A mobile-agent and SNMP based management platform built with the Java ProActive library

Emmanuel Reuter, Françoise Baude
Oasis, INRIA - CNRS - I3S
2004 route des Lucioles, BP 93
06902 Sophia Antipolis cedex – France
First.Last@inria.fr

Abstract

This paper presents our research into determining an adaptive and an up-to-date service developed for system and network management. By using a discovery process, the effective network topology is recorded and refreshed as necessary. In that way, by mixing collected information at each sub-network for example, an itinerary can be obtained that spans the whole administrative domain. Based on such possibility, we have developed a pre-programmed library that can be easily used in order to obtain a Mobile Agent for Network and System Management whose itinerary is dynamic.

I. Introduction

For several years now, the applicability and usefulness of mobile agent technologies for distributed System and Network Management (SNM) have been recognized. One of the main points is to delegate to autonomous and possibly mobile agents the administration tasks, as such, distributing the network and computation loads instead of centralizing them towards and on the manager host [1]. For a recent discussion of advantages of mobile agent based approaches in SNM, refer to [2].

The Java programming language is today the most adequate for building such SNM platforms, as it provides: (1) a total portability on all kind of operating systems (due to the Java Virtual Machine), (2) built-in distribution and mobility management mechanisms (RMI – Remote Method Invocation–, dynamic class loading, serialization, etc.), (3) built-in security management mechanisms (permissions, security policies). Moreover, for this specific application domain, SNMP operations can be invoked from Java programs, in particular, using the AdventNet SNMP package [3].

Several academic research platforms have been recently built in order to prove the effectiveness of Java mobile-agents based SNM: Mole [4], MAMAS [5], MAP [6], just to mention a few. They all have as a prerequisite the following: prior to the execution of any management operation, the network and system elements must run a daemon specific to the SNM platform, in order to be able to host a mobile agent. The daemon's role is to control the arrival and departure of mobile agents that come in order to execute their management operations locally, control their life-cycle and the multi-agent coordination. This of course requires that the system or the network element be Java-compliant to run this specific platform daemon. The management function is not mandatory pure JVM operations because it can be mixed with SNMP operations thanks to the Java/SNMP API.

However, in realistic infrastructures, network and system elements that the supervisor must manage are heterogeneous in the sense that not all of them are Java compliant (for instance, routers, printers are not currently able to execute JVMs); nevertheless, one can assume that they all run a standard SNMP agent, which can be remotely monitored through the SNMP protocol. Also in realistic infrastructures, the effective topology of the interconnected network and system elements is dynamic, as devices or computers may be up or down, devices or laptops may be added or removed, etc. Those elements may be part of different sub-networks (i.e., LANs), probably interconnected by higher latency and slower bandwidth links (i.e. WANs), as for instance in a multi-national or multi-regional enterprise.

One of the most tedious day-to-day task for a network and system manager is to keep the effective topology he/she has the responsibility, in an up-to-date state, mainly in order to execute health monitoring. Fault diagnosis and network configuration are other important tasks which can

also have some effect of the effective topology of the managed whole network.

Using any of the above mentioned mobile agent based platform implies to first deploy the infrastructure (daemons) and then to be able to tell a mobile agent which system or network elements it must visit in order to locally run the management function. As the topology may dynamically change and as some of the elements that must be managed can not be able to host a Java mobile agent, we think that those platforms lack some functionalities in order to be applicable for realistic infrastructures.

II. Approach

In order to solve those real-world problems, our approach takes the form of a mobile agent based SNM platform which:

(1) automatically maintains an up-to-date effective topology of the network under management, which is composed of either SNMP compliant or Java-plus-SNMP compliant elements, structured into several sub-networks. In each sub-network, the communication bandwidth is assumed to be high enough such as to avoid using one mobile agent visiting each element in turn. Instead, one mobile agent could be moved on any one of the possible hosts running a platform specific daemon, and remotely execute the function on every element of the sub-network using the standard SNMP operations.

(2) provides to the mobile agent programmer, an API for building various *itineraries* [7] for mobile agents that reflect the up-to-date effective topology or part of it. Those itineraries are built up with two types of *destinations*: a destination type onto which the mobile agent can effectively move to[1] and then, if required, locally execute one pure Java or SNMP-based management function; an other "destination" type for which a SNMP-based management function will be remotely triggered, as it is not possible to host the mobile agent (either because the element is not Java-compliant, or it is not running the platform specific daemon). In this second type, the SNMP management function is remotely triggered by the mobile agent, from a destination of the first type it is currently located on.

Our SNM platform is built with ProActive, a 100% pure Java library for mobile and distributed computing based on active objects (`www.inria.fr/oasis/ProActive`) [8]. As ProActive's aim is to ease distributed programming (for instance, by abstracting away from synchronization and management of method invocations among remote active objects), extending the SNM platform with new management functions should be readily affordable to system and network managers and end-users of the platform.

[1]this is the usual sense of what is a destination in a mobile-agent itinerary

This paper will not specifically focus on the programming methodology, but instead, we will explain in section III how we build and maintain the effective topology, and in section IV, how itineraries are built in relation with the effective topology and transparently used by a mobile agent which "moves" from destination to destination in such itineraries. Section V studies some performance tradeoffs between pure SNMP remote management compared with this mixing of mobile agent and SNMP based management, on a real test bed. Section VI concludes while comparing with related works.

III. Building and maintaining the effective topology of the network

The purpose is to dynamically discover all system or network elements that are reachable on the network, gather some information on each element, and register all this in a specific server where this will be used for building itineraries for mobile agents. Such a server is called an *ItineraryServer*. This discovery process is then periodically re-executed in background, in order to have an up-to-date vision of the effective topology of the network.

A. Implementation

In the following, we consider a network as an IP subnet. Of course, the administrative domain of the managed enterprise may be composed of several networks,

A *DiscoveryAgent* programmed as a ProActive active and mobile object is in charge of the discovery of elements of a network, using only SNMP queries. The first and only element that needs to be queried in order to discover all other elements in the network is a piece of active equipment such as a seed router or a switch with a SNMP agent. Indeed, as such an active equipment systematically records all Ethernet addresses of the alive hosts on the network, it is enough to read and to correlate the corresponding SNMP MIB variables (e.g. `ip.ipNetToMediaTable`, `dot1dtBridge`) in order to build the topology of the network (the list of elements and the way they are interconnected). Nevertheless, it is necessary to filter those data (pairs IP/Ethernet addresses) such as to avoid to scan an IP subnet different as the current one. For each element that is discovered, the following kind of information is recorded in the *ItineraryServer* associated to each *DiscoveryAgent*: the state (alive or not); network parameters (IP and Ethernet addresses); interface types; if this element executes an SNMP agent; if this element currently executes a ProActive node, that is a specific daemon of our platform that could host a ProActive mobile object dedicated to a SNM task.

The *DiscoveryAgent* executing the discovery process must run on a ProActive node that may be local to the

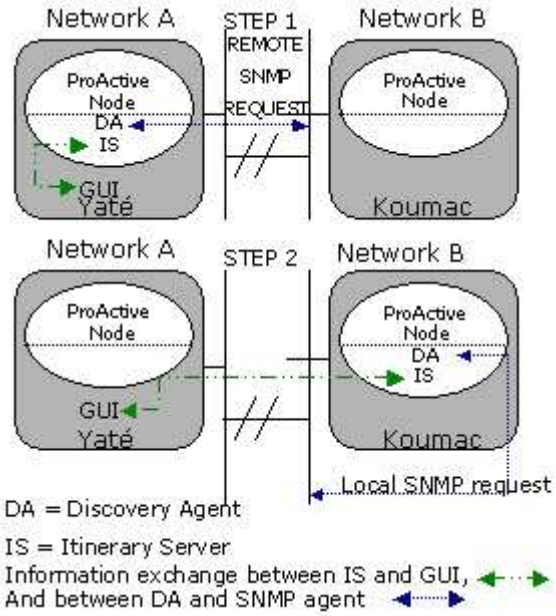

Fig. 1. Discovery of the topology of network B, by remote SNMP investigation from network A (step 1), or by local SNMP investigation after the DiscoveryAgent was able to migrate from network A to network B (step 2).

current network or not. For instance, this ProActive node may be the one which hosts the GUI (also a ProActive active object), but it is not mandatory. However, if during the discovery process of a remote network, the *DiscoveryAgent* locates a ProActive node, it migrates (and its associated *ItineraryServer* also) onto this node. As such, the discovery process of the network executes locally and so ends up faster (figure 1). On the contrary to some other SNM platforms, we do not need to make the assumption that prior to use[2] we already have at least one daemon specific to the platform running on each network.

IV. Building and using itineraries for management tasks by mobile agents

A. General idea and principles

An itinerary is a mixing of destinations onto which a SNM agent will effectively migrate and execute some Java or SNMP code on arrival, and of destinations that only represent elements for which the SNM task must take place without a move of the mobile agent (this requires the presence of an SNMP agent on those elements). Instructions for an SNMP agent will be triggered through a classical SNMP client-server interaction, originating from the mobile SNM agent, wherever the host it is actually located (it can be running on the same host, or it can be running on a host on the same network or even be located on a different network).

*ItineraryServer*s are able to cooperate on demand in order to build itineraries that span several networks. Of course, elements belonging to the same network will appear close in an itinerary in order for a SNM mobile agent to avoid migrating more than once towards a given network.

By requesting the up-to-date information recorded in the local *ItineraryServer* (which itself queries the others *ItineraryServer*s if required), any mobile agent can be provided with an itinerary that will enable to apply the SNM function to a set of elements, for instance:

- in an SNMP way only, inside the current network or among several networks (without any migration),
- in a mixed SNMP-Java way inside the same network (without any migration, except one in order to reach a ProActive node in the target network),
- in a mixed SNMP-Java way among several networks (with at least one migration for reaching every network in turn).

B. Structure and usage of an itinerary

An itinerary is a list of *Destination*s, which can either be a *NodeDestination* or a *SNMPDestination*. Each destination must provide the following information: an identifier of the destination, and an identifier of a method name that must be executed on arrival. For instance: `<''//koumac/node/'', ''echo''>` for a *NodeDestination*, `<''Bourail'', ''public'', ''snmpOnArrival''>` for a *SNMPDestination* (see Code in figure 2). As the SNM itineraries are built upon the basic ProActive mobile object itineraries, we were constrained by the fact that the method to execute on arrival can not contain any parameter. But, it is not very restrictive as while executing this method, the active object can locally trigger the execution of an other method with parameters (when an active object migrates, all its state is preserved).

An *ItineraryManager* is a class that provides some programming functions and as such serves as an interface between the SNM agent and the *ItineraryServer*. Upon creation, the agent provides some information regarding the type of itinerary it will need for visiting the system or network elements (e.g. MIX in the code in figure 2). Then, in order to effectively obtain the itinerary it must follow, it just calls one specific method defined in the *ItineraryManager* class (i.e. *setItinerary* which is in charge of querying `myLocalItineraryServer`), after that, it

[2]use in the broad sense, including the discovery process

only has to initiate the start of its ”visit” of all elements in the itinerary.

At any time, thanks to a method call originating for instance from an other mobile agent or from the GUI running on the host manager, the SNM agent (its *Itinerary-Manager*) may be told to insert into its itinerary a new *Destination*, possibly in front of the itinerary. This is an easy way of forcing an agent to go back home for instance, or to urgently manage an element.

```
public class MyAgent implements java.io.serializable {
  // Triggered for a NodeDestination
  public void echo() {
    System.out.println("MyAgent.echo()");
  }
  // Triggered for a SNMPDestination
  public void snmpOnArrival() {
    // Gets  SNMP parameters from ItineraryManager
    SNMPDestination snmpDest =   (SNMPDestination)
       itiManager.getCurrentDestination();
      // do your SNM job !
  }
 // Prepare and start to follow an Itinerary
 public void start(String myLocalItineraryServer) {
    // Create an ItineraryManager
    itiManager = new ItineraryManager(MIX);
    // Set the home for locating our local
    // network ItineraryServer
    // and prepare for our test an itinerary
    // in order to 'visit' all the networks
    itiManager.setItinerary(myLocalItineraryServer);
    // Ask the ItineraryManager to start the migration
    itiManager.startItinerary();
  }
 public static void main(String args[]) {
    try {
      // Create an Active Object
      MyAgent myAgent =   (MyAgent)
       ProActive.newActive("mgt.agents.MyAgent", null);
      // prepare the itinerary and go !
      myAgent.start(args[0]);
     } catch (Exception e) { e.printStackTrace(); }
  } // main
}// end of class MyAgent
```

Fig. 2. Code Example of a mobile agent and a transparent itinerary usage

V. Performance Evaluation

As itineraries built with our platform can mix remote SNMP management or local SNMP management after a migration on the host, we have studied tradeoffs of such mixing. The tradeoffs depend on the bandwidth of the links that connect networks (in our case, 2 networks), on the size of the mobile agent when it must cross this link in order to reach some other network, on the size of the data collected by the SNMP management function (either a fix number of SNMP variables, or a variable number as when collecting a routing table for instance). Moreover, the purpose of those evaluations is to check that our SNM platform behaves correctly and yields reasonable performances for realistic infrastructures.

A. Benchmark Configuration

In our test-bed, there are two networks (see figure 3). In network A, two computers (`Yate` and `Bourail`) and in the other, eleven computers of different power and capabilities. Those machines are PCs (running Win95, Linux, WinNT, Sco OpenServer, WinNT Terminal Server), Network Printers (HP 4100 and HP 2100) all executing an SNMP agent. In order to make the network bandwidth between the two local LANs varies, we have used a Pentium at 133Mhz (`Bourail`) running a Free BSD operating system with `ip_dummynet` (a bandwidth limiter) [9] (fig 3). As such, we could simulate a 50Kbps up to a 10Mbps link connecting both LANs. Each network is a 100Mbps switched Ethernet LAN.

In order to simulate a bigger configuration, longer itineraries are obtained by increasing the number of round trips. In the case of the itinerary using mobility, each round trip is as follows: one migration at the start to go from the workstation `Yate` to `Koumac` and one at the end to go back to the initial ProActive node on `Yate`. Notice that we do not empty the agent when it comes back on `Yate`, because the purpose is to simulate an itinerary that spans a whole administrative domain compound of several networks interconnected by low bandwidth links. As such, its size will grow. Of course, previous work has already pointed out a possible improvement: empty (or temporarily store) the data the mobile agent is carrying out with it before migrating again [10]. Alternatively, provide an itinerary whose pattern is a star-shape route [11], where the mobile agent migrates back and forth between the central node and the other nodes, just to deliver its results[3]. In our framework, we also might program a remote method call between the agent and the source node, such as to transmit the results before migrating to the next network. However, it is not the purpose here to evaluate those optimizations or alternative traveling patterns. In the case of the itinerary not using mobility, each round trip is as follows: each element mentioned in the itinerary is a *SNMPDestination*, and as such, all SNMP read operations have to be executed from `Yate`, whatever be the number of networks.

B. Comparison between remote SNMP function and mobile agent plus local SNMP function

The first SNMP function we have programmed is to read onto each element mentioned in the itinerary, a fixed number of SNMP variables that lie in the System MIB-II branch (e.g. system.sysDescr).

When the link bandwidth is equal to 50Kbps, as when for instance the host manager is running on a laptop

[3]Such an itinerary type can easily be provided by extending the *ItineraryManager* class

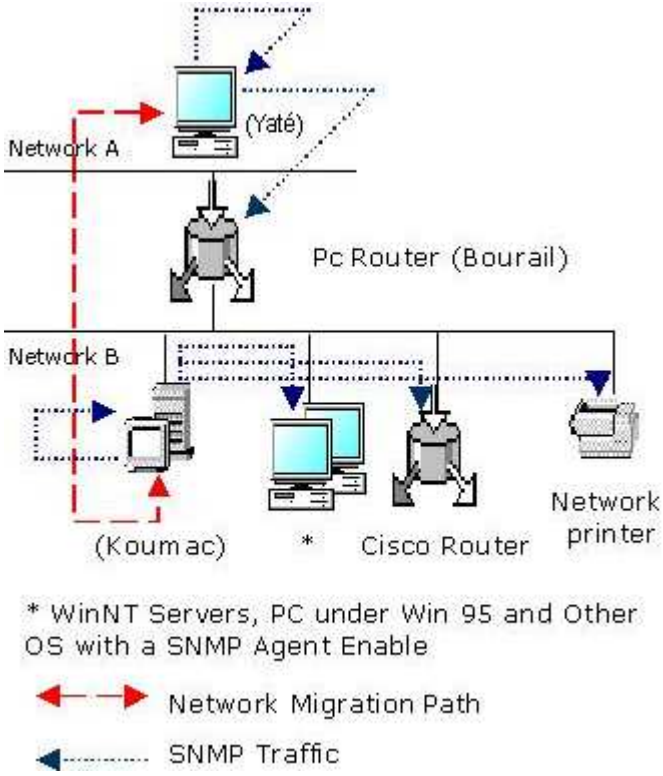

Fig. 3. Network diagram

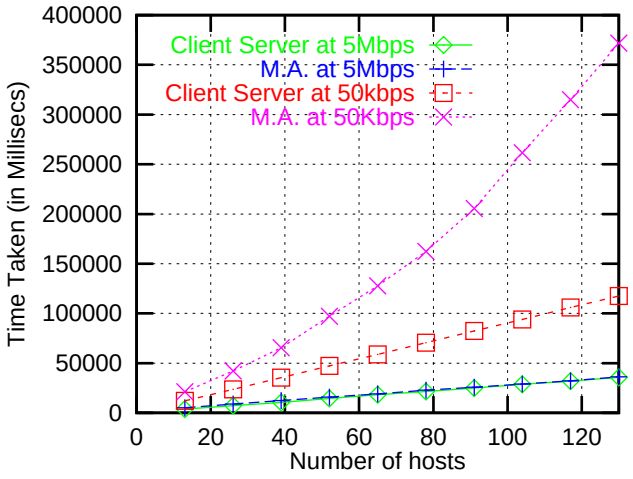

Fig. 4. Retrieval of a small number of SNMP variables per element

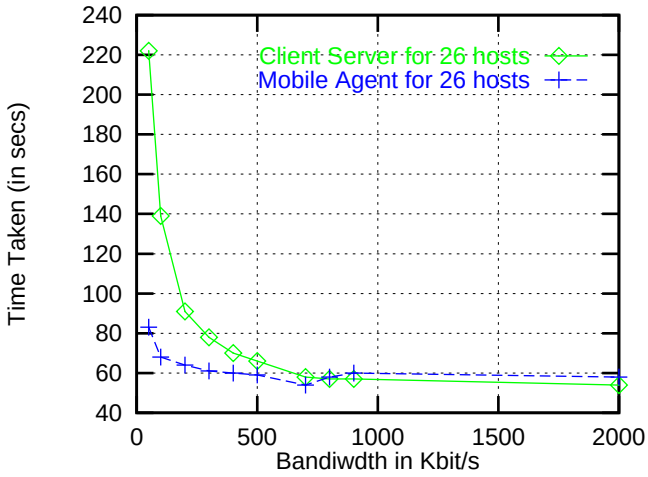

Fig. 5. 26 Destinations in the itinerary; retrieval of more than 7 SNMP variables for each

connected to the network with such a very low bandwidth connection, figure 4 proves that classical SNMP monitoring shows better performances. Indeed, in the mobile agent based experiment, the performance is constrained by the migration performance: the mobile agent migrates twice at each round trip and its size grows with the number of visited hosts. As such, its migration is especially time consuming on a low bandwidth link. The same behaviour can be observed as when the link bandwidth is equal to 100Kbps. However, as soon as the link bandwidth exceeds 1Mbps (as for 5Mbps as shown on figure 4) both kinds of experiments have the same duration: the growing size of the agent is no more a bottleneck for the usage of mobility in the itinerary. If the total amount of information read in the SNMP MIBs increases (e.g. ip.ipRouteTable and ip.ipNetToMediaTable), then the link bandwidth might be the limiting factor even for the pure SNMP-based experiment (see figure 5): in this case, the experiment does not not take advantage of the effect of proximity (proximity means that the read operations in the MIBs of network elements are triggered locally from a host located in the same network instead of remotely, from the initial host for instance).

VI. Discussion and conclusions

Distributed SNM platforms using mobile agents are useful and even efficient in a wide range of hardware configurations, especially when the links interconnecting networks are of low bandwidth, and the output data size of SNM tasks are huge: it is then possible to aggregate and maybe compress and filter those data at the place or in the same network where they have been collected, before sending them back to the manager host. Such considerations and conlusions had already been made for instance in [12], [13], [14] but we had to check that the same behaviour occurs within our SNM platform. Many models have been defined in [14] in order to characterize

SNM applications: the itinerary pattern mixing migration and SNMP retrieval of information we have introduced could correspond to a new model combining the Static Centralized (SC) and Migratory (MG) ones, taking advantage of proximity and locality. Nevertheless, we have not yet studied the usage and effect of distribution in our SNM platform, as evaluated in [14]. However, as the ProActive library does provide the notion of groups of mobile active objects [15], it should be feasible to create a group and to dynamically provide to each peer one itinerary that spans a given part of the network. Then, each peer would independently follow its own itinerary, and peers might also be able to communicate in order to aggregate, correlate and exchange the collected data. This pattern of SNM can be modeled by an extension of the Static Delegated (SD) model [14], in which an agent collecting information is not static but can migrate towards an other network or towards the manager host, or even within its assigned network if necessary.

Another important focus of our SNM platform is the ease of deployment of the support infrastructure for mobile agents. Indeed, it is very constraining if the requirement is to install and run a mobile agent support for every managed element prior executing any SNM task[4]. Thanks to the itinerary pattern we have introduced, it is not mandatory to meet this requirement. As the dynamic discovery of the topology of the administrative domain executes, running ProActive nodes will be located and registered into *ItineraryServer*s so as to subsequently be included as *NodeDestination*s into mobile agent itineraries. However, if no ProActive node is found on a given network, then, all its elements can still be managed using a classical SC model.

Arguments for isolating the behavioral logic part from the itinerary part of mobile agents, as done here, include the one mentioned in [11]: building an efficient itinerary customized for each different network is a time-consuming and difficult operation without any knowledge of the network. We can add the following argument: traveling along an itinerary that reflects an up-to-date topology is impossible if the itinerary is statically embedded in the agent at programming time or at the time of the SNM platform deployment. Solutions to both problems rely on mobile agents dynamically retrieving their itinerary from some predefined entities (*AgentPool*s and *NavigatorAgent*s in [11], *ItineraryServer*s in the present work), whose task is to manage and combine information about the network. It would be possible to implement the first of the above mentioned arguments by adequate computations within *ItineraryServer*s at the end of each new topology discovery process execution.

[4]However, we must assume that every managed element runs an SNMP agent, if it does not run a ProActive node

References

[1] A. Bieszczad, B. Pagurek, and T. White., "Mobile Agents for Network Management," *IEEE Communications Surveys 1, 1*, 1998.

[2] G. M. Gavalas D., Greenwood D. and O. M, "Advanced network monitoring applications based on obile/intelligent agent technology," *Computer Communications Journal*, vol. 23, no. 8, pp. 720–730, April 2000.

[3] "AdventNet SNMP tools," http://www.adventnet.net, 1998.

[4] J. Baumann, F. Hohl, M. Straber, and K. Rothermel, "Mole – Concepts of a Mobile Agent System," *World Wide Web 1, 3*, 1998.

[5] P. Bellavista, A. Corradi, and C. Stefanelli, "An Open Secure Mobile Agent Framework for Systems Management," *Journal of Network and Systems Management*, vol. 7, no. 3, 1999.

[6] A. Puliafi to and O. Tomarchio, "Using Mobile Agents to implement fexible Network Management strategies," *Computer Communication Journal*, vol. 23, no. 8, 2000.

[7] E. Reuter and F. Baude, "System and network management itineraries for mobile agents," in *4th International Workshop on Mobile Agents for Telecommunications Applications, MATA*, 2002.

[8] D. Caromel, W. Klauser, and J. Vayssière, "Towards Seamless Computing and Metacomputing in Java," *pp. 1043–1061 in Concurrency Practice and Experience, 10(11–13)*, 1998.

[9] L. Rizzo, "Ip dummynet," Dip. di Ingegneria dell'Informazione, Univ. di Pisa, http://info.iet.unipi.it/˜luigi/ipdummynet.

[10] Y. Aridor and D. Lange, "Agent Design Patterns: Elements of Agent Application Design," in *Second International Conference on Autonomous Agents (Agents'98)*. ACM Press, pp. 108–115.

[11] I. Satoh, "A Framework for Building Reusable Mobile Agents for Network Management," in *Network Operations and Managements Symposium (NOMS'2002)*. IEEE Communication Society.

[12] M. Rubinstein and O. Duarte., "Evaluating tradeoffs of mobile agents in network management," *Networking and Information Systems Journal*, vol. 2, 1999.

[13] A. Sahai and C. Morin, *Software Agents for Future Communications Systems*. A.L.G.Hayzelden and J. Bigham (Eds), Springer Verlag, 1999, ch. Mobile Agents for Managing Networks : MAGENTA perspective.

[14] P. Simões, J. Rogrigues, L. Silva, and F. Boavida, "Distributed Retrieval of Management Information: Is It About Mobility, Locality or Distribution ?" in *Network Operations and Managements Symposium (NOMS'2002)*. IEEE Communication Society.

[15] L. Baduel, F. Baude, and D. Caromel, "Effi cient, Flexible, and Typed Group Communications in Java," in *Joint ACM Java Grande - ISCOPE 2002 Conference*.