Paradigms for Mobile Agent-Based Active Monitoring of Network Systems*

Anand Tripathi, Tanvir Ahmed, Sumedh Pathak, Megan Carney† and Paul Dokas‡
{tripathi, tahmed, spathak, mcarney, dokas}@cs.umn.edu
Department of Computer Science
University of Minnesota, Minneapolis MN 55455

Abstract

We present here a framework together with a set of paradigms for mobile agent based active monitoring of network systems. In our framework mobile agents are used to perform remote information filtering and control functions. Such agents can detect basic events or correlate existing events that are stored in a database to enforce system policies. A system administrator can securely modify the monitoring policies and information filtering functions of its agents, or install new agents at a node. The framework presented here includes monitor, subscriber, auditor and inspector agents. The policies and itineraries of these agents can be modified dynamically. In response to certain trigger events agents may change their itineraries to correlate event data. We present here a set of experiments that we have conducted using the Ajanta mobile agent system to evaluate and demonstrate the capabilities of our mobile agent framework.

1 Introduction

Managing and monitoring large networks with hundreds of computers has become a challenging and tedious task for today's system administrators. A typical computing infrastructure in a medium to large-scale organization contains many nodes, possibly of different kinds, organized into multiple local-area networks and administrative domains. Administration functions require periodic upgrading of software as well monitoring of user activities at various nodes to defend against potential attacks by miscreants. The complexity of monitoring large organizational networks — with different kinds of hardware/software components frequently added to the environment or some

*This work was supported by National Science Foundation grants ANIR 9813703, EIA 9818338, and ANI-0087514.

†The participation of Sumedh Pathak and Megan Carney was supported by REU (Research Experience for Undergraduates) funds with NSF grant ANIR 9813703.

‡Paul Dokas is a system security administrator at the Univeristy of Minnesota

existing ones periodically upgraded — requires new approaches of building system monitoring protocols and functions. The mobile agent technology offers several unique capabilities to address the challenges in this area.

A mobile agent represents a program capable of migrating from one node to another in a network to perform certain designated tasks [7]. The ability to migrate code and processing functions to a remote node offers the potential benefits of reduced network traffic and bandwidth requirements. The use of mobile agents for network management has been proposed and investigated by several researchers in the recent years with the primary goal of reducing network traffic and building scalable systems [2, 3, 4, 11].

This paper presents a set of paradigms for using mobile agents in monitoring and managing nodes in an organization's local networks. Our research in the application of mobile agents in network monitoring systems is motivated by many factors. Many large-scale computing environments, such as university campus networks, tend to be relatively open. It is desired for a system administrator to actively monitor the environment for suspicious activities. Large distributed systems need dynamic and scalable architectures for monitoring. Dynamic structures are needed to support changes to policies for monitoring, collection, and processing of information at all levels of a system's organizational hierarchy. It should support definition of new event types and installation of specific detection mechanisms at target nodes. It should be possible to install a new monitoring agent at a node, change an existing one, or within a domain, update the existing event notification policies to implement new data management structures. It should also be possible to enforce desired security policies for event notification and processing functions across different administrative domains. For scalability, the infrastructure should support any desired hierarchical and decentralized organization for information collection and processing. Moreover, the system should support incorporation of new correlation and search functions across different event databases.

Network monitoring systems based on SNMP[5, 15] or periodic execution of script-based cron jobs tend to be largely static and require tedious procedures to define new monitoring policies. The SNMP agents provide a limited and fixed set of functions.

The primary motivation underlying the work presented here is to demonstrate that a mobile agent based management system is a natural fit to meet the above mentioned requirements. Such systems can be used in a network, in addition to an existing monitoring infrastructure such as SNMP, to provide dynamic and extensible functionalities. Network management and monitoring is a promising area where agents can be used to perform remote information filtering, data correlation, and control functions [13, 6]. Agents can be modified remotely to change their monitoring and aggregation policies and functions, and if needed, new agents could be installed at a node to perform functions different from the existing ones. We characterize our agent-based approach as *active monitoring* because it permits easy installation of new monitoring and information filtering functions by launching agents with new functionality to the network nodes.

This paper presents our initial efforts towards building an infrastructure to experiment with future generations of network monitoring and management techniques using mobile agents. Following are the major contributions of the work presented in this

paper:

- A set of paradigms for mobile agent based active monitoring that can be used as building-blocks in a monitoring system. These paradigms exploit code mobility as well autonomous migration of agents.
- A mobile agent based monitoring framework supporting the various paradigms for agent-based monitoring. This system has been implemented using the Ajanta mobile agent programming system [18, 10]. It uses Ajanta's security facilities to build secure monitoring systems.
- A set of experiments using this framework are presented here to demonstrate the use of mobile agents in supporting active monitoring, dynamic extensibility, and correlation of distributed data for detecting compound events.

In the agent-based monitoring system that we have developed, it is possible for an administrator to remotely control the functions of its agents. Agents can be directed to perform high level pattern detection and filtering functions. A monitoring agent responsible for checking a given pattern can autonomously move to different nodes in response to certain trigger events to check for other related events at those nodes. Moreover, agents are able to monitor and generate event notifications based on canonical definitions and representations of events, independent of differences in host operating system mechanisms. In another paper [17] we have shown how an agent can maintain Prolog-based logic databases to perform event data correlation for detecting high level compound events.

In Section 2 we present a framework for agent-based monitoring. This provides an infrastructure to support different paradigms for mobile agent based monitoring, presented in Section 3. Section 4 presents a set of experiments using this framework and the paradigms. In Section 5 we discuss the related work in this area. Section 6 presents the conclusions.

2 A Framework for Agent-Based Network Monitoring

Our agent-based network monitoring framework is designed to support a dynamically extensible environment for monitoring network systems. The purpose and motivation of this framework is to make an attempt to address the issues and requirements for an emerging monitoring system, as described in the previous section, using the mobile agent paradigm. In a large system, it may be inefficient to maintain a central database of all events. Therefore this framework allows one to define any desired policies for decentralized filtering, collection and correlation of event data, and these policies can be changed dynamically. It facilitates designs of hierarchical and distributed organizations for managing event data, but no specific hierarchy is imposed. It also provides definitions for new events in the system, and installation of appropriate monitoring agents at different nodes in the system.

This monitoring framework is implemented using the Ajanta mobile agent system. Each node in the monitored network environment executes a facility (an Ajanta agent server), which allows migration and installation of new agents for event monitoring, data collection, and correlation. We refer to this as the *monitor server*, which can maintain resources to be shared by visiting agents.

2.1 Mobile Agents in Ajanta

Ajanta [18] is a secure Java-based framework for programming mobile agents in the Internet. In Ajanta, mobile agents are *mobile objects*, which can migrate autonomously in distributed environments. Agents encapsulate code and execution context along with data, and they are executed on behalf of a user. The Ajanta system provides facilities to build customizable *servers* to host mobile agents, a set of primitives for the creation and management of agents, security related components and a global naming service. All globally accessible entities are given location-independent names. This name service is implemented to provide secure management of its data.

Security is an integral part of Ajanta design. Ajanta provides components for authentication, public key maintenance, access control, host resource protection, and cryptographic services. In contrast to other widely used mobile agent platforms such as Aglets [9] and Voyager [12], the Ajanta system provides a comprehensive security architecture to protect host resources as well as agents. Each agent is given a set of unforgeable credentials, which contain the agent's name, its owner's name, and a set of privileges granted by the owner. At a remote node an agent executes in a protection domain created corresponding to its owner. An agent-server can enforce any desired policies for accepting remote agents. An agent server grants to a visiting agent restricted access of its local resources, based on the agent's credentials. Agents at two different nodes can communicate with each other using RMI, if permitted by the host servers. It is also possible for an application to remotely invoke methods of its agents. This communication can be authenticated, provided the remote hosting server is trusted. An application can also securely control its remote agents to either terminate their execution or recall them back.

2.2 Event Definition Model

A *basic event* represents some significant change in the state of a resource to be monitored. Higher level *compound events* are derived from other events by applying certain inference rules.

We need a canonical definition and representation of events, independent of any operating system specific details. We keep the events from different nodes in a standard format. Each event has the following attributes: name, category, source-location, and time of occurrence. Additionally, an event may have more attributes as needed. The name field indicates the event type, and the category field indicates the broad class of system functions in whose context the event is generated.

Associated with each event is the definition of a detection procedure. This determines the conditions when an event is detected and signaled. The detection could be based directly on operating system services, and such events can be considered to be basic events. For basic events, the detection rules can be of many different kinds, and they may vary from one OS platform to another. Therefore, in our framework an agent can probe its environment and then execute the appropriate methods. Compound events can be detected and created by correlation of existing event data from the database.

2.3 System Level Architecture

The Ajanta system allows agents to be created and sent to visit a set of hosts on an itinerary. It is also possible to create and launch an agent that creates its own itinerary – as it executes in the network, it makes successive migration requests. All globally accessible entities are given location-independent names, and the Ajanta name service is used to find the current location and other attributes of an entity. This name service is implemented to provide secure management of its data. It also stores public-keys of various entities and principals in the system. Each agent is given a set of unforgeable credentials, which contain the agent's name, its owner's name, and a set of privileges granted by the owner. An agent server grants to a visiting agent restricted access to its local resources, based on the agent's credentials.

Figure 1 shows a typical system organization in our framework. The system administration functions are performed through a set of secure nodes, termed *System Management Stations*. These stations can launch new agents into the system, or can modify or extend the capabilities of existing ones. At a node to be monitored, an agent executes under privileges granted to it by the network monitoring system. For example, based on its assigned privileges it can read certain system log files, check process status, or remove old files from the `/tmp` directory.

2.4 Event Subscription, Notifications, and Correlation

Event communication between agents is based on the publisher-subscriber model. An agent can be sent to a node to monitor a set of events. For each event to be monitored, the monitoring agent has a list of subscribers. A new subscriber can be added or an existing one can be removed from a monitor's subscriber list. This is done through Ajanta's secure RMI communication between agents. A subscriber agent can add (delete) itself to a monitor agent's subscription list, or suspend (resume) its subscription. These functions can also be executed by a system administration agent on its remote monitor agents, and it can also add a new event detection rule and event notification format.

In Figure 1, we have shown two subscriber agents executing at nodes where event databases are to be maintained. These agents are registered as subscribers with the event monitoring agents. How the event databases should be organized and distributed needs to be determined by the system level architecture and policy. For example, all records for events in a subdomain can be maintained at one node. Additionally, event

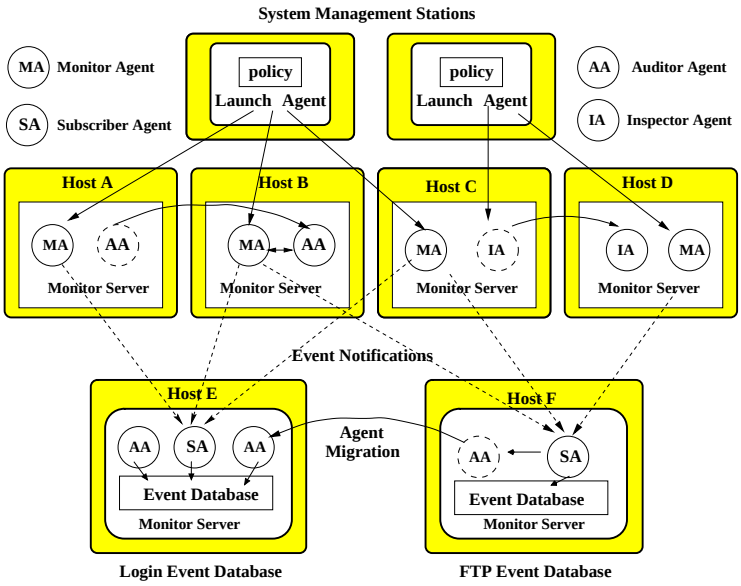

Figure 1: Architecture for an Agent-Based Infrastructure for Network Monitoring

notifications can be recorded at other nodes based on the classifications of the events. For example, one node can collect all successful or failed remote login events, another one could collect all events for device crashes and disk system problems, and a third node could collect all events related to FTP access at different nodes.

Subscriber agents can maintain their event data in a logic database to correlate events received from different hosts. In our present experimental system, this database is maintained using a Java-based Prolog system [8]. An agent can access this database and add events to it as new facts. An agent can also execute a Prolog rule to perform event correlation. New Prolog rules can be added dynamically to this database to add new correlation capabilities.

Every agent also implements the *RemoteControl Interface* defined by Ajanta, for its owner to remotely terminate or recall it. Additionally, an agent may implement the *Monitor Interface* and the *Subscriber Interface* as needed by the specific roles for which it is created. The roles are defined according to different paradigms identified below. The *Monitor Interface* defines methods that allow another authorized agent to add, delete, suspend, or resume a subscriber for an event. It can also add or remove an event from the detection set. The *Subscriber Interface* defines remote methods that are invoked by a monitor agent to deliver notification messages to a subscriber.

3 Paradigms for Agent-based Monitoring Mechanisms

Here we define some paradigms that we have adopted for using mobile agents in our network monitoring framework. These paradigms are centered around the core set of agent roles, namely: monitors, subscribers, information filters.

Transportable Monitor Agents: Such agents are launched to a remote node to reside there permanently to perform functions that require continuous monitoring of certain events. A monitor agent of this kind is given a set of events to be monitored, along with a list of subscribers for each event. A monitor agent supports interfaces for adding a new event detection functionality, besides adding or removing subscribers for events. The system administrator can remotely control such an agent, and possibly replace it with a new one. The structure of such an agent is shown in Figure 2.

A monitoring agent, when transported to a target node, probes its host environment and determines which kind of OS-specific detection mechanisms to use for a given event. It is possible for an administrator to remotely instruct its agent to start monitoring a new kind of event, for which the agent is able to obtain the detection code from a trusted server.

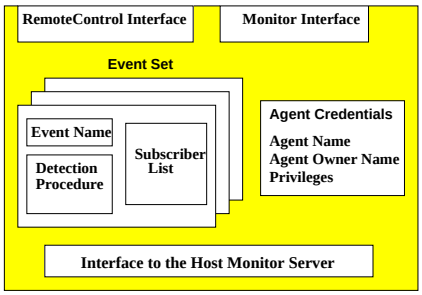

Figure 2: Transported Monitor Agent

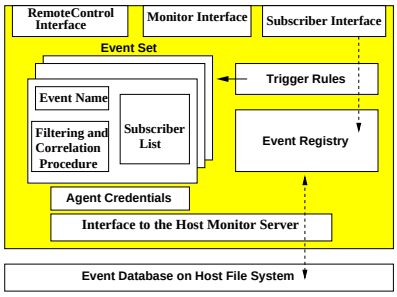

Figure 3: Transported Subscriber Agent

Transportable Subscriber Agents: An agent of this kind is shown in Figure 3. Its primary function is to receive notification of events from a set of monitors, perform any filtering functions, store them in a database in stable storage, and possibly forward them to other subscribers. It receives event notifications through its *Subscriber Interface* and stores them in the local database. An agent of this kind also contains a set of *trigger rules*, which indicate if some additional actions are needed to process a received event, besides storing it in the local database. An event may initiate execution of some correlation functions to detect a compound event representing a pattern of suspicious activities, such as a large number of failed remote login attempts originating from a foreign site. A trigger rule for an event contains a *target* list of compound events whose detection should be performed when a notification for this event is received. The definitions of such compound events together with the associated detection procedures and a set of subscribers are contained in the *event set* of the agent. It is possible for the detection procedure of a compound event to create and send a mobile auditor agent to

other nodes in the system.

Mobile Auditor Agents: The structure of a mobile auditor agent is shown in Figure 4. An agent responsible for event data correlation can also be mobile, carrying with it code for performing event correlation and pattern detection functions using databases at different nodes. The system administrator can define policies determining when a mobile auditor agent is created. Such an agent's itinerary is dynamic, determined as the agent visits different nodes. The function of a mobile auditor is to visit other nodes, as determined during the correlation function execution, to audit the event logs to check if some other corroborating events appear at other nodes, possibly indicating an intrusion or abnormal situation that needs the attention of the security administrator.

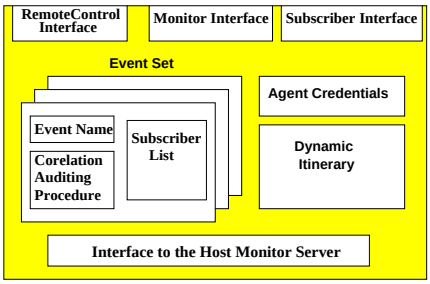

Figure 4: Mobile Auditor Agent

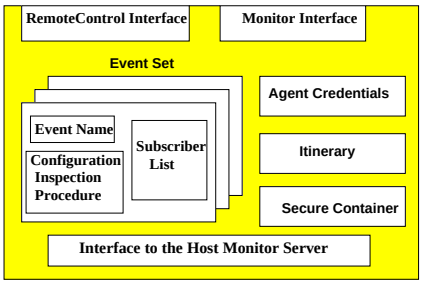

Figure 5: Itinerant Inspector Agent

Itinerant Inspector Agents: The primary role of such an agent is to periodically visit a node and perform certain consistency checks. For example, an inspector would compute the checksums of all important files for critical system services. For this it may retrieve previously computed checksum values from some secure site, or it may carry with it in a tamperproof container facility provided by Ajanta. An inspector agent may also check that the permissions of certain files have not been altered or no new suspicious changes have been made to the system configuration, such as the addition of a new device or hosts. On detecting an abnormal condition, it would send event notifications to the registered subscribers. The tasks of an inspector agent are to periodically check the status of the nodes, or to change the system configuration such as adding software patches for all the nodes. In an environment that consists of hundreds of nodes, usually there are numerous such tasks to be performed, and new ones are added everyday. If a separate agent is launched to each node for each of these tasks, the agent-server on a node would be overloaded. Using itinerant inspector agents, such periodic tasks can be encapsulated and new tasks can be added in an agent in every round.

Though, an inspector agent is programmed with a fixed itinerary, which is repeated periodically, the choice of the next hop from this itinerary can be dynamic based on the state of the nodes the agent has visited. For example, if an inspector agent finds an abnormal condition at a node, such as the file system being full or many run-away processes created by a user, it may postpone its current activity and start checking other

nodes for similar conditions. During an inspection round, such an agent can also decide to migrate to lightly loaded nodes first. An inspector agent can thus prioritize its tasks and change its plans based on the state on the nodes.

4 Experiments in Agent-Based Network Monitoring

In this section, we present five of our experiments to demonstrate the novel utilities of mobile agents in a network monitoring framework as described in the previous section. We are not currently concerned with scalability and have conducted these experiments on a small set of nodes in our laboratory environment. However, these experiments clearly demonstrate the functional capabilities of the agent-based monitoring framework presented here. We conducted a set of experiments to add to our experimental environment, incrementally and dynamically, capabilities for monitoring a few select situations. These situations are of interest to a system security administrator, as they represent a possible threat to the integrity and security of the system. They are as follows:

1. A user has switched to more than two different users. No user in our network is assigned more than two user accounts. A user's ability to log into more than two accounts is either a possible unauthorized sharing of accounts or a system breach showing compromised accounts.

2. A user is logged in at a local domain host and also simultaneously from a remote domain. This activity shows possible misuse of a user's account.

3. Root login activities reach more than a certain threshold number. An unusual number of root login activities require system administrators' attention. These activities should increase an intrusion monitoring system's alert level. These activities may also need to be correlated with other activities, like the above two, to generate higher levels of alarm events.

4. An FTP to a local host from an outside domain fails, and login activities are also observed from that domain. FTP failure is an indication of a potential intrusion step towards finding systems vulnerability, known as *FTP scan*. A domain may be classified as untrusted based on this situation.

5. Periodically check the consistency of files in *bin* directories of all UNIX hosts. Inconsistency of checksum values indicates modifications of files in *bin* directories.

Our experimental system was set up on our lab environment of ten nodes and it was similar to the configuration shown in Figure 1. In our setup, each monitored host executes a monitor server, and provides a database where events can be stored. The syslog file generates lines such as :

"May 1 14:15:25 a.umn.edu in.rlogind[460]: connect from X@b.umn.edu"

The monitor agents, which read these lines, have Perl regular expression[1] patterns which correspond to the strings they want to match. If a match is made, then the corresponding event object is created with information from the string. In this experimental environment, monitor agents are launched to monitor for all login-success, login-failure, and FTP events. A *login event subscriber* agent and an *FTP event subscriber* agent are also installed at nodes E and F, respectively. All the alarm events generated by the subscriber agents are sent to the management stations. The management stations, running under the same user's privilege, can launch other monitor, subscriber, auditor, or inspector agents with new tasks as detailed below for these experiments.

The purpose of the first experiment is to show how host-based attacks can be easily identified by the agent-based framework. When a user switches to a different user, a string, as mentioned above, is generated in the system log file. The monitor agent at the node detects this string through its pattern matching rules, generates the corresponding *switchUser* event, and delivers it to the login subscriber agent at host E in Figure 1. The login subscriber agent contains a trigger procedure for this event, which gets executed upon receiving this event. This procedure queries its event registry for previous *switchUser* events. If a user is found to have switched to more than one other user, the subscriber delivers an alarm event to the management stations.

The second experiment shows how capabilities to monitor new kinds of events can be added through correlation of existing event database entries. For this experiment, we launch an auditor agent to colocate with the login event subscriber at node E in Figure 1. Upon arriving at the monitor server, this agent subscribes the successful login events from the login subscriber. It also has access to the event registry object using the agent server's resource access protocol. Whenever any successful remote login or console login events occur, the auditor queries the local event registry and database for a concurrent presence of the user. If there is a login event of a user from an outside domain, it may indicate a password security compromise.

For the third experiment, another auditor agent is created by a management station and is launched to colocate with the login event subscriber. Whenever this agent sees a root login event (success or failure), it checks for the count of such activities over some specified period of time, and if the count passes a threshold number an alarm event is delivered to the management stations. The management stations raise the monitoring system's alert level based on this event.

The fourth experiment is designed to illustrate how a remote auditor could be created and launched to a remote host for correlation purposes as a part of the actions of a trigger procedure. This experiment requires correlation of events recorded by the FTP subscriber and the login subscriber agents. Whenever the the FTP subscriber agent notices an FTP failure event from an outside domain, it creates an auditor agent and launches it to the login subscriber. At the login subscriber's server, the auditor queries the event registry for remote login events initiated from the same domain that caused

[1] OROMatcher(TM), a Perl5 compatible regular expression library for Java from ORO, Inc. is used. It is available at http://www.oroinc.com.

the FTP failure event. It delivers an alarm event to the management stations when a successful correlation is found.

The last experiment involves use of an itinerant agent with an itinerary containing all the hosts to be monitored. This agent contains the one way hash values for the *bin* directories of the monitoring UNIX hosts in a read-only container. It periodically migrates to a host on its itinerary, computes and verifies the hash values of files in certain directories, delivers an alarm event to the management nodes, and moves to the next host.

The experiments presented above concentrate on the use of mobile-agent based paradigms for network monitoring. These paradigms are the building blocks for a future generation monitoring system, and the experiments proved them to be capable and useful for that purpose. The Ajanta mobile agent system provided the right set of capabilities to implement an infrastructure, which supports the monitoring paradigms. We found that Ajanta provides adequate mechanisms to address the security requirements of this agent-based monitoring architecture. The specific features utilized are: proxy based access control of resources, policy based access control of server resources, agent's credentials based allowance, secure public key distribution by the name registry, and secure containers.

5 Related Work

Network system monitoring is mainly addressed by various SNMP-based network management tools. However, SNMP manages and monitors only network elements. Our goal is to monitor activities of users in a network and their impact on the system consistency. Thus our monitoring system relates more to network intrusion detection systems (IDS). Network monitoring not only provides functionality to monitor users' intentional misuses or intrusions but also monitors inconsistency of systems introduced by any means. Though our work resembles the monitoring task of SNMP-based systems, we try to provide solutions to the shortcomings of SNMP as mentioned earlier.

Only the new generation of IDS, like Emerald [14], Grid [16], are hierarchical, can support both host and network based monitoring, and provides some extensibility of adding new data analysis and reporting modules. However, they are difficult to reconfigure or add new capabilities and functionalities. These IDS usually have to be restarted to make any changes to take effect. Our monitoring system can be dynamically reconfigured or new functionalities can be added in the system by creating and launching new agents with added functionalities to co-exist with the old ones. Moreover, Emerald and Grid do not provide mechanism for detecting what events need to be monitored and where such events should be collected [19]. Correlator agents in our mobile agent based monitoring framework can move to the event publishing hosts for efficiency.

Advantages of using agents have been mentioned in the context of an IDS called AAFID (Autonomous Agents For Intrusion Detection) [1]. Agents can be hierarchically organized such that lower level agents would perform information filtering and

digesting, and then report digested data to the upper layer agents, which makes such systems scalable [1]. Agents can be upgraded when increased functionality is required while keeping backward compatibility. AAFID is programmed in Perl, and it demonstrates the feasibility of an agent-based intrusion detection system, but it lacks a well defined middleware or mobile agent programming environment to fully exploit the benefits of the agent paradigm in such systems. Also the rigid hierarchy in AAFID imposes a delay on the intrusion notification as an intrusion event has to traverse through the intermediate entities in a hierarchy.

The use of mobile agent technology in network management and monitoring is relatively new and its functional capability is recently addressed by other researchers [13, 2, 3]. The primary motivation of most of these efforts is to reduce network traffic. In [11], a hierarchical organization of mobile agent for network monitoring is presented. Our framework is not tied to any kind of hierarchy and can be dynamically modified. We do not address the issues of using our framework to query SNMP agents as in [20], where such an experiment is conducted. However, this paper is also motivated by mobile agents' capability to reduce network load. Contrary to most of the existing mobile agent based monitoring systems, our work concentrates on identifying mobile agent paradigms to realize a secure, extensible, and dynamically configurable monitoring system.

6 Conclusions

This work is motivated by the need of having capabilities to dynamically introduce new kinds of monitoring procedures and correlation functions in today's evolving network systems. We also need capabilities for supporting flexible organizations for decentralized collections and correlation of event data. We have presented here a framework with a set of paradigms for agent-based active monitoring to dynamically add new functionalities. Based on these paradigms, we have built an experimental network monitoring system for our lab environment using the Ajanta system. Through a set of experiments we have demonstrated the functional capabilities of this system to dynamically add monitoring of new kinds of events and correlation functions over data stored at different nodes.

The use of mobile agents provides a richer execution model than just code mobility. In a system with code mobility only, we need to have at each node an entity to download from a remote location code to perform any new monitoring functions. In contrast, the use of the mobile agent paradigm allows an agent to determine the node to which it should migrate to perform certain monitoring functions, as demanded by the present situation. In case of mobile code, with Java 2 security model, one can assign different protection domains to the downloaded code based on its origin. In comparison, an Ajanta mobile agent is executed in a separate protection domain with privileges based on its owner's id. This allows us to define different protection domains for agents with different kinds of functionalities, rather than having all agents to have equal access rights to all system resources. This adds another level of security in the monitoring

system itself. Our work demonstrates that a secure agent programming system such as Ajanta can be effectively used for implementing the framework presented here.

We have primarily concentrated in this paper on the functionalities of an agent based monitoring system using different paradigms. In our future research, we plan to investigate performance and scalability of this system. Our future goal is to experiment in an environment with close to hundred nodes with different types of operating system platforms. We also plan to investigate use of SQL databases instead of Java-Prolog for more efficient event correlation involving large databases.

Acknowledgment
We would like to thank Juergen Schoenwaelder of Technical University Braunschweig, Germany and the anonymous reviewers for their comments.

References

[1] BALASUBRAMANIYAN, J., GARCIA-FERNANDEZ, J. O., ISACOFF, D., SPAFFORD, E. H., AND ZAMBONI, D. An Architecture for Intrusion Detection using Autonomous Agents. Tech. Rep. Coast TR 98-05, Department of Computer Sciences, Purdue University, 1998.

[2] BALDI, M., GAI, S., AND PICCO, G. P. Exploiting Code Mobility in Decentralized and Flexible Network Management. In *Proceedings of the Workshop on Mobile Agents (MA'97) – LNCS 1219* (April 1997), pp. 13–26.

[3] BELLAVISTA, P., CORRADI, A., AND STEFANELLI, C. An Open Secure Mobile Agent Framework for Systems Management. *Journal of Network and Systems Management (JNSM) 7*, 3 (September 1999), 323–339.

[4] BOHORIS, C., PAVLOU, G., AND CRUICKSHANK, H. Using Mobile Agents for Network Performance Management. In *Network Operations and Management Symposium* (2000), pp. 637–652.

[5] BOUTABA, R., GUEMHIOUI, K. E., AND DINI, P. An Outlook on Intranet Management. *IEEE Communications Magazine 35*, 1 (October 1997), 92–9.

[6] CROSBIE, M., AND SPAFFORD, E. H. Defending a Computer System using Autonomous Agents. In *Proceedings of the 18th National Information Systems Security Conference, Baltimore MD, USA* (October 1995), pp. 549–558.

[7] HARRISON, C. G., CHESS, D. M., AND KERSHENBAUM, A. Mobile Agents: Are They a Good Idea? Tech. rep., IBM Research Division, T.J.Watson Research Center, March 1995. Available at URL http://www.research.ibm.com/massdist/mobag.ps.

[8] JIPL: Java Interface for Prolog. Available at URL http://Prolog.isac.co.jp/index_e.html, 2001.

[9] KARJOTH, G., LANGE, D., AND OSHIMA, M. A Security Model for Aglets. *IEEE Internet Computing* (July-August 1997), 68–77.

[10] KARNIK, N., AND TRIPATHI, A. A Security Architecture for Mobile Agents in Ajanta. In *Proceedings of the International Conference on Distributed Computing Systems 2000* (April 2000).

[11] LIOTTA, A., KNIGHT, G., AND PAVLOU, G. Modelling Network and System Monitoring over the Internet with Mobile Agents. In *Network Operations and Management Symposium* (1998), pp. 303–312.

[12] OBJECTSPACE. ObjectSpace Voyager Core Package Technical Overview. Tech. rep., ObjectSpace, Inc., July 1997. Available at http://www.objectspace.com/.

[13] PINHEIRO, R., POYLISHER, A., AND CALDWELL, H. Mobile Agents for Aggregation of Network Mangagement Data. In *First International Symposium on Agents and Applications, and Third International Symposium on Mobile Agents* (October 1999), pp. 130–140.

[14] PORRAS, P. A., AND NEUMANN, P. G. EMERALD: Event Monitoring Enabling Responses to Anomalous Live Disturbances. In *Proceedings of the Nineteenth National Computer Security Conference* (May), 1990, pp. 296–304.

[15] STALLINGS, W. SNMP and SNMPv2: the Infrastructure for Network Management. *IEEE Communications Magazine 36*, 3 (March 1998), 37–43.

[16] STANIFORD-CHEN, S., CHEUNG, S., CRAWFORD, R., DILGER, M., FRANK, J., HOAGLAND, J., K. LEVITT, C. W., YIP, R., AND ZERKLE, D. GrIDS: A Graph Based Intrusion Detection System for Large Networks. In *Proceedings of the 19th National Information Systems Security Conference* (October 1996), National Institute of Standards and Technology, pp. 361–370.

[17] TRIPATHI, A., AHMED, T., PATHAK, S., AND CARNEY, M. Design of a Dynamically Extensible System for Network Monitoring using Mobile Agents. Tech. rep., Department of Computer Science, University of Minnesota, July 2001. http://www.cs.umn.edu/Ajanta.

[18] TRIPATHI, A., KARNIK, N., VORA, M., AHMED, T., AND SINGH, R. Mobile Agent Programming in Ajanta. In *Proceedings of the 19th International Conference on Distributed Computing Systems* (May 1999).

[19] VIGNA, G., AND KEMMERER, R. NetSTAT: A Network-based Intrusion Detection System. *Journal of Computer Security 7*, 1 (1999), 37–71.

[20] ZAPF, M., HERRMANN, K., AND GEIHS, K. Decentralized SNMP Management with Mobile Agents. In *Proceedings of the Sixth IFIP/IEEE International Symposium on Integrated Network Management* (1999), pp. 623 –635.