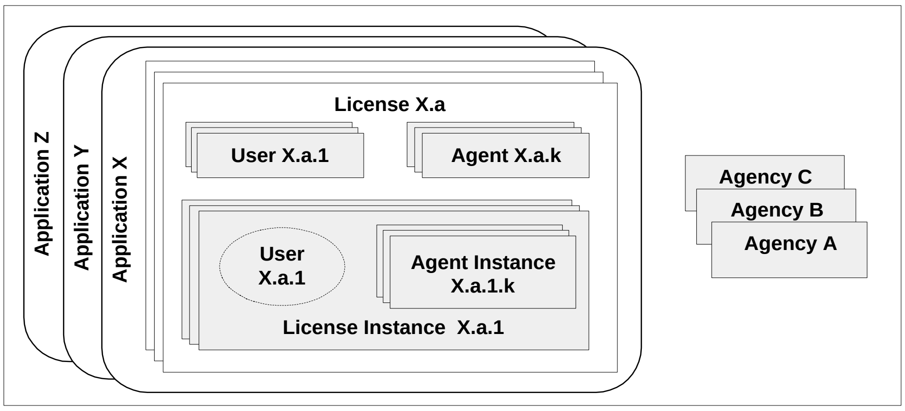

Fig. 3. Administrative Model of the Remote API

Remote API presents some similarities with MASIF [2], a CORBA-based interface proposed by OMG as a standard for interoperability between different mobile agent systems. However, while MASIF focus on the mobile agent lifecycle, providing services for agent instantiation, control and monitoring, the Remote

API also addresses the infrastructure management, including a more elaborate administrative model and also encompassing the maintenance of the agencies. The Remote API could probably be designed as an extension of MASIF, since its functionality is a superset of the functionality covered by MASIF. However, this decision would imply the simplification of the `JAMES` administrative model (in order to comply with the model of MASIF) without major interoperability gains, since the acceptance of MASIF is still reduced.

3.3 Platform Management From Legacy Applications

The third management service included in the `JAMES` platform is motivated by the desire to provide integration between MA systems and legacy management applications based in architectures like SNMP [6]. With this goal in mind a subset of the functionality provided by the Remote API was "translated" into an SNMP MIB (see Fig. 4) and made available through an AgentX-based extensible agent [7]. In this way the mobile agent infrastructure can be treated by general-purpose legacy management platforms like any other component of the whole system. It should be noted, however, that due to the intrinsic limitations of the SNMP framework this service does not provide the same level of functionality as the Remote API, with its rich programming model and detailed security framework. Instead, the JAMES-MIB acts as a complement directed to legacy management applications that can not be converted to use the Remote API.

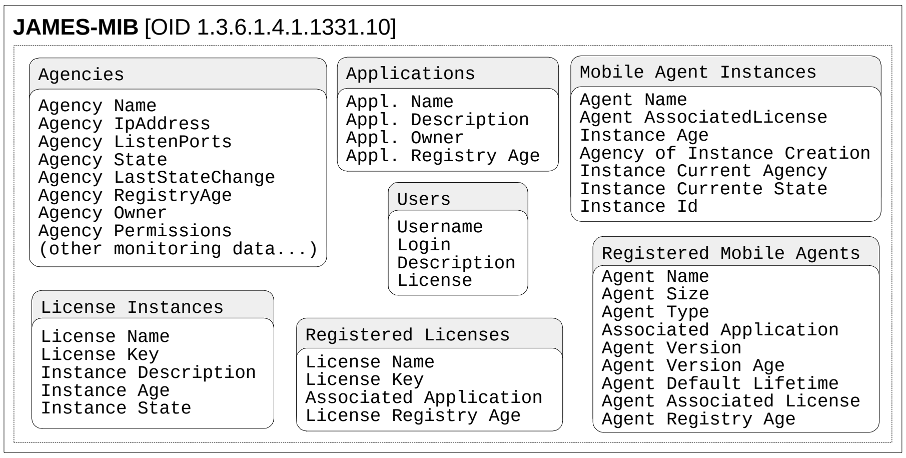

Fig. 4. Main Components of the JAMES-MIB

4 One Step Further: Maintenance of Application-Centric Mobile Agents

As already mentioned, the solutions devised for the JAMES platform tackle with some of the identified problems with the maintenance of MA systems simply by adding ad-hoc management support to an already established architecture, common to most MA implementations. In the context of the M&M project [8] we are now working on a new architecture for mobile agent support. The most distinctive characteristic is that in this new approach there are no agent platforms. Instead, agents arrive and leave from the applications they are part of. The application is central and MAs are just a part of the system playing specific roles (see Fig. 5). The applications are able of sending, receiving and interacting with mobile agents by using well-defined binary software components (Javabeans or ActiveX components). Management applications are developed using the current industry best-practice software methods and can become agent-enabled by integrating mobility components. There is one small component that provides the basic support for mobile agents (migration, lifecycle control) and, using a flexible extension mechanism, more sophisticated services (agent coordination and communication, agent tracking, security, persistence, infrastructure management, etc.) are added as components, if and where needed.

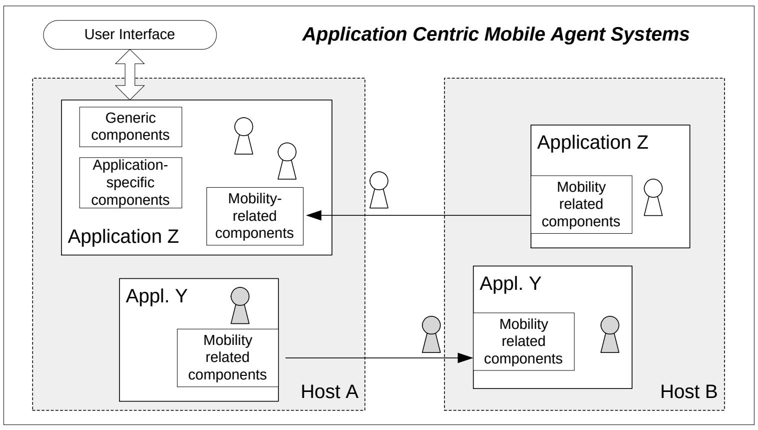
User Interface
Application Centric Mobile Agent Systems
Generic components
Application-specific components
Mobility-related components
Application Z
Appl. Y
Mobility related components
Host A
Application Z
Mobility related components
Appl. Y
Mobility related components
Host B

Fig. 5. The M&M Architecture for Application Centric Mobile Agents

A more detailed description of this novel perspective on mobile agents, that brings potential advantages in several fields, is presented in [9, 10]. From a strict manageability viewpoint, the key differences are:

– there is no agent platform to install and maintain. Although there are still distributed applications to install and manage, this is much simpler than managing a separate infrastructure shared by a large number of distributed applications with different policies and requirements;

- agents interact directly with the applications from the inside. This eliminates the need to set up interface agents and configure and manage its security policies. This also partially eliminates the need for mechanisms like the Remote API;
- for each application only the required components (including even the management services) are installed, resulting in a simpler and lighter framework to manage.

This framework results, therefore, in a significant reduction on the specific infrastructure management requirements and in a shift towards the use of more generic application management tools. The manageability problem becomes less complex and more generic. Nevertheless, some agent and infrastructure management services are available in the form of extension service components.

These services provide an additional service layer for managing the agent support components, available both from within the agent-enabled application and from external applications (see Fig. 6). Available functionality includes agent management (instantiation, monitoring, shutdown, etc.) and mobility component control (start/shutdown, monitoring, configuration, resource management, etc.). Additional services, such as agent tracking, are also available through similar service components. This architecture results in a simple but very flexible

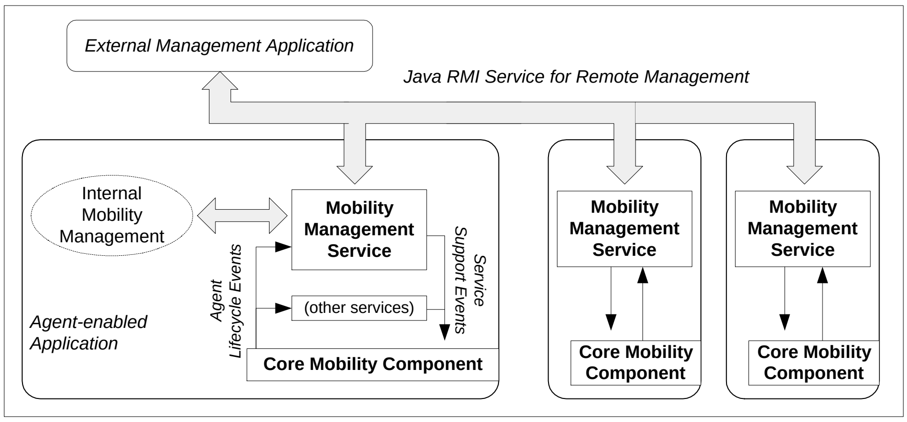
External Management Application
Java RMI Service for Remote Management
Internal Mobility Management
Mobility Management Service
Agent Lifecycle Events
Service Support Events
(other services)
Agent-enabled Application
Core Mobility Component
Mobility Management Service
Core Mobility Component
Mobility Management Service
Core Mobility Component

Fig. 6. `M&M` Management Architecture

framework for maintenance of the mobility-related components of the application. New interfaces (such as SNMP, Corba, WBEM or HTML) can easily be added building new components, and the management functionality can even be extended to entail the administration of the whole agent-enabled application. The implementation of the `M&M` framework is partially available for download at [8], including a prototype external management application for demonstration purposes.

One important point is that the component infrastructure must be manageable itself. For instance, some applications require runtime instantiation or installation of new services. The need to change certain component parameters at runtime (e.g. the maximum number of running agents, a listening port, or the logging level) is also quite common. In our case this is accomplished by having a management component service that performs several functions:

- it listens for external management requests;
- when a service instantiation is requested it analyses the Service Deployment Descriptor that was issued and then instantiates and configures the service. This involves close interaction with the core Mobility Component for registering the service with the appropriate configuration and security permissions;
- when it receives a request to modify a component parameter it propagates the request to that component. A unique identifier distinguishes each component, and each component understands certain administration tasks. For instance, the Mobility Component has a property named *listen port*, which represents the port for the incoming agents. A request can be made to change the *mm.mob.Mobility.listenPort* property to a new value. The request first arrives at the management component, which identifies the target component and then propagates the request to that component through a callback interface;
- finally, the management component also allows to broadcast a property change request to all the components. For instance, most of the components use a property called *mm.mob.logLevel* that determines which calls and actions must be sent to the log file. It is quite easy to request all the running components to change their log level by issuing a broadcast that will notify each of the components of the change.

Despite its simplicity, this management component provides a high degree of flexibility. Its strength resides in its generality, that allows any changes to be propagated to the running components, even when those components were not known at the time of development or deployment. The management interface of each of the components is not hard-coded in a class but propagated to the components themselves. Each component includes and understands its specific set of management properties and actions.

Another interesting point of the framework is that it allows the runtime deployment of new services not only using the management interface but also through administration agents. This means that an agent with the appropriate permissions has the ability to examine which components are available at a host, to modify their properties, and to change that configuration by shutting down services or instantiating new ones. This feature is especially interesting in applications where monitor agents roam through the network examining the state of the machines, quickly reacting to malfunctions (e.g. crashes, failures and environment changes) by autonomously recovering the applications.

5 Conclusions

The high costs associated with the installation and administration of distributed mobile agent infrastructures are an important but often overlooked obstacle to widespread deployment of mobile agents. In this paper we present two approaches to reduce those costs. In the first approach, validated in the JAMES project, several ad-hoc solutions are added to a classic platform-based architecture, resulting in enhanced communication with external applications, integration with legacy SNMP-based management applications and remote control of the distributed infrastructure. The second approach is based on the `M&M` framework, where a significant change in the agent-system architecture results in much simpler management requirements. Using the `M&M` integration framework, a small management service component is available both for internal and external management of the agent-support components. In the future new interfaces can be added - for instance for SNMP support - and new management functionality can be provided, such as application management.

Acknowledgements

The `M&M` project is partially funded by CISUC (R&D Unit 326/97) and by FCT (Project Reference POSI/33596/CHS1999). `JAMES` was an Eureka Project (Reference Σ!1921) partially funded by ADI (*Agência de Inovacão*).

References

1. Pham, V., Karmouch, A.: Mobile Software Agents: An Overview. IEEE Communications Magazine, pp. 26-37, July (1998)
2. Mobile Agent System Interoperability Facilities Specification. OMG TC Document orbos/97-10-05 (1998)
3. Foundation for Intelligent Physical Agents, `http://www.fipa.org/`
4. Silva, L., Simões, 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, Springer-Verlag LNCS 1699 (1999)
5. University of Coimbra, `JAMES` Project Homepage, `http://james.dei.uc.pt/`
6. Simões, P., Silva, L., Boavida, F.: Integrating SNMP into a Mobile Agents Infrastructure. Proceedings of DSOM'99, Springer-Verlag LNCS 1700 (1999)
7. Simões, P., Lourenco, E., Pereira, P., Silva, L., Boavida, F.: J.AgentX: a Tool for Dynamic Deployment of Open Management Services. Proceedings of 2000 International Conference on Software, Telecommunications and Computer Networks (SoftCOM'2000), Split (2000)
8. University of Coimbra, `M&M` Project Homepage, `http://mm.dei.uc.pt/`
9. Marques, P., Silva, L., Silva, J.: Going Beyond Mobile Agent Platforms: Component-Based Development of Mobile Agent Systems. Proceedings of the 4th International Conference on Software Engineering and Applications (SEA'2000), Las Vegas (2000)
10. Marques, P., Simões, P., Silva, L., Boavida, F., Silva, J.: Providing Applications with Mobile Agent Technology. Proceedings of the 4th IEEE International Conference on Open Architectures and Network Programming (OpenArch'01), Anchorage (2001)