Intelligent Mobile Agents: Towards Network Fault Management Automation

M. El-Darieby
Carleton University,
1125 Colonel By Drive, Ottawa,
ON, K1S 5B6
Canada
mdarieby@sce.carleton.ca

A. Bieszczad
Bell Laboratories, Lucent Technologies,
2L-325 101 Crawfords Corner Rd, Holmdel,
NJ, 07733
USA
bieszczad@lucent.com

Abstract

Mobile agents, equipped with intelligence, provide a relatively new technology that will help automate Network Management activities, which are becoming increasingly data intensive, thus demanding more direct human manager expertise and involvement.

The research reported in this paper is concerned with the design of an Intelligent Mobile Agent, which accomplishes a set of Network Management tasks delegated to it from the human manager. The agent exploits the mobile code technology to roam the network from one node to another, accessing information from each node, processing this information at each node locally, and carrying the results of this processing during the migration. The agent possesses intelligence that allows it to carry out the tasks without involving the human manager. The objective is to present the manager with a set of conclusions or recommendations rather than large volumes of raw alarm data.

This paper focuses on applying an Intelligent Mobile Agent to automate simple fault management tasks and proposes a framework for realizing this automation.

Keywords

Intelligent Agents, Programmable Networks, Active Networks, Mobile Code, Fault Management, Automatic Network Management

1. Introduction

Traditionally, Network Management activities are performed with direct human involvement. However, today's high-speed heterogeneous networks represent complex and data intensive environments demanding higher levels of human manager expertise and involving increasingly complex overhead. The automation of management activities can minimize the human involvement. This results in reducing management errors, and, in consequence, lowering the cost of network operation and maintenance. Mobile agents equipped with some degree of intelligence provide a relatively new technology that may help, as we will argue in this presentation.

The two main communication protocols used for Network Management, SNMP and CMIP, are characterized by centralization, which is usually considered the source of a low degree of flexibility and scalability in large systems [1]. A network

manager residing on a central station contains most of the management logic and processes the data collected from physically distributed agents. The agents are rigid servers that are closely associated with the hosting network components. The involved management paradigm concentrates most of the processing into a single manager that usually interacts with a large number of agent servers. Hence, these two protocols do not provide the required scalability that is needed in today's predominantly complex networks due to the large number of network components, vast topologies, unpredictable network dynamics, etc.

The basic idea to solve these problems is to bring management intelligence as close as possible to the managed resources. One of the most prominent techniques providing a solution is Management by Delegation (MbD) [8]. It represents a clear effort towards decentralization and increased flexibility of management functionality. Instead of the traditional methods of exchanging client/server messages, the management station can specify a task to be carried out by locating a program (an agent) on involved devices, where the actual execution of the task takes place. Such execution is completely asynchronous, producing a higher degree of parallelism, and thus enabling the management station to perform other tasks. The management station action of delegating a specific function to a remote process is described as *function delegation*; this *autonomy* is what determines the *agency* of the remote program.

Another property that enhances the functionality of the Intelligent Mobile Agent is the ability to travel from one node to another. This capability is termed *mobility*. Code Mobility is defined to be "*the capability to reconfigure dynamically, at run-time, the binding among software components of applications and their physical location within a computer network*" [2]. The roaming program is termed a Mobile Agent.

The third capability from which Network Management systems can benefit is the integration of Artificial Intelligence technologies, which capture the human manager expertise in solving network problems. Thus, the management station delegates tasks to autonomous intelligent code that is capable of making independent decisions on the manager's behalf. The decisions are based on the data that the code analyzes locally at the hosting nodes without human manager involvement.

Combining these three capabilities (i.e., agency, mobility, and intelligence) is a software entity coined an **Intelligent Mobile Agent**. Through the integration of the three properties, many of the Network Management activities including fault management, performance analysis and configuration management can be automated.

In this paper, we are concerned with automating network fault management activities, rather than providing new techniques or algorithms for network fault management. The focus of this research is on building a prototype of an Intelligent Mobile Agent through integrating the following technologies:

The Java Language, which is a platform-independent language that enables the execution of an agent on any network component running a Java Virtual Machine (JVM);

The Mobile Code Toolkit (MCT) [17] that provides the agent with the mobility features, enabling it to migrate from one node to another; and

The Java Expert System Shell (JESS) [5], which enables encoding of the network manager expertise in solving network problems in a LISP-like language derived from CLIPS.

Section II of this paper provides a literature review of intelligent Network Management and the mobile code paradigm. The following section describes the general framework adopted to automate management of network faults. In section IV, the implementation aspects are discussed followed by an introduction to some of the system features. We conclude with our observations and remarks, and a prospectus for the future.

2. Related Work

Fault management refers to efforts made to identify, trace and solve network problems. Network problems can be classified as hardware and software faults that cause elements of the network to behave incorrectly [20].

Typically, the flow of fault management activity is as follows:
Collect messages from network components;
Generate alarms by filtering the messages;
Diagnose the faults that caused the messages by correlating the alarms;
Determine a plan for correction of faults;
Correct the fault; and
Verify that the network problem is eliminated.

These activities imply a number of decisions such as when and where to send a technician, whom to send (with what qualifications), and which equipment is required in order to restore the normal operation of the network. These decisions must be made either by an operator or by some intelligent component that must be integrated with the fault management system.

The automation of the telecommunication industry has made remarkable progress since the Strowger switch began replacing the operator-controlled switchboards. Automated network management systems, among its many promises, can provide network administrators with efficient access to operational information supporting them in decision making. The increased productivity improves the performance of the entire network and minimizes network outages [15]. However, several factors contribute to the complexity of managing network faults. They include:
Various types of control and monitoring are used, because each supplier offers their own network control tools to manage their products; and
Several levels of management personnel are involved; for example, maintenance technicians and information managers.

2.1 Intelligent Network Fault Management

One of the major research efforts to automate network management is the integration of Artificial Intelligence techniques with network management systems. Introducing intelligent components within Network Management frameworks has been the focus of a large number of researchers in the last several years [16]. This is due to the increasing complexity of management activities that are becoming unmanageable with a naturally limited human involvement.

The use of Expert systems in Network Management has been an exploratory area for many researchers. In [7], a part of an Integrated Network Management System was designed to manage network faults. They assumed a hierarchical network architecture where each individual sub-network has a sub-network manager. The problem that they tried to solve was to isolate and diagnose faults in the entire network based on the faults reported from the sub-network managers. Their approach was to organize the knowledge used to manage network faults into a rule-based system. This knowledge was partitioned into four different rule-bases according to their functionality. They specified knowledge bases for classifying and filtering alert messages, localizing the fault to a specific network device, deciding whether previous alert sequences warrant the generation of a new alarm, and providing diagnostic information.

One of the major drawbacks of expert systems is that they tend to be stand-alone systems. Although they provide a means to capture valuable expert knowledge (management policies, management rules, diagnostic knowledge, etc.), they do not help in fully using this knowledge because they are rigid and hard to maintain monoliths [16, 20]. The solution that many vendors proposed was to separate the knowledge and control structure used to solve a problem from the application code. This fostered the idea of "*dynamic rules update"* [11] supporting flexibility and modularity because the rule sets can be easily modified or even replaced. Using the Java language to implement a shell provides an advantage of being able to apply the expert system in many contexts and environments [11]. JESS is an example of such a shell.

Other research is concerned with building hybrid systems combining different Artificial Intelligence approaches. These researchers argue that due to the diverse nature of the tasks of network fault management, it is better to accomplish these tasks with many Artificial Intelligence techniques instead of using just one. They believe that using multiple techniques emphasizes the advantages of each and overcomes the individual disadvantages. One such system is described in [10]. This system exploits the pattern recognition ability of neural networks to identify alarm patterns, and in consequence correlating the alarms. The output of the neural network is the type of the fault. It is fed in turn into a Case-Based Reasoning (CBR) system. The CBR system decides whether the collected evidence is sufficient for it to operate. If no, then more data is collected and processed, or just processed by the neural network to produce an acceptable level of evidence. The CBR component then evaluates the fault identification process and stores the problem solving episode as a new case for future reference.

Although automation results in improved network performance, it also introduces a new level of complexity that requires present staff to update their skills. Occasionally, automation may even require an addition of highly specialized personnel [15]. In many cases, augmenting existing management systems with automated procedures and intelligent tools is a more realistic goal, rather than a vision of a completely automated network management.

2.2 Mobile Code

Network-centric applications usually require two levels of support: 1) the communication infrastructure, which is provided by computer networks, and 2) the computing infrastructure, which is a conglomerate of paradigms and technologies

that are used to build network-centric applications. One technology almost omnipotent in the context of network computing is based on code mobility. Three main paradigms for mobile computations have been identified [6]. These are: Remote Evaluation (REV), Code On Demand (COD), and Mobile Agents. These paradigms differ in how the *know-how*, the *processor* and the *resources* are distributed among the components of the distributed system. The know-how represents the code necessary to accomplish the computation. The resources (i.e. data) are that located at the machine that will execute the specific computation.

In the Mobile Agent paradigm, component A has the know-how capabilities and a processor, but it lacks the resources. The computation associated with the interaction takes place on component B that has a processor and the required resources. For example, a client owns the code to perform a service, but does not own the resources necessary to provide the service. Therefore, the client delegates the know-how to the server where the know-how will gain access to the required resources and the service will be provided. An entity encompassing the know-how is a mobile agent. It has the ability to migrate autonomously to a different computing node where the required resources are available. Furthermore, it is capable of resuming its execution seamlessly, because it preserves its execution state.

Mobile code is important for network-centric systems because it represents an alternate, or at least a complementary, solution to traditional client/server architectures. Such solutions may contribute to a reduction of the overall communication traffic in network [1]. For example, mobile code has the ability to engage with a server locally for searching large databases. The proximity to the server ensures high bandwidth of the communication.

Many programming languages support code mobility. In [6] a list of examples of such languages is described. The list includes Java, TCL-DP, Oblique and Python. The technologies based on Java that facilitate the implementation of mobile agents include IBM's Aglet project, university of Stuttgart Mole project, and Carleton University Mobile Code Toolkit [19].

3. Fault Management Automation: The Framework

Intelligent Mobile Agents improve network manageability by dynamic distribution of management intelligence to the networked devices. Instead of querying each node in a client/server mode to gather the information needed to perform fault management, the management station generates a mobile agent and then dispatches it to the network. The agent's goal is to visit all of the nodes that contain any information relevant to the assigned task. The agent processes the information locally, draws conclusions, undertakes appropriate actions, and either communicates the findings to the dispatcher or carries the results away as it leaves this node. What the agent does is termed as a *semantic compression* of the network component data, which produces a set of hypotheses about the network status. When the agent returns to the network manager, it presents the supervisor with the findings (if it has not done it already). Moreover, relying on the intelligence embedded in the agent, the management station may allow an agent to exercise more freedom. For example, an agent may be allowed to determine the nodes that it should visit dynamically based on the information that the agent gathers in the course of its trek.

The following characteristics of an Intelligent Mobile Agent determine its usefulness in solving the network problems discussed in the preceding paragraph and the introduction. They include:

- *Delegation*: the ability to take responsibility for the task assigned by the manager;
- *Communication*: the ability to interact with other agents and the manager and exchange data with them;
- *Reasoning*: the ability to reason about the current situation, make decisions and undertake appropriate actions; and
- *Autonomy*: the ability to make independent decisions based on the available information. For example, the agent may determine its migration patterns; i.e., the nodes that the agent visits depend on the data collected from the nodes visited previously.

Figure 1 illustrates the framework. Each network component (NC) runs a Mobile Code Daemon (MCD) within a Java Virtual Machine (JVM). Virtual Managed Component (VMC) represents a virtual interface to the managed resources (e.g. MIB objects). The interface provides controlled access to the managed objects of the network element [19]. Through this interface, this framework can be integrated with legacy systems (e.g. SNMP) [17, 19].

The process of performing a certain fault management task by a mobile agent consists of several steps. The first step is to equip the agent with the management intelligence required to accomplish the task. This process is called *functional delegation*. The agent is then injected (step (1) in Figure 1) into the network by shipping it to one of the sub-domain managers [7]. The agent will interact (2) with that sub-domain manager according to its *communication characteristics*. The goal of the interaction is to acquire local facts about the status from the messages received by this sub-manger. The facts may include message severity, the time at which the message was received, and the identifier of the network component that generated the message. The agent then applies its self-contained network management intelligence (*reasoning characteristics*) to classify the facts according to the message severity discarding messages of low severity and asserting others as alarms.

The next step is to filter these alarms eliminating multiple occurrences of the same alarm. Afterwards, the agent correlates (3) the remaining alarms substituting a set of alarms matching a specific pattern with a new alarm. After this task is accomplished at a particular location, the agent migrates (4) to the network components that are sources of the alarms. That is the first aspect of automation: **auto-migration**. The agent, and not the manager, initiates its own migration. This decision is based totally on the information found in the sub-domain manager and the alarm filtering and correlation processes carried out by the agent autonomously. The network manager is not involved at all. This is the essence of the agent's *autonomy characteristics*.

After arriving at a certain location, the agent applies its intelligence in order to diagnose the information about the network component. It attempts (5) to determine the causes of the generated alarms and whether the alarms are false (for example, generated due to other alarms at another network elements). This process represents the second aspect of automation: **auto-diagnosis** (6). As before, the agent initiates and conducts the analysis of the local data autonomously without the involvement of the delegating network manager. The agent continues visiting the nodes as

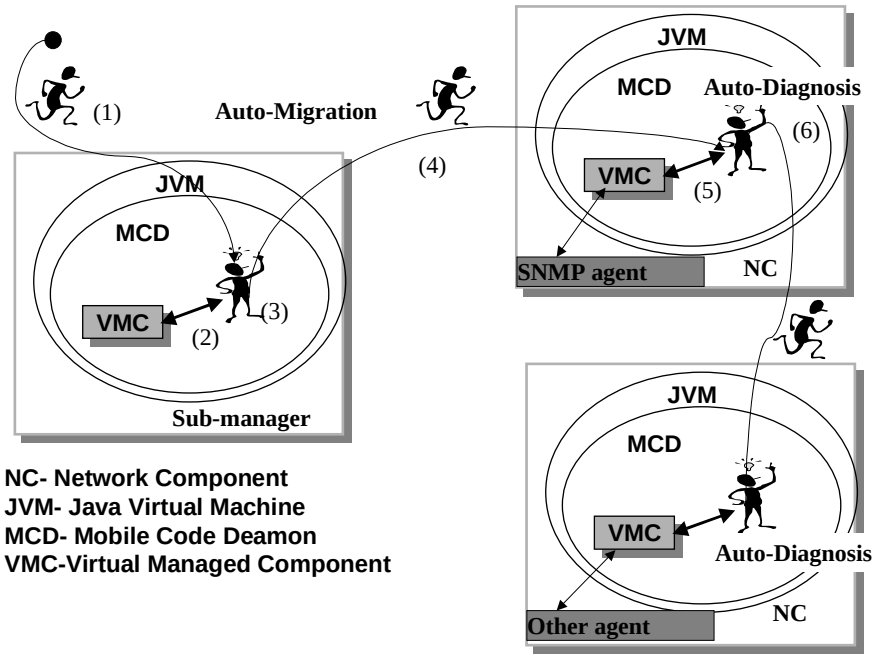

Figure 1: Intelligent Mobile Agent: The Framework.

determined at each location until the end of the logic instructs it to end the migration process.

Ultimately, the agent returns to the dispatching network manager with the results of its trip. The collection and presentation of this information are two important functions of a network management system. A centralized log for all system faults and critical events is a commonly used tool to preserve data for further analysis in the future. In [18], the author states: "*At a minimum, a fault-monitoring agent should maintain a log of significant events and errors. These logs or summaries should be available to authorized managers.*" This information is examined periodically to determine trends affecting network performance. As the result, the error detection capabilities of the network manager are improved. For example, an analytical process well known as *Maintenance Tracking* might reveal that a particular device is a consistent source of errors. It is accomplished through an analysis of a database of accumulated trouble tickets. Trouble ticket information includes the date and time a problem occurred, the specific devices and elements involved, and other related raw data. In addition, the trouble ticket summarizes the diagnosis of the problem and the actions taken to resolve this problem. A comprehensive trouble ticket database can be used in many other ways; for example, for long-term planning and decision-support. In one of the techniques, in order to apply appropriate solution for future occurrences of a problem, all problems are categorized as to determine the most likely cause of a persistent problem [14].

4. Implementation

For the implementation of a prototype, a hierarchical network model was assumed, in which a sub-domain manager is responsible for receiving messages from network nodes in the corresponding network sub-domain. Another assumption is that a network model that represents the physical and logical connectivity information is kept at the sub-domain manager. The third assumption is that status messages have a common format that contains the following parameters: a unique ID, the time the message was received, the severity of the error, and the identification of the source of the message.

The Intelligent Mobile Agent uses Carleton University's Mobile Code Toolkit (MCT) [19] and the JESS expert system shell from Sandia National Laboratories [5]. The MCT provides facilities that enable the agent to migrate from one node to another. The agent acquires the mobility property by extending a Java class called *Netlet* [19]. One of the most important modules of the MCT is the Mobile Code Daemon (MCD). The MCD provides the execution environment for the Intelligent Mobile Agent on a local component. As indicated earlier, every network component runs an MCD within the Java Virtual Machine (JVM). The MCD also provides the Intelligent Mobile Agent with a communication facility that allows for communication with other agents on the same or another component. The MCD provides also the Virtual Managed Component (VMC) concept. Through VMC the Intelligent Mobile Agent is able to communicate indirectly with the managed resources of the network component independent of the underlying platform. Figure 2 shows the sequence diagram for the agent action at each network element of this procedure.

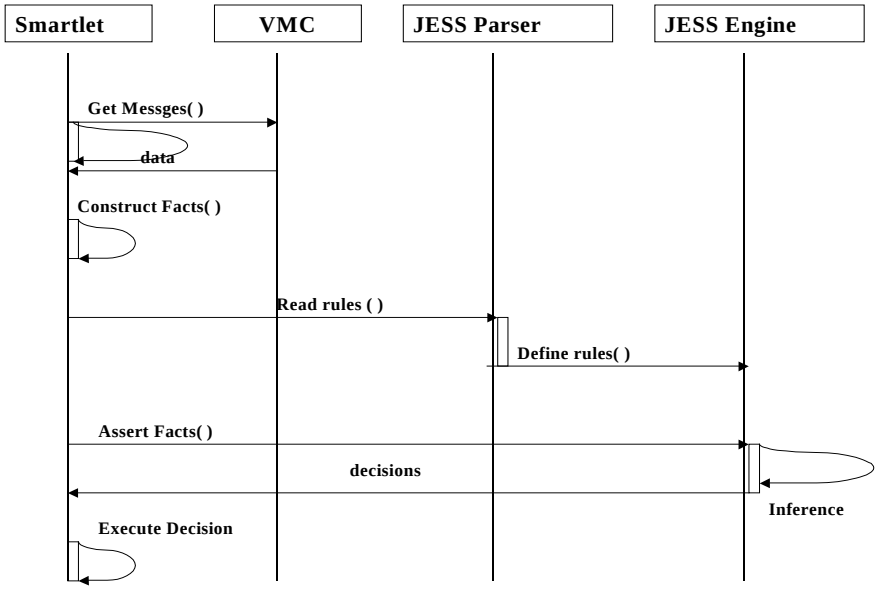

Figure 2: A sequence diagram for the agent implementation

JESS provides an expert system shell, which is the basis of the agent intelligent behavior. Together with the knowledge base, it gives the agents the smarts and a name *Smartlet*.

As it arrives at the network element, the Smartlet communicates with the VMC in order to collect status data about this network component. From the analysis of the data, the Smartlet constructs a set of facts about the status of the network element. Next, a JESS parser object is created to parse a CLIPS-like expert system rule file checking for syntax errors. The rules are linked to the JESS inference engine object that is also created. Then, the Smartlet asserts the facts into the fact list of the inference engine. The inference engine is now ready to run. It produces a set of hypotheses about the status of the network element and the causes of the faults. The decisions and conclusions are handed back to the Smartlet, where they are parsed and executed.

The above functional description is the same for all network elements (i.e., network components and sub-managers). However, the goal of the Smartlet at a network sub-manager differs from the one at a network component. The difference in the objectives implies a difference in the operations applied to the facts about each type of elements. Figure 3 illustrates the differences in the intelligent process applied at sub-managers and network components.

As illustrated in Figure 3(a), at a sub-manager, the first action that the rule base defines is to identify the alert messages, which determine the message category (severity). Then, the messages are filtered to remove the ones that are considered

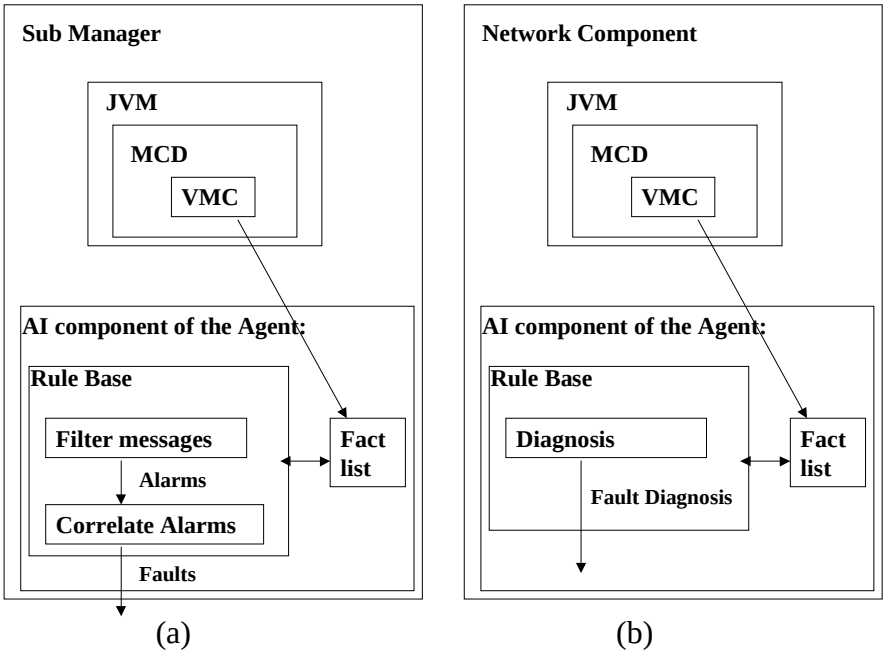

Figure 3: Knowledge application at the sub-manager and network component

trivial or undesirable. This filtration process leads to a detection of actual alarms. The alarms are examined and correlated by JESS in order to localize faults and isolate them from false alarms. This is done in conjunction with the available network connectivity database. The output of this procedure is a set of network addresses of the network components that generated the faults. The Smartlet maintains a list of these addresses and tries to visit each of the involved elements (auto-migration). The agent uses the PING utility to determine if an element is down. If the element is down, this fact is reported to the manager and the address of the component is deleted from the list.

At a network element (Figure 3(b)), the agent collects the status data of this element through a local VMC. The data are asserted as facts in the fact list of the expert system. Using the facts and the rule base, the inference engine of JESS attempts to deduce a possible cause of the alarm (*auto-diagnosis*). For example, the agent may check if the utilization or error rate of a network element exceeds a specific threshold or not. In another place in [18], the author states that "*A good fault-monitoring system should be able to anticipate fault. This involves setting up thresholds and issuing reports when a monitored variable crosses this threshold*."

4.1 Results

As a proof-of-concept, the following preliminary comparison is held. In [12], the author presents an estimate of the cost of the traditional client/server approach of network fault management. The author assumes a network of 30 devices. Each query and response is 100 bytes long including data and header information. This yields 6KB of bandwidth for each polling interval, if polling each device just once is enough to collect the required status information. Polling devices every 60 seconds results in 21.6 MB of bandwidth per each hour of operation. Consequently, in order to poll the status of a single device every 60 seconds the network manager consumes about 720 KB of network bandwidth per each hour.

Intelligent Mobile agents are far more efficient. The size of the prototype agent is 3.25 KB. This includes the JESS classes and the Smartlet class. Our approach is interrupt-driven, so its operation is driven primarily by the status of the network. A problem may occur once per day, or per hour, or it might not occur at all. The network would have to experience 720/3.25~=222 problems per hour for the agent approach to consume a bandwidth equivalent to that of the 60-sec polling.

5. Conclusions

Intelligent Mobile Agents represent a step forward towards automating the management activities of communication networks.

In this paper, we reported research involving building an Intelligent Mobile Agent, which combines network management functionality with intelligence mobility. The Smartlet performs simple alarm classification and correlation at a network sub-domain manager. Then, it migrates to the nodes that caused the alarms. There, it autonomously diagnoses the alarms and determines the cause of faults using a built-in expert system based on JESS. After completing its trip and visiting all nodes, the agent returns to the manager where it presents its findings. A log of the critical events and their causes is kept for a future reference. An Intelligent Mobile Agent has the capability to extract necessary data from the network element over a high

bandwidth, local communication session. This does not consume overall network resources reducing the overall communication traffic. In addition, the Intelligent Mobile Agent has the ability to integrate knowledge from both the manager and the network element. It performs inference asynchronously at network elements allowing the manager to focus on other tasks.

In summary, Intelligent Mobile Agents provide the following advantages:

A flexible and scalable approach to network management.

Enabling a robust response to network problems owing to the intelligent behavior encoded in the agent's knowledge base;

Reduction of administrations overhead and cost owing to function delegation. The human manager only initiates the process, and the agent performs the whole task autonomously;

Reduction of management traffic on the network (no bandwidth-intensive client/server message exchange); and

Providing a convenient programming paradigm. Intelligent Mobile Agents are lightweight (the first version of our Smartlet is only 3.25K), flexible modular entities that can be created, deployed, enhanced and deleted in real time.

Nevertheless, the advantages provided by Intelligent Mobile Agents should be assessed carefully. The environment, in which the agent will be deployed, should be taken into consideration. For example, if the agent is free to determine its own migration patterns, its code may produce a high overhead on network components that have low-processing power. Another obstacle to exploiting code mobility in Network Management is the lack of network devices supporting direct execution of Intelligent Mobile Agent. Currently, Intelligent Mobile Agents require complex run-time environments such as JVM and MCD. Advances in distributed computing and implementing Java virtual machines on chips are making this problem of an increasingly lesser weight.

Future work in our research will be directed towards deploying this technique in actual computer networks. This certainly will require extending the agent properties (intelligence and management components) and adding new components; for example, adding a fault repair component that will enable the agent to suggest repair actions to the network operators. In addition, our future efforts will be to apply Intelligent Mobile Agents in other Network Management domains (i.e. performance and configuration management).

References

[1]. Baldi, M.; Gai, S.; and Picco, G.: "Exploiting Code Mobility in Decentralized and Flexible Network Management." In Proceedings of the First International Workshop on Mobile Agents, Berlin, Germany, April 97.

[2]. Carzaniga, A.; Picco, G.; and Vigna, G.: "Designing Distributed Applications with a Mobile Code Paradigm." In Proceedings of the 19th International Conference on Software Engineering, Boston, May 1997.

[3]. Chess, D.; Grosof, B.; Harrison, C.; Levine, D.; Paris, C.; and Tsudik, G.: "Itinerant agents for Mobile Computing." IBM Research Report RC 20010, IBM, March 1995.

[4]. Dejan, M.; and Guo, K.: "Mobile Agents: Architectural Specifications." The

open group, Open Software Foundation, October 1996.
[5]. Friedman-Hill, E.J.: "JESS: The Java Expert System Shell." http://herzberg.ca.sandia/Jess
[6]. Ghezzi, C.; and Vigna, G.: "Mobile Code Paradigms and Technologies: A Case Study." In Proceedings of the First International Workshop on Mobile Agents, Berlin, Germany, April 1997.
[7]. Goldman, J.; Hong, P.; Jermnimon, C.; Louit, G.; Min, J.; and Sen, P.: "Integrated Fault Management in Interconnected Networks." In Integrated Network Management (I), North-Holland, 1989.
[8]. Goldszmidt G.; and Yemini Y.:" Distributed Management by Delegation." In proceedings of 15th International conference on Distributed Computing systems, 1995.
[9]. Grosof, B.; David, L.; Hoi, C.; Coli, P.; and Joushua, A.: "Reusable Architectures for Embedding Rule-base Intelligence in Information Agents." IBM research report, Computer Science, RC20305, December 1995.
[10]. Gruer, D.; Khan, I.; Ogier, R.; and Keffer, R.: "An Artificial Intelligence Approach to Network Fault Management." SRI International, Menlo Park, California, USA.
[11]. Hall, C.: "Java Expert systems Tools." Intelligent Software Strategies, Cutter Information Corp., summer 1997.
[12]. Leinwand A.; and Conroy K.: "Network Management: A Practical Perspective." 2nd Edition, Addison-Wesley, 1996.
[13]. Lingnaw, A.; and Oswald, D.: " An Infrastructure for Mobile Agents: Requirements and Architectures." Telematic, Frankfaurt ain Main, Germany
[14]. Muller N.: "Network Planning, Procurement and Management." McGraw Hill series on computer communication, 1996.
[15]. Muller N.; and Davidson R.: "LANs to WANs: Network Management in the 1990's." Artech house, Boston-London, 1990.
[16]. Patel, A.; McDermatt, G.; and Mulvilill, C.: "Integrated Network Management and Artificial Intelligence." In Integrated Network Management (I), North-Holland, 1989.
[17]. Schramm, C.; Bieszczad, A.; and Pagurek, B.: "Application-Oriented Network Modeling with Mobile Agents." Presented at the IEEE/IFIP Network Operations and Management Symposium NOMS'98, New Orleans, Louisiana, February 1998.
[18]. Stallings, W.: "SNMP, SNMPv2, and CMIP: The practical Guide to Network Management Standards." Addison-wesley publishing, 1993.
[19]. Susilo, G., Bieszczad, A. and Pagurek, B. (1998), Infrastructure for Advanced Network Management based on Mobile Code, in Proceedings of the IEEE/IFIP Network Operations and Management Symposium NOMS'98, New Orleans, Louisiana, February 1998.
[20]. Yamahira, T.: "Unified Fault Management scheme for a Network Troubleshooting Expert System." In Integrated Network Management (I), North-Holland, 1989.