Using Mobile Agents for Network Performance Management

C. Bohoris, G. Pavlou, H. Cruickshank
Center for Communication Systems Research
School of Electronic Engineering and Information Technology
University of Surrey, Guildford, Surrey GU2 5XH, UK
{C.Bohoris, G.Pavlou, H.Cruickshank}@ee.surrey.ac.uk

Abstract

Mobile agent frameworks have attracted a lot of attention in recent years, seen as counterparts of static distributed object frameworks but allowing also for object or agent mobility. A lot of research is currently being carried out trying to assess their applicability to network management and control environments. In this paper, we present our experiences of using a mobile agent framework to design and realize a performance management system which exhibits "constrained mobility", constrained in the sense that performance monitoring agents are sent to execute within network elements and stay there until their task is accomplished. We present the architecture, design and implementation of such a system, compare and contrast it to static object approaches and present a detailed performance comparison to a similar Java-RMI based implementation, trying to assess the overhead of mobile agent solutions.

Keywords

Mobile Agents, Performance Management, CORBA, Java-RMI, TMN.

1. Introduction and Background

Network management has been the subject of intense research over the last decade, with the relevant progress being twofold: on the one hand, architectures and algorithms for solving management problems have been devised; and on the other hand, different management technologies have been proposed and standardized. From the protocol-based approaches of the early 90's, exemplified by the Simple Network Management Protocol (SNMP) [1] and OSI Systems Management (OSI-SM) [2], the focus moved to distributed object-based approaches in the mid to late 90's, exemplified first by the Common Object Request Broker Architecture (CORBA) [3] and later by Java's Remote Method Invocation (Java-RMI). More recently, the focus seems to be shifting back to protocol-based approaches the emerging Directory Enabled Networks (DEN) framework.

The paradigm of moving management logic close to the data it requires is a technique that has been conceived early in the evolution of management architectures, the relevant framework known as "management by delegation" [4]. Subsequent research showed the applicability of this concept in the context of OSI-SM [5] while a similar approach was subsequently standardized, the Command Sequencer Systems Management Function (SMF). More recently, the same concept has been proposed in the context of SNMP through the Scripting MIB. While such approaches are specific to the respective management frameworks, delegation in the context of general distributed object frameworks is achieved through *object mobility*. Mobile objects are usually termed *mobile agents* and when they act through emerging behavior in the Artificial Intelligence (AI) sense, they become *intelligent agents*. Mobility and intelligence are though orthogonal properties.

The emergence of mobile agent frameworks has led many researchers to examine their applicability to network management and control environments. [6] considered first code mobility and presented a taxonomy of the relevant aspects. [7] considered mobile agents in the context of the Intelligent Network (IN) and proposed an agent-based architecture for “active” IN service control. [8] discussed the general issues of using mobile agents for network management while a number of other researchers have attempted to use mobile agents in specific network management case studies. It is believed that mobile agents can provide better solutions at least to performance and fault management problems, given the large amount of data that needs to be moved around in respective solutions based on traditional approaches.

Mobile agents may move around the network from node to node and clone / destroy themselves according to their intelligence. We term this situation “full mobility” and it is this property that has not yet been exploited to good effect in network management. An alternative possibility for mobile agents is to move from node A to B, typically guided by a “parent” stationary agent, and stay there until their task is accomplished. We term this situation “constrained mobility” and we believe it is this approach that can be readily exploited in management environments. In this case, instead of predicting the required functionality, standardizing and providing it through static objects in network elements, mobile agents could support it in a dynamic, customizable fashion. The key advantage in this case is that the target node needs only to provide the required “bare-bones” capability which could be dynamically augmented through mobile agents, with the mobile agent logic able to change to reflect evolving requirements over time. Such a possibility would obviate the use of functionality such as the OSI-SM Systems Management Functions (SMFs) and similar capabilities provided in SNMP.

In the work described in this paper we are trying to evaluate the use of mobile agents for network performance management, assuming a constrained mobility paradigm in which a mobile agent is sent to execute and monitor information within a network element. The latter can be managed through a collection of static agents that offer similar capabilities to a OSI-SM or SNMP Management Information Base (MIB). The evaluation is twofold: first, we are interested in assessing the usability of a mobile agent platform as opposed to a static object platform such as CORBA or Java-RMI, and in particular the agent customization aspects; and second, we would like to examine the performance implications of using mobile agents in order to assess if the provided flexibility is potentially outweighted by the additional performance overhead. This work has been carried out in the context of the MIAMI ACTS project (Mobile Intelligent Agents in the Management of the Information infrastructure) [9], which examines the impact and possibilities of using mobile agent technology for network and service management.

The rest of this paper has the following structure. In Section 2 we summarize briefly the way in which performance monitoring is supported through generic but predefined functionality in the context of OSI-SM, SNMP and CORBA-based management systems. In Section 3 we examine the general benefits of mobile agents in network management and present an overview of the network management architecture of the MIAMI project. In

Section 4 we concentrate on the performance management domain and present the system architecture, design and implementation. In Section 5 we present an evaluation and assessment of the performance management system and in Section 6 we present a summary and conclusions.

2. Static Performance Monitoring

Performance management is one of the management functional areas identified initially in OSI Systems Management (OSI-SM) [2]. It addresses the availability of management information in order to be able to determine network load under both natural and artificial conditions. It supports the collection of performance information periodically in order to provide statistics and allow for capacity planning activities. Performance management needs access to a large quantity of dynamic network information. An important issue is to provide this information to management applications with a small impact on the managed network. A key requirement is the ability to convert raw traffic information to traffic rates with thresholds applied to them so that Quality of Service (QoS) alarms can be generated. An additional requirement is the periodic summarization of a variety of performance information for trend identification and capacity planning purposes.

QoS management applications monitor performance aspects both “within” the network and at “edge” nodes where customer services are offered, trying to identify potential performance degradations. They may subsequently trigger the reconfiguration of parts of the network in order to alleviate congestion e.g. by changing the routing strategy, re-allocating resources such as bandwidth to trails, etc. Monitored aspects of services may include the service availability, the supported capacity in terms of available bandwidth and the end-to-end delay. In the case of a “leased line” service with guaranteed QoS, e.g. as part of a Virtual Private Network (VPN) service, its availability may be affected by faults while the available capacity and delay may be affected by network congestion when the provided bandwidth is multiplexed. In general, performance management is coupled with both fault and configuration management.

A simplistic approach for collecting the required performance information is through periodic polling. In this case, the collected raw data is processed either at a centralized Network Management Station (NMS) or at Element Managers (EMs) which may form part of a hierarchical management system e.g. following Telecommunications Management Network (TMN) [10] principles. The problem with this approach is that it generates substantial management traffic and, subsequently, does not scale (it should be mentioned though that the generated traffic is smaller in the hierarchical compared to the centralized case). An alternative approach is to delegate monitoring activities to the network elements, reporting only QoS alarms or summarized reports to higher-level managers. The OSI-SM Metric Monitoring [11] and Summarization [12] systems management functions have addressed this requirement through generic Support Managed Objects (SMOs), which need to be provided in managed network elements. Facilities similar to metric monitoring have been subsequently provided in SNMP environments, initially in the Remote Network Monitoring (RMON) [13] specification. In addition, similar facilities could be provided in CORBA-based network elements.

The problem with such generic functionality is that it needs to be first researched, standardized, implemented and eventually deployed in network elements; this process typically takes a long time. In addition, any modification, e.g. for providing more sophisticated features that were not thought out in advance, needs to go through the full research, standardization and deployment cycle. For example, [14] identified additional functionality that combines the capabilities of metric monitoring and summarization objects in a powerful fashion but such additions need to go through a new standardization cycle. The specification of the OSI-SM systems management functions took a long time and is partly responsible for the perceived complexity and lateness in the deployment of OSI-SM-based network elements.

Mobile agents could provide similar facilities in a dynamic fashion, allowing for network elements with bare-bones *real resource* only and no *support* management information. Additional generic capabilities could be provided through mobile agents which would be sent to execute within a network element. The behavior of those agents could be altered dynamically at any time. The flexibility provided to management applications would be enormous, since they could now "customize" network elements for performance and other management activities according to their requirements and would not be restricted by the available standardized facilities. On the other hand, network elements should be able to host mobile agents through suitable platform infrastructure and real resource managed objects should be realized as static agents. The performance implications of using mobile agents universally are addressed later in this paper.

3. Mobile Agents in Network Management

3.1 The General Benefits of Mobile Agents

In the context of communication networks, mobile agents enable the transformation of current networks into remotely programmable platforms. The concept of "Remote Programming" using mobile agents is considered as an alternative to the traditional "Client/Server programming" based on the Remote Procedure Call (RPC) or the static distributed object paradigm (e.g. CORBA, Java-RMI). In contrast to "shouting" requests across the network from a client to a server, mobile agents transport themselves to the remote (server) computer, where the work must be done. Compared to static approaches, mobile agents provide two major advantages:

- *tactical advantage*: Improved performance, where client software can migrate and work locally with the server software instead of communicating across the network. The performance advantage of remote programming through mobile agents depends partly upon the network: the lower its throughput or availability, or the higher its latency or cost, the greater the advantage.
- *strategic advantage*: The strategic advantage of remote programming is customization. Agents provide flexibility in extending the functionality offered by existing communication systems.

This means that mobile agents enable control tasks to be performed in a real distributed manner. In particular this concept enables telecommunications services to be provided instantly and to be customized directly at the locations where the intelligence is needed. An additional key benefit is that access to local information at a particular network node does not suffer from varying delays as it is the case with remote access. This means that that the collected information is more accurate while fine-grain intervals may be used in cases of periodic polling.

Mobile agent frameworks are currently addressed by two standards bodies. The Federation of Intelligent Physical Agents (FIPA) [15] looks at high-level semantically rich interactions between software agents that deploy some form of intelligence and adaptability, having its roots in Distributed Artificial Intelligence (DAI). The Object Management Group (OMG) looks mostly at the issue of mobility according to a standard interoperable framework through its Mobile Agent System Interoperability Facility (MASIF) [16]. In the latter, *agencies* model the execution environment able to host mobile agents and correspond roughly to OSI-SM/SNMP *agents* in the manager-agent framework or to the concept of the Distributed Processing Environment (DPE) *node* in distributed object frameworks. Within an agency, *static* agents provide the basic functionality which is statically defined but can be augmented dynamically through *mobile* agents which are sent to execute there.

3.2 Overview of the MIAMI Network Management Architecture

The MIAMI project objective [9] is to examine the applicability of Mobile Intelligent Agents (MIAs) to network and service management. The OMG MASIF standards [16] are to be validated, refined and enhanced while the project will provide a reference implementation for the unified MIA framework. In order to achieve this, MIAMI has defined a case study and associated environment which will allow co-operating customers to dynamically form Virtual Enterprises (VEs) for providing services to end-users. The VE makes use of services offered by a Active Virtual Pipe Provider (AVPP), a business role similar to a TMN Value Added Service Provider (VASP) or a Telecommunications Information Networking Architecture (TINA) Retailer. The AVPP provides a programmable, dynamic virtual private network, as needed by the virtual enterprise. The AVPP makes use of connectivity services offered by a Connectivity Provider (CP), which is a business role similar to a TMN Public Network Operator (PNO).

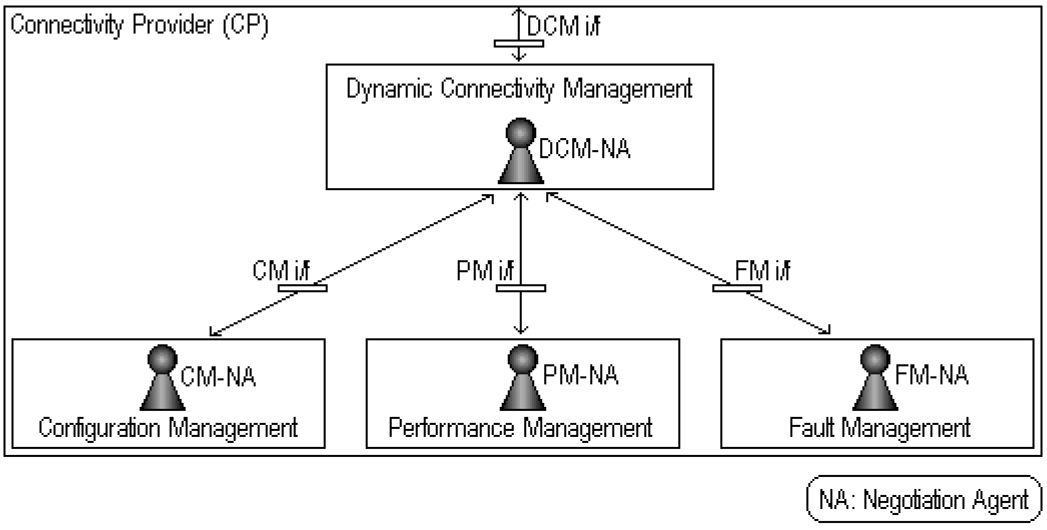


Figure 1: The MIAMI Active Virtual Pipe Domain

The Active Virtual Pipe (AVP) provides an abstract view of the dynamically configurable global connectivity resources for the transfer of audio, video and data streams. The AVP provides a programmable, dynamic extranet according to the demand of the Virtual Enterprise. The AVP does not provide a static Virtual Private Network but scheduled and dynamic connectivity, which adapts to the constantly changing needs of Virtual Enterprises. For instance, it is possible that the capacity of connections can be configured dynamically by the VE in accordance with the current business activities. The VE can describe complex connectivity requirements, which are realized by the AVP. This is achieved through mobile intelligent agents which are sent to execute in the CP domain. The latter comprises configuration, performance and fault management functionality [18]. An overall simplified architecture of the MIAMI CP domain is shown in Figure 1.

The DCM interface provides dynamic connectivity management capabilities to customers, which in the MIAMI case is the AVPP domain. Its functionality is similar to the TMN Xuser or the TINA ConS reference points. Through this interface a contract may be negotiated, and for this intelligent agent capabilities and semantically rich, high-level interactions are used. After a contract has been established, a user may ask for connectivity services. In the case of MIAMI, dynamically managed Asynchronous Transfer Mode (ATM) Virtual Path (VP) services are provided to support the dynamic extranet required by the virtual enterprise. By dynamically managed services it is meant that the user may provide its own logic which will monitor the connectivity resources and will dynamically re-negotiate parameters such as bandwidth in the case of under or over-utilization. It also may ask for the re-establishment of connectivity in the case of faults or take other action This functionality is provided by the DCM sub-domain which provides a programmable connectivity service i.e. an active virtual pipe.

The DCM sub-domain uses the subordinate Configuration, Performance and Fault Management sub-domains (CM, PM and FM respectively). Configuration management is responsible for the configuration of network resources, in this case the setting-up and releasing ATM VPs. It also has an overall view of the underlying network topology in terms of both static (links, switches) and semi-dynamic resources (VP trails). Fault management is responsible for identifying and isolating faults, which is done first through alarm correlation and then through testing activities. Finally, performance management is responsible for monitoring the utilization of VP trails and generating quality of service alarms and summarization reports which may be used for dynamic capacity re-negotiation by the users of the DCM service. A more detailed description of the Connectivity Provider domain as a whole can be found in [18]. For the rest of this paper, we will concentrate on aspects related to the performance management sub-domain.

4. Performance Management System Architecture, Design and Implementation

4.1 System Architecture and Design

The Performance Management (PM) domain provides monitoring facilities for trails, i.e. ATM Virtual Paths (VPs) in the context of the MIAMI Connectivity Provider domain. The PM design is generic but it is only used by the Dynamic Connectivity Management (DCM) domain in the MIAMI system. It consists of three agents, two stationary and one mobile, as described below.

- *Performance Negotiation Agent* (PNA) - a stationary agent which provides an interface that negotiates information between the Performance Management domain and the DCM domain. This agent corresponds to Network and Element Management Layer (NML and EML) functionality in terms of the TMN model.
- *Performance Monitor Agent* (PMA) - a mobile agent that is created by the PNA and migrates to a network element in order to perform local monitoring and summarization functions. This agent corresponds to the functionality of a combined metric monitor and summarization object [11][12] but it is provided dynamically and can be easily changed and customized to reflect changing requirements.
- *Performance Element Agent* (PEA) - a stationary agent located at a network element that provides a performance management "wrapper" or adapter of the underlying technology (SNMP, TMN Q3, etc.) This is necessary given the fact that network elements do not provide yet native agent-based interfaces. In a target environment, the MIB of a Network Element (NE) would be provided through a set of stationary agents. Functionality at this level corresponds to the TMN NE level.

These agents and their interactions can be seen in Figure 2 which depicts the performance management system architecture using agent technology.

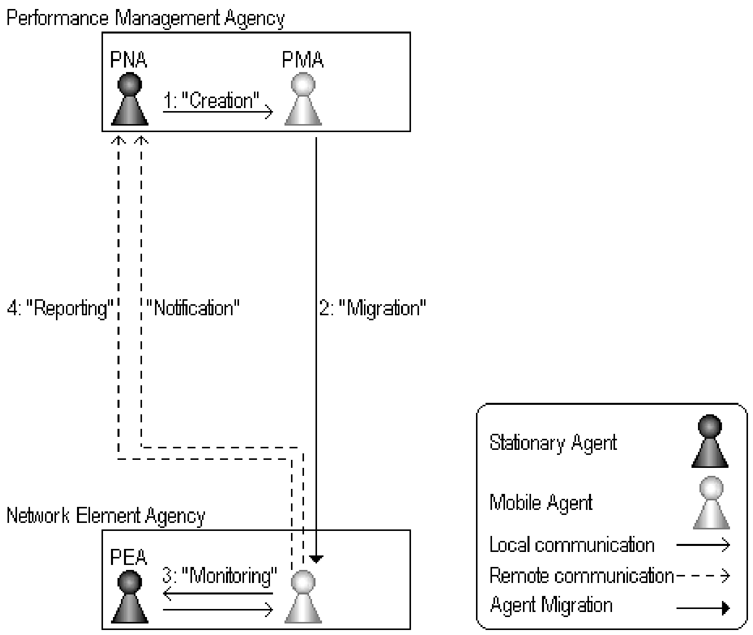


Figure 2: Performance Management System Architecture Using Mobile Agents

A typical scenario commences with a request for performance monitoring that comes along with a set of required network configuration and user parameters. The request arrives to the Performance Negotiation Agent, which creates a new instance of the Performance Monitor Agent that possesses the necessary parameters that came with the request regarding the monitoring activities required. This agent migrates to an agency located at the specified network element. On arrival at the network element, the Performance Monitor Agent periodically obtains raw utilization and loss information (counter type values) provided by the stationary Performance Element Agent on request. This information is used by the Performance Monitor Agent to perform metric monitoring and summarization activities, check the currently set thresholds, and gather utilization rates along with other performance measurements. Monitoring results and notifications are remotely passed to the Performance Negotiation Agent which then informs the user. Summarized monitoring results are sent from the Performance Monitor Agent at scheduled periods, while notifications are sent immediately in an asynchronous manner, every time a threshold is crossed. The algorithm used for triggering such alarms may be customized by the user.

4.2 System Implementation Using the Grasshopper Agent Platform

Following the above system architecture and design for the performance management system, the Unified Modeling Language (UML) was used to create various design diagrams. These diagrams helped us to clarify the behavior of the system, the various interactions involved, and the functionality to be provided. As the software design phase progressed, a detailed UML class diagram was created, providing a visual model of the classes involved in the system.

For the development of the Performance Management domain, the Grasshopper mobile agent platform was used [17]. The Grasshopper platform itself and mobile agents are implemented in Java, which means that they can be used in any computing platform that supports a Java Virtual Machine. Some of the main elements of the Grasshopper platform are:

- *Agency*: Provides a runtime environment for agents. Every agency has the ability to interact with other agencies either through remote communication or agent migration.
- *Region*: Facilitates the management of agencies and agents. A region contains one region registry and several agencies that serve a common purpose.
- *Region Registry*: Provides directory-like information about agencies, places and agents in a region.

Along with these elements there is of course the concept of an *agent*, a software object that can act autonomously and migrate. There are two types of agents in Grasshopper, mobile and stationary. Mobile agents can migrate autonomously between different locations. Stationary agents reside permanently in their creation agency, offering services to other agents and possibly encapsulating non-agent based entities. The development of a simple agent using the Grasshopper platform involves the creation of a class that simply inherits from the Grasshopper *MobileAgent* or *StationaryAgent* classes. Figure 3 shows the class hierarchy for the agents of the performance management system.

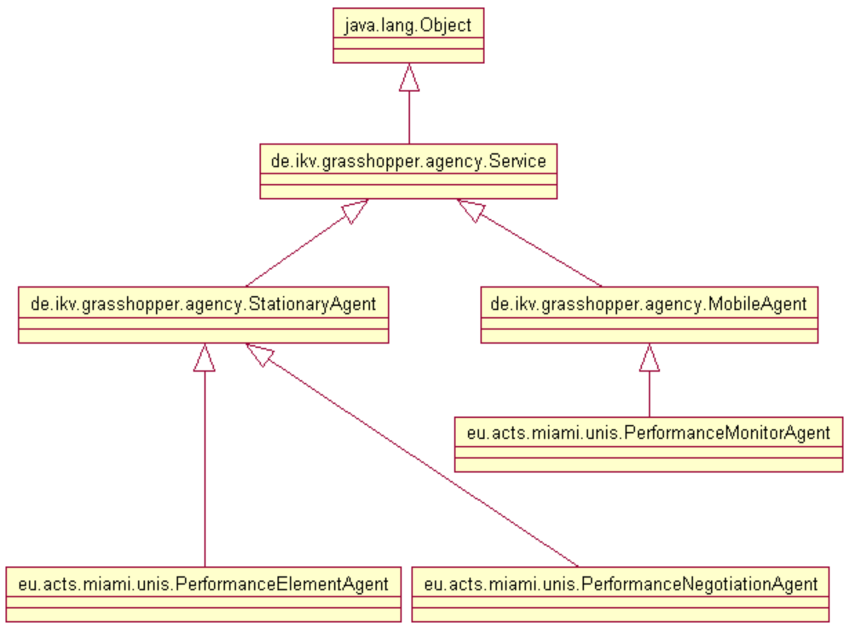

java.lang.Object
de.ikv.grasshopper.agency.Service
de.ikv.grasshopper.agency.StationaryAgent
de.ikv.grasshopper.agency.MobileAgent
eu.acts.miami.unis.PerformanceMonitorAgent
eu.acts.miami.unis.PerformanceElementAgent
eu.acts.miami.unis.PerformanceNegotiationAgent

Figure 3: Inheritance Hierarchy of Agent Classes

As already mentioned, the idea behind the Performance Monitor Agent is based on the Metric Monitoring and Summarization OSI-SM SMFs [11][12]. [11] defines a basic metric monitor object as well as the mechanism for threshold handling. The monitor obtains counter or gauge type values that converts to utilization rates by subtracting two successive counter type measurements, dividing them by the time interval between them (Granularity Period) and possibly applying statistical smoothing algorithms. Thresholds are checked by the monitor according to the "Severity Indicating Gauge-Threshold" model. When a threshold is triggered a notification is sent by the monitor, while for subsequent measured values close to this threshold level, notifications won't be repeatedly sent unless the measured value passes a specified "Threshold Clear" level [11][14]. While this functionality is fixed in OSI-SM, the PMA logic may be customized by a client e.g. to provide a different model for triggering notifications, based on semantic knowledge of the monitored resources. The important customization aspects are described next.

4.3 User Customization

Today's equipment comes with some built-in functionality that can be used for network management purposes. Manufacturers provide a set of standardized as well as proprietary objects that can be remotely invoked to support manageability of that network element. This functionality is fixed so that it cannot be altered or extended. In that sense a network element can be characterized as a black box with pre-programmed management capabilities. As the telecommunications industry evolves support at the network element level for distributed object architectures , mobile agents could be eventually introduced. The key benefit is that mobile agents implementing customized functionality could migrate and execute there, augmenting dynamically the element's capabilities. We examine here how the functionality of a mobile agent can be customized in our system.

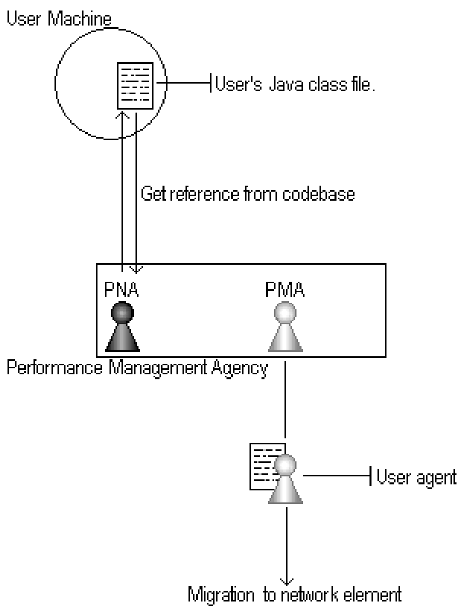

User Machine
User's Java class file.
Get reference from codebase
PNA
PMA
Performance Management Agency
User agent
Migration to network element

Figure 4: An Advanced Scenario Involving User Customization

A user of the performance management system can customize the standard functionality of the Performance Monitor Agent by providing a user agent class tailored to specific needs (see Figure 4). The User Agent along with its inherited functionality from the PerformanceMonitorAgent migrates to the network element to perform customized QoS monitoring. The class diagram of Figure 5 shows the design approach followed in order to allow the customization or extension of the standard PerformanceMonitorAgent class. The UserAgent class inherits all the methods provided by the PerformanceMonitorAgent class. By doing this, the user can extend or overwrite the functionality of the PerformanceMonitorAgent, also being able to reuse its standard methods. The UserAgent class also implements a PerformanceMonitor interface in order to override the standard "metricMonitor" method of the PerformanceMonitorAgent. In that way, it can provide an implementation that customizes the way metric monitoring is performed, providing its own algorithms.

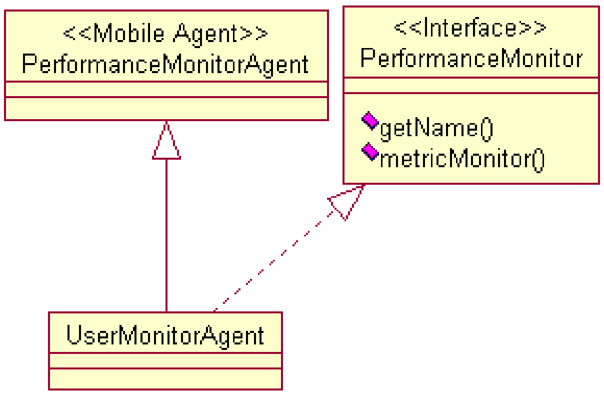
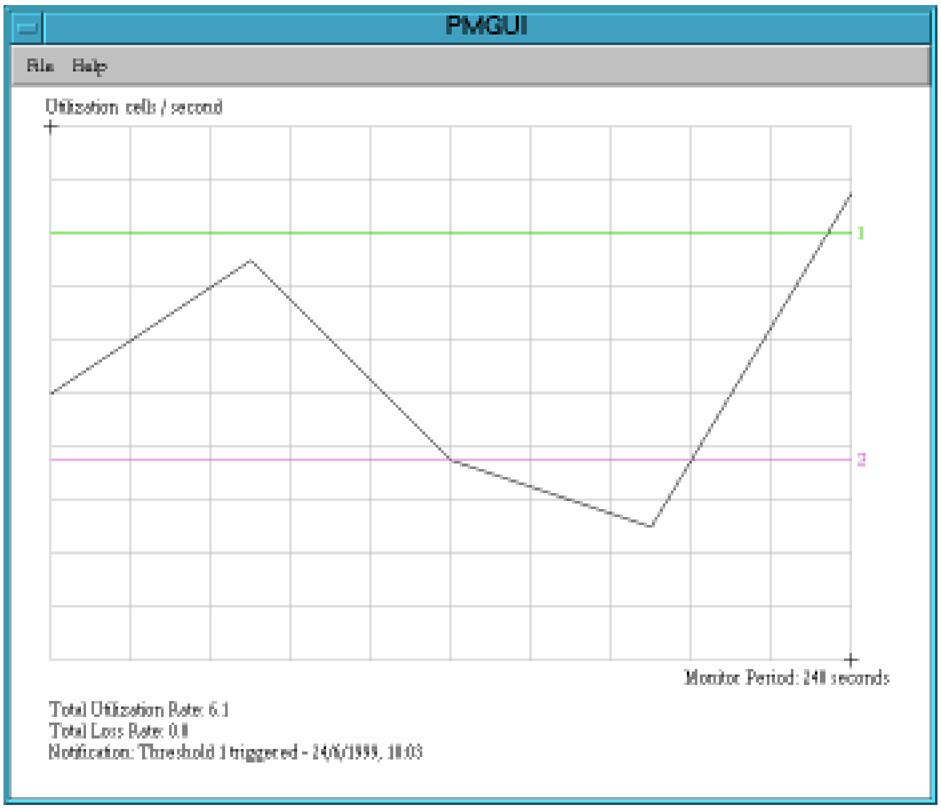


Figure 5: Class Diagram for a Customized User Agent

4.4 The System in Operation

The performance management system has been developed using Sun’s JDK version 1.1.7b and the Grasshopper platform version 1.2 on Sun’s Solaris version of the UNIX operating system. Along with the libraries provided by the JDK and the Grasshopper platform, the AdventNet SNMP Manager version 2.0 libraries were used as a means of interfacing with SNMP. The AdventNet libraries are used in the PerformanceElementAgent for accessing the requested raw information. The network element monitored is an ATM switch running an SNMP agent that provides management information. The raw information requested is a counter value representing the number of cells that have been transmitted through a specific port and virtual path (VP). In the same manner, a counter for cell loss is also requested. An agency running in a workstation directly connected to the switch provided a mobile agent environment near the network element. The Performance Monitor Agent performs metric monitoring using the raw information and then summarizes its results to be sent to the Performance Negotiation Agent when a scheduled reporting time comes. The Performance Negotiation Agent receives the results which finally reach the user through a graphical user interface (GUI) as seen in the Figure 6. While the GUI is updated with utilization and cell loss results in a scheduled way, asynchronous notifications are displayed in real time.



Total Utilization Rate: 6.1
Total Loss Rate: 0.0
Notification: Threshold 1 triggered - 24/6/1999, 18:03

Figure 6: The Performance Management Graphical User Interface

5. Evaluation and Assessment

While in the previous section we showed how mobile agent technology can be used intead of a static distributed object technology for designing and building hierarchical management systems with the additional advantage of dynamic customization and object migration, in this section we will look at the performance implications of using mobile agent technology. As such, we decided to design and build two additional versions of the system, using Java-RMI and CORBA respectively as static distributed object platforms. The reason we chose Java-RMI is that the Grasshopper platform also uses Java-RMI (as well as a proprietary protocol), so we would be able to see the precise overheads incurred by the mobile agent support infrastructure. In addition, the comparison with CORBA would allow us draw conclusions on the overheads of mobile agents platforms compared to an emerging distributed object technology for network/service management.

In the case of the Java-RMI/CORBA based performance management system, a Performance Negotiation object located in one machine sends a request for QoS monitoring to a Performance Monitor Factory (PMF) object located at a network element. When a request arrives to the PMF, it locally creates a new instance of a Performance Monitor object that will perform QoS monitoring and summarization functions. A Performance Element object is also located at the network element and provides raw performance information. The functionality and algorithms in both systems were identical so that we could directly compare the two approaches. It should be noted though that in the case of a static distributed object approach the functionality of the performance monitor object is static and cannot be altered, in a similar way to OSI-SM and SNMP support object facilities. Finally, we chose to use CORBA with the Java mapping for reasons of uniformity and we used the Sun Microsystems openly available version of CORBA with the Java mapping.

We were interested in measuring the following aspects of the system. First, we measured remote invocation response times. Timestamps were taken using the *currentTimeMillis* method of the *java.lang.System* class. A list of 25 elements was transferred 100 times between two objects located in different machines, each time recording the total time and finally calculating the average and standard deviation of these measurements. The same procedure was repeated while increasing the number of elements in the list to 50, 75, and 100. This operation is in fact modeling the periodic summarization reports generated by the Performance Monitor Agent. And second, we measured the TCP packet sizes using the tcpdump program that originated at the Lawrence Berkeley laboratory. The sizes reported reflect the payload at the TCP level. All these measurements where taken using two different machines over a lightly loaded Ethernet in the role of the management network with the following specification: Sun Microsystems Ultra-10, 256MB of memory, Sun's Solaris 2.5.1 version of UNIX.

5.1 Response Times

The response times of management operations for the two versions of performance management systems have been considered. First we examined the performance aspects of remotely invoking operations between two objects located in different machines. An array of objects (class "java.util.Vector") containing 25, 50, 75 and 100 "Double" numbers respectively was passed as a parameter using both the Mobile Agent and the RMI system.

Each time, the response times for the transfer were monitored. The protocol that Java RMI relies on is the Java Remote Method Protocol (JRMP).

	25	**50**	**75**	**100**
Mobile Agents	37.74	64.9	85.64	108.23
Mobile Agents over RMI	18.28	25.25	33.02	40.5
RMI	6.2	9.79	13.11	15.86
CORBA	4.42	5.06	6.72	8.9

Table 1: Response Times (msecs).

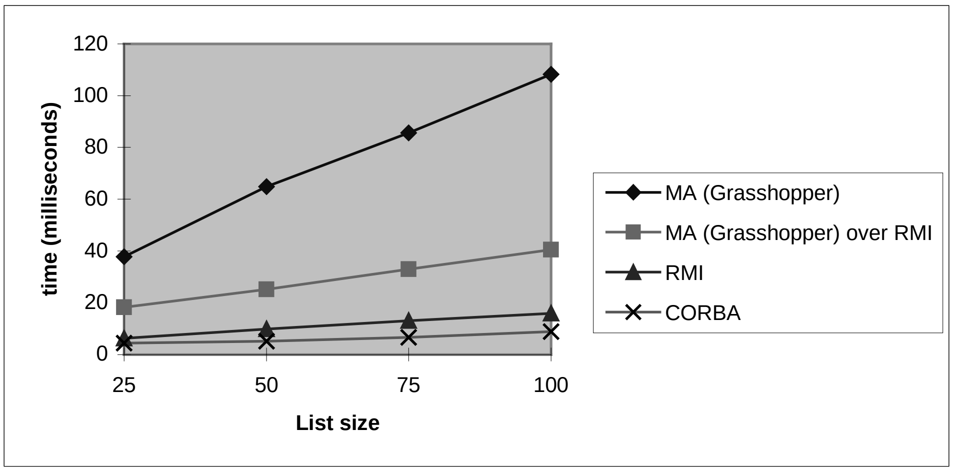


Figure 7: Response Times graph.

These graphs show some important results. First, there is a significant performance penalty to pay for remote method invocations in the context of a mobile agent platform compared to Java-RMI and CORBA. Second, Grasshopper performs much better over RMI in comparison to the default proprietary protocol. The Grasshopper approach seems to result in at least three times slower response times in comparison to Java-RMI and CORBA.

An additional performance overhead in Grasshopper is the initial time for the PerformanceMonitorAgent to migrate from the PNA node to the network element. The mobile agent needed an average of 1505 milliseconds to migrate, a performance overhead much larger than the time required to create a performance monitor object through its factory in the static RMI/CORBA approaches which is less than 15 msec i.e. there is additional overhead of two orders of magnitude. In cases of constrained mobility, which is the approach used in this paper, this is a one-off overhead and can be tolerated. On the other hand, this measurement shows that agent mobility has relatively high performance overheads and this should be born in mind when designing systems exhibiting full mobility.

5.2 Packet Sizes

We also measured the packet sizes in both cases. An array of objects (class *java.util.Vector*) containing 25, 50, 75 and 100 *double* numbers respectively was remotely sent using remote invocations both in the Mobile Agent and the RMI system. Each time, the payload of the TCP packets was measured. The results (in bytes) can be seen in Table 2 while a graph of the above results can be seen in Figure 8. It is interesting to see from these results that the proprietary Grasshopper protocol incurs the least traffic, while packet sizes decrease as we move from Grasshopper over RMI to RMI and CORBA respectively.

	25	**50**	**75**	**100**
Mobile Agents	455	820	1145	1550
Mobile Agents over RMI	1207	1572	1897	2299
RMI	654	1019	1214	1749
CORBA	588	788	988	1188

Table 2: Packet Sizes for Mobile agents and RMI

We also measured the packet overhead of migrating the PerformanceMonitorAgent from the PNA node to the network element. The required data is 2854 bytes, which again is much higher than the required amount of data to create a performance monitor object through its factory in the static approaches which is around 500 bytes. This again will incur a substantial traffic overhead in full mobility environments.

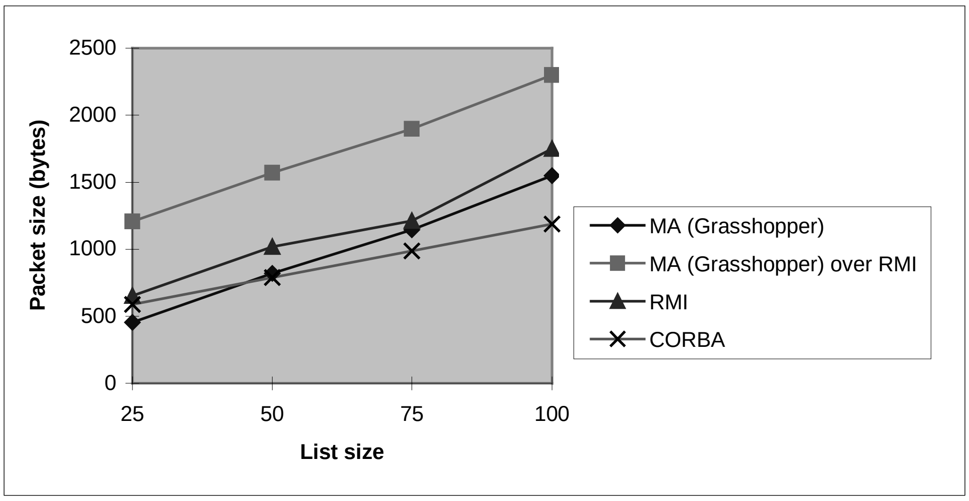


Figure 8: **Packet Sizes graph.**

6. Summary and Conclusions

In this paper we presented the design, implementation and evaluation of a simple network performance monitoring system based on mobile agent technology. In our system, mobile agents are created at the network management level according to user requests and then migrate to network elements to perform monitoring functions in a local manner. The behavior of the monitoring algorithms can be customized, enabling dynamic programmable functionality to be provided directly in the managed network elements. This performance monitoring system is part of the network management architecture of the MIAMI project which also comprises configuration and fault in addition to performance management functions.

One of the key targets in embarking in this exercise was to evaluate the use of mobile agent technology in comparison to static object approaches in network management environments. The architecture, design and implementation presented in section 4 show that mobile agent platforms exhibit the same programmability characteristics to static object platforms. In addition, both remote method invocations and local invocations are possible. The same object-oriented principles and similar Application Programming Interfaces (APIs) can be used in mobile agent environments. A key advantage of mobile agents is the provision of dynamic services in network elements that have not been pre-programmed with such facilities. The customization of mobile agent behavior can provide a powerful mechanism for “intelligence on demand”. In this work, we have shown how users can use class inheritance to customize basic performance monitoring functions to their needs.

On the other hand, while design and programmability aspects are similar to static object approaches, there is a performance overhead to pay when using mobile agents. Remote method invocations are at least three times slower than those in Java-RMI / CORBA and this difference could be more pronounced when comparing performance to the protocol-based OSI-SM and SNMP approaches. In addition, agent migration incurs a substantial overhead in terms of both latency and required data to be transported across the network. This is less of an issue in constrained mobility environments but could lead to performance and scalability problems in full mobility environments where a large number of mobile agents migrate relatively often.

While initially mobile agent frameworks were thought as rivals to static distributed object frameworks, a view also stated in [7], the two approaches need to coexist. Static object approaches can provide superior performance characteristics. Real synergy could thus be achieved if stationary agents could be provided using static objects, with method invocations being possible between mobile agents and static objects in both directions. Such an environment would combine the best of both worlds but it is not clear at present if this type of seamless integration is achievable. This is an aspect that bodies such as the OMG should try to address.

Acknowledgments

This work was undertaken in the ACTS MIAMI project, which is partially funded by the Commission of the European Union.

References

[1] J.Case, M.Fedor, M.Schoffstall, J.Davin, *A Simple Network Management Protocol (SNMP)*, IETF RFC 1157, 1990.

[2] ITU-T Rec. X.701, Information Technology - Open Systems Interconnection, *Systems Management Overview*, 1992.

[3] Object Management Group, *The Common Object Request Broker: Architecture and Specification (CORBA)*, Version 2.0, 1995.

[4] Y. Yemini, G. Goldszmidt, S. Yemini, *Network Management by Delegation*, in Integrated Network Management II, Krishnan, Zimmer, eds., pp. 95-107, Elsevier, 1991.

[5] N. Vassila, G. Pavlou, G. Knight, *Active Objects in TMN*, in Integrated Network Management V, Lazar, Saracco, Stadler, eds., pp. 139-150, Chapman & Hall, 1997.

[6] M. Baldi, S. Gai, G.P. Picco, *Exploiting Code Mobility in Decentralised and Flexible Network Management*, Proc. of the 1st International Workshop on Mobile Agents, Berlin, Germany, April 1997.

[7] Breugst, M., Magedanz, T., *Mobile Agents - Enabling Technology for Active Intelligent Network Implementation*, IEEE Network, Vol. 12, No. 3, May/June 1998.

[8] A. Bieszczad, B. Pagurek, T. White, *Mobile Agents for Network Management*, IEEE Communications Surveys, Vol. 1, No. 1, http://www.comsoc.org/pubs/ surveys, 4Q1998.

[9] Mobile Intelligent Agents for the Management of the Information Infrastructure (MIAMI) ACTS project, project page: http://www.fokus.gmd.de/research/cc/ima/ miami/, page at Univ. of Surrey: http://www.ee.surrey.ac.uk/CCSR/ACTS/Miami/

[10] ITU-T Rec. M.3010, *Principles for a Telecommunications Management Network (TMN),* Study Group IV,1996.

[11] ITU-T Rec. X.739, Information Technology - Open Systems Interconnection, *Systems Management Functions - Metric Objects and Attributes*, 1992.

[12] ITU-T Rec. X.739, Information Technology - Open Systems Interconnection, *Systems Management Functions - Summarization Function*, 1993.

[13] S. Walbusser, *Remote Network Monitoring (RMON) Management Information Base*, IETF RFC 1271, 1991.

[14] G. Pavlou, G. Mykoniatis, J. Sanchez, *Distributed Intelligent Monitoring and Reporting Facilities,* IEE Distributed Systems Engineering Journal (DSEJ), Special Issue on Management, Vol. 3, No. 2, pp. 124-135, IOP Publishing, 1996.

[15] Foundation for Intelligent Physical Agents, web page: http://www.fipa.org/.

[16] Object Management Group, *Mobile Agent System Interoperability Facilities Specification*, orbos/97-10-05, 1997, ftp://ftp.omg.org/pub/docs/orbos/97-10-05.pdf

[17] The Grasshopper Agent Platform http://www.ikv.de/products/grasshopper/index.html.

[18] D. Griffin, G. Pavlou, P. Georgatsos, *Providing Customisable Network Management Services Through Mobile Agents*, Proc. of the 7th International Conference on Intelligence in Services and Networks (IS&N'00), Athens, Greece, February 2000.