Decentralized SNMP Management with Mobile Agents

M. Zapf, K. Herrmann, K. Geihs
Johann Wolfgang Goethe-Universität, Dept. of Computer Science
Robert-Mayer-Straße 11–15
60325 Frankfurt/Main, Germany
zapf@vsb.informatik.uni-frankfurt.de

Abstract
The mobile agent paradigm can be employed effectively for the decentralized management of complex networks. Nevertheless, legacy management protocols such as SNMP need to be supported for reasons of migration and efficiency. We show how the integration of the two paradigms leads to a flexibly adjustable decentralized management architecture. *NetDoctor* is a network management application framework that allows the monitoring of hosts in a heterogeneous network. Mobile agents may query local or remote SNMP agents, process the status data, and perform certain corrective actions. The system supports run-time adjustment and reconfiguration. A quantitative analysis underlines the benefits in respect to the network load reduction.

Keywords
Mobile agents, SNMP, decentralized network management, legacy systems

1. Introduction

Network management based on the *Simple Network Management Protocol* (SNMP) [1] leads to a centralized management architecture. The polling-oriented mode of operation of SNMP has obvious drawbacks in respect to performance, scalability and inefficient use of expensive network resources [2]. Compared to SNMP, the ISO CMIS/CMIP standards [3] offer more possibilities of putting intelligence on the managed nodes to overcome the SNMP limitations. However, they have not reached a widespread acceptance among hardware manufacturers and system administrators. In complex networks there is a need for a decentralization of management tasks such that the central management authority is freed from time consuming but relatively unimportant activities. Management by delegation [2] and mobile management agents [4, 5, 6, 7] are attempts to introduce a decentralized management architecture.

Although these new management approaches seem to solve the problems of centralized management architectures, one cannot neglect the strengths of existing management solutions based on “legacy protocols” such as SNMP. The fact should not be underestimated that SNMP is an accepted standard, widely supported by commercial products, that it is an appropriate low-cost, *small-footprint* solution for elements with scarce resources, and that there are many applications out there in practice which are built on top of SNMP. We claim that any new management proposals *must* integrate legacy systems. This has been stated also by others [8, 9] as a general requirement in complex management environments where new equipment meets the installed base of old equipment.

In this paper we show how the “best of both worlds” can be combined into a flexible decentralized management architecture. We employ a general-purpose mobile agent platform and provide a systematic way of integrating legacy management protocols such as SNMP. (CMIP or even CORBA [10] can be treated in a similar fashion.) Our network management system is called *NetDoctor*. It is built on top of the *AMETAS* mobile agent platform [11] that is written entirely in Java.

The paper is organized as follows: In section 2 we will first present a brief sketch of the AMETAS mobile agent platform and the integration of SNMP into AMETAS. Section 3 explains the prototype management framework called *NetDoctor*, discusses some application scenarios, presents a quantitive evaluation of *NetDoctor*, and comments on related work. The last section contains our conclusions and an outlook on future extensions.

2. The mobile agent infrastructure

Mobile agent-based application solutions show benefits in situations where it is appropriate to migrate the application code to the data instead of shipping the data to the application, e.g. for reasons of more efficient bandwidth utilization, decoupling of decentralized activities, or system availability considerations. All of these arguments apply to network management applications. Therefore, the mobile agent paradigm has been proposed for use in management applications [4, 5, 6, 7].

We will describe the basic notions of our agent system to specify the requirements for the implementation of an interface between the agent system and the outside world, in this case the traditional SNMP network management environment.

2.1 The *AMETAS* platform

The network management framework *NetDoctor* that will be described in the next section is based on *AMETAS*[11], a general-purpose agent system that was also created in our research group. *AMETAS* is based on Java 1.1, allowing a homogeneous agent system to be established in a heterogeneous environment. The acronym

AMETAS is derived from "Asynchronous MEssage Transfer Agent System" because the communication between all parts of the agent system is based on an asynchronous message exchange system.

The basic structure is shown in Figure 1. *Places* are the stationary parts of the agent system. At least one place object must be installed and run on a host that wants to participate in the agent system. Places are only equipped with a basic functionality; their most important task is to maintain the message exchange system (*post office*) and to provide an execution environment for *agents*. These objects are usually employed by human users in order to carry out a specific task on their behalf. Although the definition of agenthood differs slightly among the agent researchers, all of them agree on agents showing a certain degree of autonomy. Agents that are intended to change their execution environment are usually called *mobile agents* while those employing information processing with strategies from the Distributed Artificial Intelligence research are often named *intelligent agents*. (Note that *traditional* agents as used in SNMP do not fit well into this view. In order to avoid a confusion of terminology in the following text, we will always use the terms *mobile agent* or *AMETAS agent* for the mobile code entities in the *AMETAS* system, and *SNMP agent* for the entities that provide access to SNMP information as defined in the SNMP standard.) One aspect that distinguishes *AMETAS* from other approaches is the strict autonomy of agents. Agents never offer references to portions of their code to any other object except to the place. This enforces a strict decoupling of the participants of the agent system. Because of the agent mobility and autonomy, it need not (and cannot) be ensured that the communication peer is actually present or willing to listen; instead, each agent can follow its own intentions and consider to react or not. The asynchronous message handling allows the agents to decide if and when to retrieve a message.

In order to start a place, an instance of the `AMETASPlace` class is generated and invoked. After that, the place object is ready to fulfill its tasks like receiving and sending agents, maintaining the mailbox system, communicating with the administrator by means of a control interface, and much more. The extension of the basic place capabilities is achieved by installing *services*. For example, a travel agency could install a booking service at their places; a network administrator would be interested in a service for system management. In *AMETAS*, services are special objects that participate in the message exchange system like agents; however, their behaviour is usually based on a request-response pattern as with servers in the client/server paradigm and not on autonomy.

Unlike agents, services cannot change their location. On the other hand, this allows services to be platform-optimized; they can employ native code for better performance or for gaining low-level access to the system. Services can be trusted like any other application that is installed locally. Mobile agents, on the other hand, may come from unknown places with unknown intentions. Moreover, the user's own agents returning from a trip in the network could have been subject to attacks (like modification). Therefore, *AMETAS* does not allow agents to gain access to the system

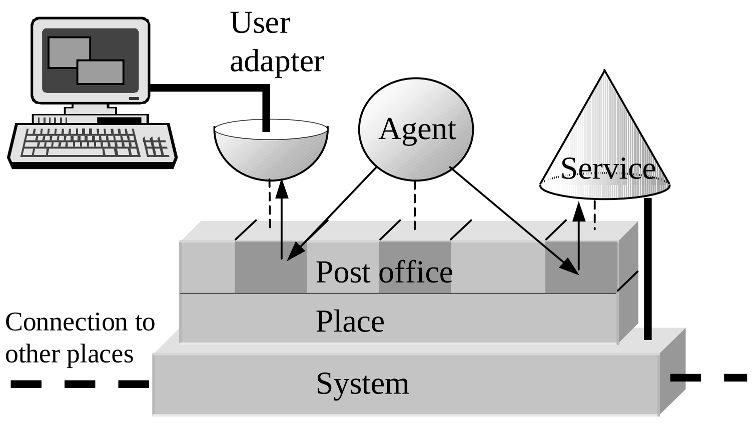

Figure 1 AMETAS basic structure

directly; they have to send requests to the appropriate services. This offers a good chance to employ a powerful security system that prevents unauthorized access to services or resources. *AMETAS* utilizes a complex security system with possibilities for authorization, access control, policies, and encryption [12]. Because of their ability to access entities outside the agent environment, *AMETAS* services can be used to interface to non-agent applications from within the agent system. *NetDoctor* demonstrates how the connection between the agent technology and traditional technologies like SNMP may be achieved with *AMETAS*.

The integration of human users in the agent system is handled by *user adapters*. These objects transform actions of users into an agent-understandable format; in addition, result data can be presented to the user in a suitable way. For example, network monitoring results transmitted by a monitoring agent are converted into a graph that can be shown on the user's display. The appropriate translation of agent messages lies in the hands of the user adapter designer. The major advantage of this user interaction model is the decoupling and symmetry of the communication. User-agent interaction is just another form of inter-agent communication. The ability to directly interact with the user by means of a graphical display, terminal, mouse, or keyboard, is characteristic for user adapters. By employing a conventional user interface, the user adapter can effectively hide the agent-specific application details.

2.2 SNMP integration

Today, SNMP is supported by all major communication and computer equipment manufacturers. Some systems are even shipped with a preconfigured SNMP agent. Thus, new management systems *must* support an interface to SNMP. Regarding the conceptual differences between decentralized mobile agent-based and centralized SNMP-based management, at first glance, it seems to be unreasonable to merge both technologies. Nevertheless, the next sections will show how a careful design can yield a solution with considerably less bandwidth comsumption.

As a first step, we built an SNMP interface, allowing us to compose SNMP messages from different Java objects and to encode these objects into a stream of bytes which can be understood by a standard SNMPv1 agent. Our *Java-SNMP interface* is a class library consisting of classes which model the ASN.1 data structures used in the original SNMP standards. SNMP messages can be easily built from these classes which are able to encode themselves into a BER-compliant byte stream using a recursive mechanism. This enables us to access management data from every SNMP-capable device. The SNMP interface is not specific to *AMETAS*, so it should be possible to employ any other Java SNMP package (e.g. the *AdventNet* package [13]) with some modifications in the calling code. Unlike these products, our interface follows a simple approach by only implementing data structures, leaving the communication mechanisms up to the agent system.

A special *AMETAS* service provides the mobile agents with an interface to the SNMP agents. Mobile agents interact with the service via agent messages and may invoke SNMP `get` or `set` operations from within the *AMETAS* system.

The key idea behind this is the employment of SNMP mainly for collecting data locally instead of remotely. The exchange of a large number of messages between the manager and a SNMP agent over the network is replaced by the migration of a SNMP-enabled *AMETAS* agent which applies SNMP locally. Thus, the network load produced by management applications can be largely reduced. However, with SNMP being designed for the remote collection of data, we can also apply the client/server-style of interaction with SNMP agents.

The employment of a mobile agent platform increases the resource consumption at each host running a place. If a host does not provide sufficient resources it can be unsuitable or even impossible to maintain a place together with numerous mobile agents on it. In order to manage such a device, it suffices to install an SNMP agent on it and have the device managed by an *AMETAS* agent from a nearby place. In this way, management applications based on *NetDoctor* are not only able to manage any SNMP-capable device, but the degree of distribution can also be tailored to the specific needs of the management task as well as to the network being managed.

By providing the SNMP service we manage to seamlessly integrate legacy management frameworks into the *AMETAS* agent system and to employ them in innovative management applications. By exploiting the existing SNMP layer we gain much more than a simple way to collect data:

- The most notable advantage is the full compatibility with older SNMP applications which can be used in addition to or in conjunction with *NetDoctor*.
- Interoperability can be achieved quite easily, e.g. by letting *AMETAS* agents generate SNMP traps for older manager applications.

- Platform-independence is ensured since SNMP is platform-independent due to its widespread usage.
- The degree and granularity of decentralization can be controlled very easily by varying the number of nodes which are remotely managed from one *AMETAS* place, as we will see in section 3.3.
- Decentralized management by mobile agents gets applicable to every device in a network that offers SNMP manageability. There is no need to adapt the devices or the SNMP agent.
- There is no need to design proprietary management interfaces since most manufacturers provide MIB extensions for their products.

While many people would hesitate to change from one management framework to another, the integration of traditional SNMP makes the step towards a new framework (such as *NetDoctor*) less cumbersome.

3. Mobile agents managing the network

NetDoctor is a framework for the design of decentralized network management applications using mobile *AMETAS* agents to solve some classical management problems. The degree of decentralization is dynamically adjustable at runtime by exploiting the agent mobility to control the distance between the agents and the managed resources. Networks can be managed effectively while preserving network bandwidth and facilitating the job of the administrator by automating management as much as possible. *NetDoctor* basically consists of three parts: a SNMP service, a user adapter, and health agents. The SNMP service was already described in section 2.2. With the Java-SNMP Interface, the implementation of this service was straight-forward since its only task is to translate agent messages into SNMP messages and vice versa.

3.1 The user adapter

The user adapter plays a central role in the design of most *AMETAS* multi-agent applications. In our case the user adapter integrates the administrator into the agent system by displaying a GUI that contains a control for every managed network device. This GUI has a look-and-feel like any other GUI displayed by conventional non-agent applications. Each control bears an indicator showing the overall state of the device. If this indicator changes its color to yellow or red, the device entered some kind of critical state. Normally, the overall state of a device is composed of several parameters which could be the CPU load, interface utilization, interface error rates, and so on. By opening a pop-up menu on a device control, the operator can view the parameters that contribute to the overall state of the device. An even more detailed display may be requested by opening the history graph of one parameter.

Through all this, the user does not need to know that *NetDoctor* is an agent-based application, as the user adapter handles all the details involved with sending agents around or exchanging messages. Compared to traditional management applications, the user adapter of *NetDoctor* is relatively simple. Its task is to give the administrator an overview over the current state of the network and to provide him with a means of control over the agents managing the net. He can start new agents, stop agents, and supply them with parameters.

3.2 The health agents

NetDoctor achieves monitoring by the use of so-called *health agents*. These health agents are started by the user adapter as a consequence of the corresponding GUI action taken by the user, and migrate to the node they have to monitor. There they start computing a *health function* which indicates the state or a small portion of the state of the network device. Health functions are a common way to semantically compress large amounts of data into a single index characterizing the status of a network or a system [14]. A health function can be as simple as a ratio computed from two or more MIB variables, resulting e.g. in the error rate of a network interface. On the other side, it can be perceptrons or even more complex neural networks which have to be trained to detect critical situations. The output of such complex structures evaluating potentially hundreds of variables is usually a decision as to whether or not an action should be taken. The big advantage of semantic data compression at the data source is the reduction of network load by only transmitting result values in the case when thresholds are crossed. The effect of this technique can be seen in section 3.4 where a quantitative evaluation of the bandwidth usage is given.

Another advantage of health agents is their ability to autonomously execute actions based on the current state of a device. They can take precautionary steps to prevent critical situations, or possibly repair the system without contacting the administrator. Routine tasks like going through the process table trying to kill misbehaving processes in order to free up system resources can be totally automated. Furthermore, the low additional network load allows an increase of the sample rate. In traditional SNMP applications there is a trade-off between the desired accuracy of monitoring and the produced network load because the network load increases linearly with the sample rate. By only polling locally the proposed health agents can use high sample rates without putting any additional load on the network.

NetDoctor's health agents periodically collect data, compute their function and compare it with the thresholds set by the administrator. If a threshold is exceeded, the value is transmitted, resulting in a change in the administrator's display. Actions for every possible threshold event can be defined in the code of the agent. An action is only triggered if a threshold is exceeded for a certain number of sample intervals so that temporary peaks are filtered out. Even more sophisticated information processing is possible as a benefit of employing a general-purpose programming language like Java for the health agents.

3.3 Building an application on top of the framework

NetDoctor applications normally consist of more than the SNMP service, the user adapter, and some health agents. The mobility aspect is brought in by additional application-specific mobile agents that can interact with customized health agents. The health agents normally remain stationary after their first migration. Specially designed services supply limited access to the system in order to control certain system parameters. For the application discussed in this section we introduce two additional entities:

1. A mobile agent called *Mobile Manager* is capable of reading the process table of the local host via SNMP and killing processes (e.g. zombies) if necessary.
2. A service called `SysAdminService` was designed. Since the killing of processes is not supported by the Unix MIB extension we used, this service had to be built to provide access to the `kill` command. In addition, it enables the mobile manager to send e-mails to users for notification purposes.

Since a specific degree of decentralization is not equally applicable to all areas of network management and to all networks, the situation has to be carefully analysed in order to achieve the best results in terms of desired functionality, bandwidth consumption, fault tolerance and so on. We distinguish between two extremes: In the first case, we consider a local network with relatively few devices connected by high-bandwidth media. For functionality reasons, it could be desirable to install an *AMETAS* system on every host and have health agents monitor each one of them locally. We do not want to include all the intelligence necessary for keeping the hosts healthy on each host, since this would put too much load on them. Therefore, a health agent may request that a Mobile Manager come to a host that displays critical values. When this manager is done and all monitored variables of the host are back to normal, it vanishes until it is requested again. The result is an application with a high degree of decentralization for a fast reaction to critical situations, and a good distribution of management-specific CPU load. It is highly flexible as the health agents monitor only small portions of a system, and thus can be configured separately. This scenario is depicted in Figure 2(a).

In the second case, we consider the same task applied to a large WAN consisting of many LANs interconnected by modem lines or similar low bandwidth connections. We should employ as much intelligence as possible in every LAN in order to reduce the usage of the modem links. To keep the total number of agent places low, we install only one agent place in each LAN. One health agent on this place is responsible for

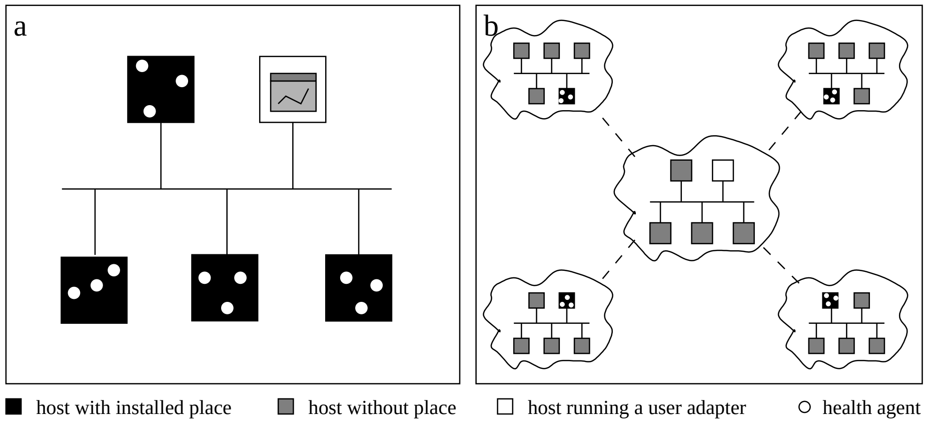

Figure 2 (a) High decentralization with low autonomy in a single LAN. (b) Low decentralization with high autonomy in a WAN.

measuring all CPU loads of the hosts in the local network, another one measures all memory usages and so on. The agents do this by remotely applying SNMP to all hosts in the LAN. Every agent has the intelligence to *cure* an ill device on its own. The computed values are collected and sent over the modem line to the management center at infrequent intervals to provide a statistical survey. If the agents responsible for the respective LAN bear a high autonomy level, we can avoid frequent long-distance modem connections. This WAN scenario is shown in Figure 2(b).

In both cases the administrator is able to remove old agents and send new ones with refined capabilities, and provide new parameters for the installed agents. This makes up for a very flexible application which can be easily configured or even completely changed during run-time which is not possible with most SNMP applications.

3.4 Quantitive evaluation

Consider a network (probably subnetted) with about 100 nodes. For simplicity, we assume that all the nodes are connected via a 10 Mbps Ethernet. We want to monitor the memory usage, the CPU load and the length of the process table on all hosts. The system may be considered running so that we do not care about any setup or installation phase. We need to collect the MIB variables `computerSystemFreeMemory.0`, `computerSystemUserCPU.0`, `processNum.0`, and `computerSystemSysCPU.0`. (These variables are taken from the HP-UNIX MIB extension.)

In the traditional SNMP solution, the variables are remotely collected using a single SNMP `get-request` for every host. This generates exactly 86 bytes per host. 50 bytes of protocol headers are added. The `get-response` to this is about 90+50 bytes long which results in 276 bytes of total data being transmitted for every request. If we set the sample interval to 5 seconds, we get

$$276 \text{ bytes} \cdot 100 \text{ hosts} \cdot 17280 \text{ samples per day} \approx 477 \text{ MB per day}$$

This holds for the monitoring part. Suppose 20 times a day one of the values exceeds a threshold resulting in some kind of action on the part of the management application. The application is interested in 10 MIB objects for every process to spot the problem and take the appropriate action. These values are stored inside the MIB as a process table. To get this data, the manager application generates successive `getnext-requests`, one per process each of which retrieves those 10 MIB variables. With an average of 50 processes per host, this results in about 500 KB of data being transmitted per day in addition to the 477 MB transmitted for monitoring. The processing of the collected data could result in the decision to kill a few processes, adding a few hundred bytes per alarm which are not significant for the overall network load.

When applying *NetDoctor* in the first application sketched in section 3.3, health agents collect and evaluate MIB data locally. A value is transmitted only in the event of an alarm. This means that 40 times a day, when a threshold is crossed in either direction, one packet containing an integer is generated. Each of these Ethernet packets is 96 bytes long (including padding) which yields 3840 bytes per day. The managing agent which is called on every alarm has a size of exactly 2500 bytes. Over the 20 alarms, the transmitted agents add up to 50 KB. About 10 KB are generated by the protocols involved in sending the agents. This summarizes to a total network load of about 64 KB per day. By decreasing the amount of network monitoring load and employing intelligent curing strategies, the ratio of effective management traffic is considerably raised.

3.5 Comparison to RMON

Decentralization of network management by itself is not a really novel aspect. In 1995, a standard for remote network management was established, i.e. the *Remote Network Monitoring Management Information Base* [15]. RMON agents (also called probes) which are installed on a device in a network segment are able to create statistics on the network traffic. They can be polled from a central manager or use *alarms* for notification.

With appropriately designed RMON probes, network load may be considerably reduced compared to standard SNMP, yielding results comparable to our solution. However, there are several arguments in favor of the mobile agent approach.

- The RMON MIB is originally targetted towards monitoring Ethernet LAN segments at the MAC layer. (Special extensions are required for other media as well as for the upper layers). — By communicating with the local SNMP agents, mobile agents may directly process any data offered in the MIB.

- RMON probes are, in effect, programs that need to be written and installed for use in a specific environment. — Mobile agents may not only be dispatched and retracted on demand but also designed to work in different environments.
- RMON configurations are static. — With *NetDoctor*, the degree of decentralization may be adjusted at runtime by exploiting the agent mobility.

3.6 Related work

The use of mobile agent systems in network management tasks has been an active research topic for some time. Conceptually, our approach is different with respect to the clarity of the architectural design, the flexibility and dynamics of decentralization, and the emphasis on the integration of legacy systems. Systems like Perpetuum Mobile Procura (PMP) [4] and INCA [5] also present frameworks for mobile agent solutions which at the moment still lack the ability for a widespread employment. This is mainly due to the implementation work still needed to make all resources accessible to mobile agents. Especially the ambitious concepts of PMP to make networks manage themselves relies heavily on appropriate support from hardware and software manufacturers. Other work in progress [6, 7] only discusses the application of mobile agents in network management in a rather theoretical way, leaving some key questions open that will arise during practical usage. Real implementations are not presented.

4. Conclusions

Several conclusions can be drawn from our work. At first, we confirm the result that decentralization is necessary to avoid high traffic during monitoring. The most important aspect for us is to demonstrate that mobile agent systems can easily and effectively be employed in traditional application areas like network management. Therefore, know-how from the area of autonomous agents can be exploited, ranging from simple configuration tasks to complex decision-making, in order to further enhance network management.

Spatial distribution of *AMETAS* agents within the managed network can solve the scalability problem by reducing network load and by distributing processing load. Cleverly designed *NetDoctor* applications can still maintain control over faulty networks since their intelligence is distributed and autonomous. This improves fault-tolerance considerably.

Although *NetDoctor* is based on our *AMETAS* agent system, the strategies and results should be applicable to any other agent system that supports mobile, autonomous agents. *NetDoctor* integrates innovative, mobile agent-based decentralization techniques with the well-known, well-tested and widespread SNMP technology. It should be emphasized that *NetDoctor* is designed as a framework, not as a ready-to-run product. It is not intended to compete against dedicated network management solutions in terms of quantitative effectivity; instead, it provides an access to the power of mobile agent processing for network management.

A test application was successfully demonstrated at the CeBIT'98 fair. From the conversations with our visitors we believe that there is a high degree of interest in applying the modern agent paradigm to support management tasks. *NetDoctor* is currently modified to cooperate with a complex security system for *AMETAS* that has been implemented, enabling *NetDoctor* to prove its usability in a real-world environment. Furthermore, AI features like KQML communication are currently integrated into the *AMETAS* system, enabling us to build more powerful agents.

Acknowledgments

We thank Helge Müller for inspiring discussions and his contributions to AMETAS.

References

[1] Marshall T. Rose: *The Simple Book, An Introduction to Networking Management*. Prentice-Hall, 1996.

[2] G. Goldszmidt, Y. Yemini: *Distributed Management by Delegation*. 15th International Conference on Distributed Computing Systems, June 1995.

[3] W. Stallings: *SNMP, SNMPv2, and CMIP*. Addison-Wesley, 1993.

[4] G. Susilo, A. Bieszczad and B. Pagurek: *Infrastructure for Advanced Network Management based on Mobile Code*. IEEE/IFIP Symposium on Network Operations and Management (NOMS'98), New Orleans, Louisiana, February 1998.

[5] J. Nicklisch, J. Quittek, A. Kind, and S. Arao: *INCA: An Agent-based Network Control Architecture*. Int. Workshop on Agents in Telecomm. Applications IATA'98, AgentWorld '98, Paris, France.

[6] A. Liotta, G. Knight and G. Pavlou: *Modelling Network and System Monitoring over the Internet with Mobile Agents*. IEEE/IFIP Symposium on Network Operations and Management (NOMS'98), New Orleans, Louisiana, February 1998.

[7] T. White and B. Pagurek: *Towards Multi-Swarm Problem Solving in Networks.* International Conference on Multi-Agent Systems (ICMAS'98), July 1998.

[8] A. Clemm: *SNMP and TL-1: Simply integrating management of legacy systems?* Fifth IFIP/IEEE International Symposium on Integrated Network Management, San Diego, California, USA, May 1997.

[9] A. Bedoyan, A. Clemm: *Experience with the Integration of a Legacy*

Management System into a TMN Environment. IEEE/IFIP Symposium on Network Operations and Management (NOMS ’98), New Orleans, Louisiana, February 1998.

[10] Object Management Group: *The Common Object Request Broker Architecture and Specification, Revision 2.0*. July 1995.

[11] M. Zapf: *Design paradigms in agent-based systems*. Distributed Applications and Interoperable Systems (DAIS ’97), Cottbus, Germany, September 1997.

[12] M. Zapf, H. Müller, and K. Geihs: *Security Requirements for Mobile Agents in Electronic Markets*. Trends in Distributed Systems for Electronic Commerce (TrEC ’98), Hamburg, Germany, June 1998.

[13] AdventNet: *Java SNMPv1 API*. `http://www.adventnet.com/`.

[14] G. Goldszmidt, Y. Yemini: *Evaluating Management Decisions via Delegation*. IFIP/IEEE International Symposium on Network Managemant, April 1993.

[15] S. Waldbusser: *Remote Network Monitoring Management Information Base (RFC 1757)*. Carnegie Mellon University, February 1995.