STANDARDS-BASED INFRASTRUCTURE FOR DYNAMIC EXTENSION OF NETWORK MANAGEMENT SERVICES USING MOBILE CODE

By

SUCHINDRA KATAGERI

A THESIS PRESENTED TO THE GRADUATE SCHOOL
OF THE UNIVERSITY OF FLORIDA IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF
MASTER OF SCIENCE

UNIVERSITY OF FLORIDA

2003

TO MY PARENTS

ACKNOWLEDGMENTS

I would like to extend my sincere gratitude to Dr. Richard Newman for being my chair and guiding and motivating me not just through this thesis but also throughout my graduate life here at University of Florida. I would also like to thank Dr. Michael Frank and Dr. Jonathan Liu for serving on my thesis committee and reviewing my work.

I would also like to extend my deepest gratitude to Dr. Paul Avery, who provided me an opportunity to work on grid systems, which has been the motivation for this thesis.

I would also like to thank my friends for providing me a congenial atmosphere and many ideas.

TABLE OF CONTENTS

page

ACKNOWLEDGMENTS iv

LIST OF FIGURES viii

ABSTRACT x

CHAPTERS

1 INTRODUCTION 1

1.1 Problem Overview 2
1.1.1 Classical Network Management Framework 2
1.1.2 Solutions Using Mobile Agents 3
1.1.2.1 Solution with scripts 4
1.1.2.2 Existing solutions using mobile agent approach 5
1.2 Motivation 7
1.3 Goal and Approach 8
1.4 Terms and definitions used 9
1.5 Organization of the Thesis 10

2 OVERVIEW OF RELATED WORK 12

2.1 Network Management 12
2.1.1 Network Management Functional Areas of OSI 12
2.1.2 Network Management Standards 13
2.1.2.1 Simple Network Management Protocol (SNMP) 13
2.1.2.2 Light Weight Directory Access Protocol (LDAP) 14
2.1.2.3 Extension mechanisms of network management services 14
2.2 Apache Web Server 16
2.2.1 Apache Module to Transfer Binary Data 18
2.2.2 Apache Module to Implement SSL 21
2.3 Protocols 21
2.3.1 Secure Socket Layer (SSL) 22
2.3.2 Hyper Text Transfer Protocol (HTTP) 26
2.4 Related Work in the Area 29
2.4.1 JAMES 29
2.4.2 Perpetuum Mobile Procura (PMP) 30
2.5 Summary 31

3 REQUIREMENT SPECIFICATION 34

3.1 Processes 34
3.1.1 Processes at the WebNMAgent manager 34
3.1.2 Protocol for Transporting the WebNMAgents 35
3.1.3 Processes at theWebNMAgent Server 35
3.1.4 Security 37
3.2 Summary 38

4 DESIGN 39

4.1 Design Issues and Processes 39
4.1.1 Choice of Protocol 39
4.1.2 User Login Process 41
4.1.3 WebNMAgent Receipt and Execution 41
4.1.4 WebNMAgent Storage Scheme 42
4.1.5 Error and Exception Handling 42
4.1.6 Failure and Recovery Handling 43
4.1.6.1 Protocol to update the state information of the WebNMAgent. 43
4.1.6.2 Fault tolerancy when manager fails 43
4.1.6.3 Fault tolerancy when remote site fails 43
4.1.7 User Authentication 45
4.1.8 Message Encryption 46
4.1.9 Authorization 46
4.2 System Design 47
4.3 Security Model 50
4.4 Summary 51

5 IMPLEMENTATION AND TESTING 52

5.1 Choice of Language 52
5.2 Choice of Web Server 52
5.3 Choice of Network Management Agent 52
5.4 Implementation Details 53
5.4.1 WebNMAgent Server 54
5.4.1.1 Apache specific implementation 54
5.4.1.2 WebNMAgent server specific implementation 56
5.4.2 WebNMAgent Manager 57
5.4.2.1 User login 59
5.4.2.2 User interface 59
5.4.2.3 WebNMAgent development environment 60
5.4.2.4 Mechanism to package the WebNMAgent 61
5.4.2.5 Mechanism to receive notifications 61
5.4.2.6 WebNMAgent interaction with the network management agent 61
5.4.3 Security Model 62
5.4.3.1 Authentication 62
5.4.3.2 Encryption 64
5.4.3.3 Authorization 64
5.5 Testing 65

5.5.1 Testing User Login 65
5.5.2 Testing Agent Controls 65
5.5.2.1 Testing create WebNMAgent 65
5.5.2.2 Testing delete agent 66
5.5.2.3 Testing list agents 67
5.5.3 Testing Node Failures 67
5.5.4 Testing Security 67
5.6 Summary 68

6 CONCLUSION AND FUTURE WORK 69

APPENDIX BUSINESS MODEL 71

LIST OF REFERENCES 73

BIOGRAPHICAL SKETCH 75

LIST OF FIGURES

Figure page

1-1 Classical Network Management Framework 3

2-1 Dispatch list of content handlers in Apache 18

2-2 Snippet of httpd.conf that shows how to configure a content handler 18

2-3 Dispatch list in Apache to hook Apache modules 19

2-4 Data structure to hold the HTTP request object in Apache 20

2-5.Configuring Secure Socket Layer in httpd.conf in Apache 24

2-6.Format of HTTP protocol request/response 26

2-7.Sample HTTP request 28

2-8.Sample HTTP response 28

2-9 verview of the JAMES platform 30

2-1 Overview of the integration of JAMES platform with SNMP 32

2-1 Overview of the PMP architecture 33

4-1 System design proposed in the thesis 50

5-1 Configuring and installing the Apache module 54

5-2 Directives of *mod_repository* module 55

5-3 Content handler table in *mod_repository* module 55

5-4 Code snippet configuring the content handler. 56

5-5 Content handler table in *mod_repository* module 56

5-6 Flow chart explaining the WebNMAgent server logic 58

5-7 Sample .htaccess file.....59

5-8 Interface to create WebNMAgents. 60

5-9 Interface to destroy the WebNMAgent 60

5-10 Interface to list the agents.....60

5-11 Snippet of httpd.conf showing client authentication enforcement 63

5-12 Snippet of httpd.conf where server certificates are defined 64

Abstract of Thesis Presented to the Graduate School
of the University of Florida in Partial Fulfillment of the
Requirements for the Degree of Master of Science

STANDARDS-BASED INFRASTRUCTURE FOR DYNAMIC EXTENSION OF NETWORK MANAGEMENT SERVICE USING MOBILE CODE

By

Suchindra Katageri

May 2003

Chair: Dr. Richard Newman
Major Department: Computer and Information Science and Engineering

The mobile code paradigm is opening up new possibilities for processing data, owing to the unique feature of transporting "code" across the network, as opposed to the conventional "data." This allows the intelligence for providing new functionality to be shifted to the client side, while keeping the server lightweight. This helps to centralize the control mechanism for extension of services on remote machines, which is not possible with the traditional client/server approach.

Network management is one of the application domains where there is a need for a flexible mechanism to extend its services dynamically (without disrupting the service). Mobile code paradigm could be used to serve the purpose. This thesis proposes a standards-based approach to implement an infrastructure to use mobile code in dynamically extending network management services. The central idea is to shift the intelligence for extending the network management service to the client side and embedding it in the mobile code. The thesis contends that such an infrastructure could be

built making use of most of the existing infrastructure with minor modifications. It would be more portable, more reliable and would be more acceptable since most of the modules would be standardized and thoroughly tested. It would also reduce learning and development cycles.

The implementation demonstrates the proof of concept for a sub-section of the network management domain, namely extending Simple Network Management Protocol (SNMP) service on remote nodes. The same can be extended for other network management services like Lightweight Directory Access Protocol (LDAP).

CHAPTER 1
INTRODUCTION

With the growth of networks in an organization, the dependency on the network and the applications that utilize it also grows. Hence a management system to ensure its availability becomes necessary. Network management becomes important because it

- improves network availability,
- centralizes control of network components,
- reduces complexity,
- reduces operational and maintenance costs.

Network management encompasses tasks like monitoring performance, having sensors to detect anomalies in the network, running diagnostic tests on detection of failures, etc. To make the system more flexible and to improve network availability, it is required that the network management service itself be able to extend its functionality without affecting the availability of the system. Such a system would further centralize the control of the network management system by allowing itself to be extended from a central point.

Various management protocols and utilities exist to manage the networks and innovative solutions are being employed and proposed to make this function as easy and standard as possible. This thesis proposes a framework to extend dynamically the functionality of a network management service from a central node using mobile agents. This further increases the availability of the system, where the services do not have to be disrupted in order to be extended. This also eliminates user intervention and eases the

task of the network administrator. This helps newer functionality to integrate seamlessly into the system.

1.1 Problem Overview

As discussed in the previous section, with the introduction of new and diverse type of networks (clusters and grids), more network parameters are introduced. Different types of networks might require different parameters to be monitored and configured. The duration of the requirement might also vary. For example, in grid systems, there might be a requirement to monitor the parameters of various applications as they are executed in the grid. The parameters can be job completion status, or percentage job completion. There can be requirements to add sensors to monitor certain conditions and to send notifications to the manager in case of any anomalies. If the network is distributed like the Grid, then to add the extra functionality to the remote site requires user intervention at each remote site. If the functionality required changes often and is not known before, then this process becomes unwieldy. This is the existing problem. The next sub-section discusses different methods and techniques that could be employed to solve this problem.

1.1.1 Classical Network Management Framework

Most of the network management solutions resemble the model illustrated in Figure 1-1. They consist of a manager, which provides the interface for the user to issue queries and obtain results and a remote agent, which listens to the requests issued by the manager and returns the results. The remote agent consists of the required circuitry to retrieve the information requested.

To extend the functionality of the remote agents, there are two methods followed depending on the capability of the remote agents. They can either allow extensions to be done via file or memory. Much of this is explained in the Appendix.

In file-based extensions parameters are added to the configuration file, which is reread by the daemon providing the service. For example in Globus MDS [2], which has an LDAP [3] backend, new configuration is added to the configuration file (schema file). The LDAP daemon is restarted to reread to the newly added configuration data.

In a memory-based extension mechanism, extensions are made in the memory, without having to restart the daemon for reading newly added data. For example, to extend SNMP[4], the protocol AgentX[5] allows one to register and unregister the extension with the main agent. Bookkeeping is maintained within memory that eliminates the need to restart the daemon.

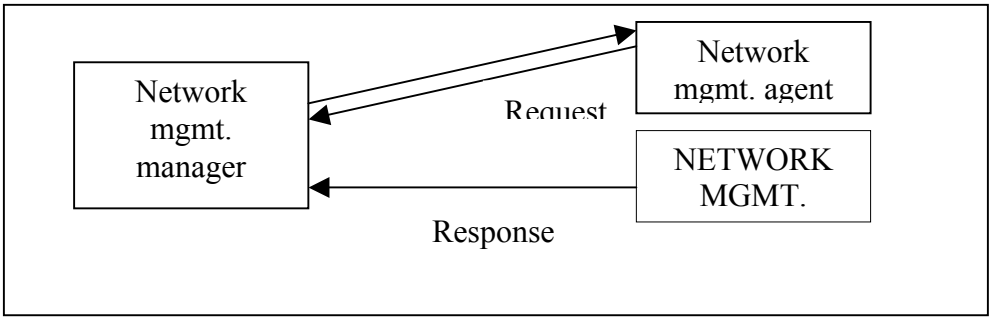

Figure 1-1. Classical Network Management Framework

Scripts based on Secure Shell (SSH) could be used if there is shared memory or a protocol to transfer data. But since this is a distributed grid, the use of shared memory would become infeasible. The use of SSH with a data transfer protocol like the FTP has disadvantages as explained later. The only solution that can be used is the transfer of code to the remote site, which would be otherwise called mobile code.

1.1.2 Solutions Using Mobile Agents

An elegant solution can be suggested based on mobile agents. Mobile agents are active code that can be transported across the network and execute some function on remote machines and then migrate to some other machine. The extension agents can be

embedded in a mobile agent and dispatched to the remote network management agents. A minimal infrastructure can be built to receive and to launch them. The functionality of the extension agents can be decided at run time and they can be removed when not required. This method offers flexibility and seems adequate for the problem at hand.

Application domains in network management. This new paradigm can be applied to a gamut of application domains as listed below.

- Scenarios where there is a dynamic requirement of functionality, which can be different at any time and ad-hoc. The kind of function is not known before hand.
- Scenarios where it is difficult to maintain a synchronous connection.
- Scenarios where a disconnected model is used.
- Scenarios where the functionality of the network is asymmetric. Some parts of the network require different functionalities.
- Larger networks where scalability is an issue. There are a large number of nodes in the network and the server side processing of such a large number of requests puts heavy load on the server. This can distribute the computation.
- Scenarios where it is better to send the code to the data rather than send data to the code in some scenarios, where volume of data transaction is more. This reduces network bandwidth.
- Scenarios where there is a need for a distributed computational model, as in real life. Each piece of code is intelligent to make decisions.
- Scenarios where storage is a constraint. It saves unnecessary memory usage by collecting data only when needed.

Before we propose a solution, it is better to understand the different existing solutions, their capabilities and limitations.

1.1.2.1 Solution with scripts

A solution can be suggested based on scripts, where SSH (Secure shell) could be used to transport control information and FTP (File transfer Protocol) or SCP (Secure

Copy) could be used to transport the data. This requires a lot of bookkeeping at the remote host, including a convention to store the agent, naming convention, maintaining procedures to handle failures and errors, etc., which make it akin to static agents. This has some disadvantages as listed below.

- This is complicated by the fact that two separate protocols are required to transport the code.
- Use of shell scripts affects heterogeneity. Scripts are specific to a platform and cannot be ported across platforms.
- Such a mechanism is not transparent and the user has to be aware of different operating platforms and of the relevant scripting environment.
- It opens up the whole system to the users. This may not be acceptable to the administrators of the remote nodes. The shell should be restricted and should not be able to disrupt the remote system in any way. Giving full control to remote users is not advisable.
- No security can be enforced and the whole environment is open. It allows not only the concerned/appropriate module to execute, but also many other modules, which are not concerned with the application proposed.
- If the environments are not consistent (heterogeneous), then controlling such agents becomes infeasible. Bookkeeping and accounting become very difficult.

The use of scripts is thus not portable across platforms and hence difficult to deploy over the heterogeneous network. It also lacks security as it allows the user to perform other activities other than the intended network management activity.

1.1.2.2 Existing solutions using mobile agent approach

There are a few solutions that provide an infrastructure for network management service extension that use the mobile agent approach:

- The Perpetuum Mobile Procura Project of Carleton University, Canada.[6,7] ;
- The JAMES platform of the Eureka project, at the Univerisity of Coimbra, Portugal [8,9,10].

Perpetuum Mobile Procura project (PMP). PMP is a project at the Carleton University that aims at using mobile agents for network management. It suggests a solution that integrates their mobile agent infrastructure with SNMP [4]. To achieve this, a new SNMP agent (XMS-SNMP Agent) that is extensible and compliant with the existing mobile agent framework is built from scratch. The mobile agents, which act as SNMP extension agents, register and un-register with the main XMS-SNMP Agent using DPI protocol (Distributed Program Interface). The mobile agents are transported across the network using a proprietary protocol. Although this seems to be a good solution, there are certain drawbacks as will be listed. Please refer to chapter 2 for more details.
The merit of PMP is that it is useful in sending diagnostic code and is well suited for telecommunication networks. However the demerits include

- It uses a proprietary protocol for agent transfer, which introduces one more protocol in the market.
- Its dynamic extension mechanism is very specific to SNMP.
- It uses a proprietary SNMP agent, XMS-SNMP Agent. This makes it less portable.
- It does not use the existing modules (proprietary protocol and proprietary agent), hence introduces redundancy.

JAMES. JAMES [8,9,10] is a mobile agent platform developed by the University of Coimbra, Portugal for telecommunication network solutions and network management. The JAMES architecture consists of a module to receive and to launch the mobile agents called the JAMES agency. The JAMES agency consists of a service agent that understands SNMP. The SNMP extension agent functionality is embedded in a mobile agent. All the SNMP extension agents register with this service agent. It also interfaces with the existing legacy SNMP agent and SNMP requests directed towards

legacy SNMP agents pass through the JAMES agency transparently to the underlying SNMP devices.

The JAMES platform also implements an SNMP agent, JAMES SNMP-agent, which arbitrates the SNMP requests from the SNMP applications. Thus requests directed to as legacy SNMP agent are passed through to the SNMP legacy agent, but the requests for SNMP extension agents are directed to the appropriate mobile agent, which is registered to serve that MIB sub-tree.

The merit of JAMES is that it introduces a layer above SNMP agent, which makes it integrate with even non-AgentX supported SNMP agents. The demerits include

- It uses proprietary protocol (on stream sockets) for migration of agents.
- It uses proprietary SNMP agent to receive the SNMP requests, which redirects it to the legacy SNMP agent or the mobile agents. This does not take any advantage of any special features in the legacy system that might have boosted performance. The performance of the JAMES SNMP agent itself becomes the bottleneck.
- It introduces redundancy since the available modules are not used (uses proprietary protocol and proprietary SNMP agent).
- This solution is also specific to SNMP. Mechanisms to handle other network management agents are not discussed.

These solutions use proprietary modules and protocols, hence making the infrastructure heavy and less portable. Their design goals aim at integrating only with SNMP.

1.2 Motivation

New networks are introduced at a fast pace. Newer types of networks like the grids have introduced collaborative networks and virtual organizations, which share the resources and functionality. In a collaborative environment, many people belonging to different organizations but same virtual organization have to work together in achieving

their goals. One of the important functions would be to add new functionality to the system. Also functionality would be required from time to time to detect errors in the distributed system, which would require code on demand. This implies that there is a need for more efficient configuration management that seamlessly integrates new functionality into the network management system.

Currently there is no standard and easy way of doing this. The existing client/server architecture is incapable of adding new functionality to the existing service, due to the fact that the service has to preexist on the server for the client to access it. There are some solutions that use the mobile agent paradigm to provide a flexible solution, but limit themselves to SNMP. This is the motivation to overcome the problems posed by the client/server model and implement a framework that uses most of the existing standard modules and protocol to extend dynamically any network management services and hence increase availability and centralize control of the system.

1.3 Goal and Approach

As discussed in the previous section, many solutions have been suggested for dynamic extension of network management services. Unfortunately they are coupled with SNMP and also use proprietary protocols and modules, which reduces portability and flexibility.

Our solution uses the mobile code approach. Mobile code is same as mobile agent but without the mobility feature. Mobile code does not migrate on the network. It proposes a standards-based infrastructure that reuses most of the existing modules (Apache web server to handle the extension agents) and protocols (HTTP [11]) to provide a dynamic extension mechanism to any network management service (SNMP, LDAP, etc.). This avoids duplication of functionality and is highly portable. It can be used to

extend any network management service as opposed to just SNMP as in the previous solutions. It is very simple, modular and optional. It also implements security mechanisms to make the system more reliable and secure.

Highlights of the proposed solution. Use of mobile code shifts the intelligence from the server to the client. As a result, the server can be lightweight and transparent to the type of service it will provide in future. The functionality can be decided at run time and plugged into the system. This makes it very flexible.

- HTTP is used to transfer the agent between hosts because it is a ubiquitous standard protocol with built in support for security (HTTPS) and mobility. It is well tested with much support available.
- The architecture is modular. The network management agent is decoupled from the mobile agent infrastructure. So any network management agent (SNMP, LDAP, etc.) can be used in the infrastructure.
- It is standards-based and hence reduces learning and development cycles.
- The infrastructure is very minimal with appropriate security mechanisms.

1.4 Terms and definitions used

This section gives a brief introduction of the terms and definitions that are frequently used in the chapters to come.

- ***WebNMAgent:*** This is the mobile code that encompasses the extension agent
- ***WebNMAgent server:*** This is the remote machine in the network that receives the WebNMAgent and executes it.
- ***WebNMAgent manager:*** This is the central node from where the WebNMAgent is dispatched.
- ***Agent:*** This is sometimes interchangeably used for WebNMAgent
- ***manager:*** This is interchangeably used with WebNMAgent manager

- ***Remote server/site:*** This is interchangeably used with WebNMAgent server
- ***SNMP: It is an a***cronym for Simple Network Management Protocol. It is a protocol used to manage TCP/IP networks.
- ***AgentX:*** It is a protocol to extend the SNMP agent dynamically.
- ***SCP:*** It is an acronym for secure copy.
- ***FTP:*** It is an acronym for file transfer protocol.

1.5 Organization of the Thesis

This chapter gave an introduction to the work done in the thesis and the topics that are about to follow. Network management is important in the continuously expanding networks to ensure centralized control of the network, reduce complexity and ensure its availability. Because of the fast paced growth of the diverse types of networks, there is a need for an efficient configuration management, which mainly consists of extending the network management services on remote machines. This thesis suggests a mobile code based solution to provide efficient configuration management.

PMP and JAMES are two of the solutions that use mobile agents to provide configuration management. However they both are very specific to SNMP and both use proprietary modules and protocols. This is the motivation for this thesis to propose a better solution and to overcome those limitations.

Chapter 2 gives a need to know knowledge about the different related topics and the related work going on in field. Chapter 3 describes the requirements of the system. Chapter 4 discusses the system design and the various issues encountered during design. It also discusses the issues regarding security. Chapter 5 discusses the implementation

and testing. Chapter 6 concludes the thesis and discusses about the future work. The appendix discusses the business model.

CHAPTER 2
OVERVIEW OF RELATED WORK

This chapter discusses the background knowledge required to understand the work done. It is divided into four parts. The first part discusses network management, the various components and standards existent in network management. The second part discusses web server specifics and the various modules used. These two components form the core of the solution. The third part discusses briefly the protocols used. The last part discusses the related work in the area.

2.1 Network Management

Network Management is the monitoring and configuring of a network to ensure its availability and service, centralize control of network components, reduce complexity and reduce operational and maintenance costs.

2.1.1 Network Management Functional Areas of OSI

The general requirements of a network management system as defined by the International Organization for Standardization (ISO) is The Open systems Interconnect (OSI) Management Functional Areas. The acronym FCAPS is used to represent the key elements of the ISO definition

- **Fault management.** Fault management includes detection, isolation and correction of abnormal network operation. It provides the means to receive and present fault indication, determine the cause of a network fault, isolate the fault and perform a corrective action.

- **Configuration management** Configuration management encompasses the configuration, maintenance and updating of network components. Configuration management also includes notification to network users of pending and performed configuration changes. It also includes adding new functionality to the system.

- **Accounting management** It includes tracking network usage to detect inefficient network use, abuse of network privileges or usage patterns and is a key component for planning network growth.

- **Performance management** Performance management involves monitoring parameters like network traffic, load and recognize impending performance issues that can cause problems to the users.

- **Security management** Security management includes controlling and monitoring the access to the network and associated network management information. This consists of controlling passwords and user authorization and collecting and analyzing security or access logs. The goal of a network management system is to provide the above functionality in a concise manner that views the entire network as one homogeneous entity.[1]

2.1.2 Network Management Standards

Network management systems depend on defined standards to interface with network devices for monitoring and controlling their configuration, performance and functionality. There are various protocols and methods depending upon the type of network being managed.

- Simple Network Management Protocol (SNMP) [4] for TCP/IP networks.

- Common Management Information Protocol (CMIP) for OSI networks.

- Light Weight Directory Access Protocol (LDAP)[3] for specialized networks like grids.

2.1.2.1 Simple Network Management Protocol (SNMP)

SNMP [4] is the one of the most widely used protocols and is the basis for most modern network management since it provides multi-vendor network management systems the ability to manage network devices from a central location. SNMP includes standard protocols, databases and procedures that are used to monitor and manage devices connected to the network. Nearly all vendors of network-based components, computers, bridges, routers, switches, etc., offer SNMP. Basic components of SNMP are:

Management Station. The user can interact with the system through the management station. It provides the applications to configure, monitor, analyze and control the various components that comprise the network.

Management Agent. An agent is the program that resides on a given network component that responds to requests from the Management Console. It also generates events (traps or notifications) based on configured parameters.

Management Information Base (MIB). The MIB is the local management database maintained by a given network component. There is a standard definition of a MIB for every device that is supported by SNMP. The Management Station monitors and updates the values in the MIB, via the agent. SNMP provides three main functions, GET, SET, TRAP to retrieve, set device values and receive notification of network events.[1]

2.1.2.2 Light Weight Directory Access Protocol (LDAP)

LDAP [3] is the lightweight directory access protocol. It is a database that is optimized for data retrieval, but addition to or modifying the database might be costly. It is based on a client-server model in which a client makes a TCP connection to an LDAP server, over which it sends requests and receives responses.

The LDAP information model is based on the entry, which contains information about some object (e.g., a person). Entries are composed of attributes, which have a type and one or more values. Each attribute has a syntax that determines what kinds of values are allowed in the attribute (e.g., ASCII characters, a jpeg photograph, etc.) and how those values behave during directory operations

Entries are organized in a tree structure, usually based on political, geographical and organizational boundaries. Each entry is uniquely named relative to its sibling entries by its relative distinguished name (RDN) consisting of one or more distinguished attribute values from the entry. At most one value from each attribute may be used in the RDN.

For example, the entry for the person Suchindra Katageri might be named with the "Suchindra Katageri" value from the commonName attribute. A globally unique name for an entry, called a distinguished name or DN, is constructed by concatenating the sequence of RDNs from the root of the tree down to the entry. For example, if Suchindra belonged to the University of Florida, the DN of his UFL entry might be "cn=Suchindra Katageri, o=University of Florida, c=US".

Operations are provided to authenticate, search for and retrieve information, modify information and add and delete entries from the tree.[12].

2.1.2.3 Extension mechanisms of network management services

The above mentioned network management standards form the two types of services which have either a memory based extension mechanism or a file based

extension mechanism. Many other network management services like MonaLisa [13], MDS [2] etc. fall into either one of the above categories.

SNMP Agent extension. Each MIB provides a set of parameters that can be monitored or managed by the management station. SNMP agent can be extended by providing a new MIB and the corresponding implementation. There are many methods to extend the SNMP agent, to provide new functionality. Many toolkits provide library support for the users to develop their own extensions. There are also protocols like AgentX [5], which allow a user to dynamically extend the agent. They even provide the flexibility for the implementations of the agent and the subagent to be in different languages.

MDS (Monitoring and Discovery Service). MDS is the information services component of the Globus toolkit. This is used to monitor grids. The network follows a hierarchical format, namely all the networks in the grid belong to an organization called grid. The organization is subdivided into various administrative domains called virtual organizations and further percolates to the end computer.

MDS uses an LDAP backend to store and process the data and the data are stored in the LDIF (format used by LDAP) format. The data follow LDAP conventions. They are based on attributes. LDAP objects can be defined by classes having attributes. These classes act as templates for the data that will be generated and filtered.

A schema file defines the structure of the data. All types of data generated have an entry in the schema file. To add new data, an entry has to be created in the schema file. An information provider that produces the new data has to be implemented and specified

in a configuration file “grid-info-resource-ldif.conf.” To include the newly configured information provider the LDAP daemon has to be stopped and restarted.

MonaLisa. MonaLisa [13] is a monitoring utility developed at Caltech that provides distributed monitoring service using JINI/JAVA and WSDL/SOAP technologies. It acts as a dynamic service system and provides functionality to be discovered and used by other services or clients requiring such information. The framework can integrate existing monitoring tools and procedures to collect parameters describing computational nodes, applications and network performance.

The system is divided into two parts, remote nodes (farm) and the MonaLisa client. A daemon runs on each remote node that periodically collects information for the configured parameters. A set of such nodes report to a farm monitor through the push mechanism. The MonaLisa client collects information from the farm monitor. It allows for dynamic addition of user-defined modules. The modules to be included are specified in a configuration file. The module can be integrated into the system by including the path of the module while starting the daemon.

2.2 Apache Web Server

Apache is one of the widely used open source web servers. It has a modular architecture that makes extension of its functionality easy. Many developers have developed different Apache modules, which are an indication of the benefits of having modular open source architecture.

Development with Apache. Apache provides a set of APIs for programmers who intend to develop new modules for Apache. The basic concepts involved behind the Apache API are discussed in the subsequent sections.

Handlers, modules and requests. Apache breaks down the request handling into a series of steps namely [14]:

- Uniform Resource Identifiers (URI) to Filename translation,
- Auth ID checking [Authenticating the user],
- Auth access checking [Authorization],
- Access checking other than auth,
- MIME type determination of the object requested,
- Response dispatch to the client,
- Request logging.

These phases are handled by looking at each of the modules configured to find a handler that can handle the request. The handler can do one of the three things:

- Handle the request and return the status as OK,
- Decline the request and send the status as DECLINED,
- Signal an error.

Handlers. A handler from is an internal Apache representation of the action to be performed when a file is called. Generally, files have implicit handlers based on the file type, but Apache 1.1 and above enable the developer to use the handlers explicitly based on either the file extensions or on location. This makes it independent of the file type and hence more flexible. Handlers can either be included in the server or included in a module, or can be added by an "Action" directive.

A brief tour of a module. Every module maintains a dispatch list for the content handlers it provides. It is keyed in by the name of the handler, with the name of the function as the value. Figure 2-1 illustrates the data structure in the Apache module that does this. Apache provides hooks for the modules to register their callback functions with the main module. It is through this callback function that the handler is called. Each module provides a dispatch list for the hooks, which is illustrated in Figure 2.3.

```
/* Dispatch list of content handlers */
static const handler_rec repository_handlers[] = {
    { "repository", repository_handler },
    { NULL, NULL }
};
```

Figure 2-1. Dispatch list of content handlers in Apache

How handlers work. The only argument to the handlers is the *request_req* structure, which contains the request from a client. Each connection generates one such structure.

Data structure holding the request object. The *request_rec* structure contains per-server, per-connection and information about the request. An abridged version of *request_rec* is illustrated in Figure 2-5

The *request_rec* structure has character strings describing attributes of the object being requested. The set of attributes consist of URI, filename, query arguments, content type, content encoding, etc. The handlers thus have access to the request object through the *request_rec* structure.

Configuring *httpd.conf* to include the module. The module can be included in the Apache by inserting the lines illustarted in Figure 2-3.

```
AddHandler repository agentx
```

Figure 2-2. Snippet of httpd.conf that shows how to configure a content handler

2.2.1 Apache Module to Transfer Binary Data

mod_repository [15] is an Apache module written by Brian Aker, which is a freely available in the Apache module database. *mod_repository* can be used to create repository for any type of content. It supports binary data to be transported through HTTP POST methods. It can be configured through these directives.

- *RepositoryRoot:* Tells the server where to store the data.

```
/* Dispatch list for API hooks */
module MODULE_VAR_EXPORT repository_module = {
    STANDARD_MODULE_STUFF,
    NULL,               /* module initializer                  */
    create_mconfig,       /* create per-dir    config structures */
    NULL,               /* merge  per-dir    config structures */
    NULL,               /* create per-server config structures */
    NULL,               /* merge  per-server config structures */
    repository_cmds,       /* table of config file commands       */
    repository_handlers,   /* [#8] MIME-typed-dispatched handlers */
    NULL,               /* [#1] URI to filename translation    */
    NULL,               /* [#4] validate user id from request  */
    NULL,               /* [#5] check if the user is ok _here_ */
    NULL,               /* [#3] check access by host address   */
    NULL,               /* [#6] determine MIME type            */
    NULL,               /* [#7] pre-run fixups                 */
    NULL,               /* [#9] log a transaction              */
    NULL,               /* [#2] header parser                  */
    NULL,               /* child_init                          */
    NULL,               /* child_exit                          */
    NULL                /* [#0] post read-request              */
#ifdef EAPI
   ,NULL,               /* EAPI: add_module                    */
    NULL,               /* EAPI: remove_module                 */
    NULL,               /* EAPI: rewrite_command               */
    NULL                /* EAPI: new_connection                */
#endif
};
```

Figure 2-3. Dispatch list in Apache to hook Apache modules

- *RepositoryFileSize:* This limits the size of files that can be inserted into the repository. By default it is limited to 12megs

- *RepositoryFileLevels:* This number adjusts the number of directories that are created during file storage. Setting the number higher then 32 will cause the module to default to 32. If you do not specify this directive the module defaults to 3.

- *RepositoryTrigger:* Using this you can speficy a URI that will be run whenever the a file type specified by RepositoryType is requested. An environmental variable called REPOSITORY_KEY contains the key.

- *RepositoryType:* This allows specifying mime types that the trigger will be run against.

- *RepositoryHideKey:* If the file type will be captured by a trigger and RepositoryHideKey is set to On, then the key will not be sent out to the requester[15]

```
struct request_rec {

  pool *pool;
  conn_rec *connection;
  server_rec *server;

  /* What object is being requested */

  char *uri;
  char *filename;
  char *path_info;
  char *args;           /* QUERY_ARGS, if any */
  struct stat finfo;    /* Set by server core; st_mode set to zero if no such file */

  char *content_type;
  char *content_encoding;

  /* MIME header environments, in and out.  Also, an array containing environment variables to be
passed to subprocesses, so people can write modules to add to that environment.
   *
   * The difference between headers_out and err_headers_out is that the latter are printed even on error
and persist across internal redirects (so the headers printed for ErrorDocument handlers will have
them).
   */

  table *headers_in;
  table *headers_out;
  table *err_headers_out;
  table *subprocess_env;

  /* Info about the request itself... */

  int header_only;     /* HEAD request, as opposed to GET */
  char *protocol;      /* Protocol, as given to us, or HTTP/0.9 */
  char *method;        /* GET, HEAD, POST, etc. */
  int method_number;   /* M_GET, M_POST, etc. */

  /* Info for logging */

  char *the_request;
  int bytes_sent;

  /* A flag which modules can set, to indicate that the data being returned is volatile and clients should
be told not to cache it.*/

  int no_cache;

  /* Various other config info which may change with .htaccess files.These are config vectors, with one
void* pointer for each module (the thing pointed to being the module's business).
   */

  void *per_dir_config;   /* Options set in config files, etc. */
  void *request_config;   /* Notes on *this* request */

};
```

Figure 2-4.Data structure to hold the HTTP request object in Apache

2.2.2 Apache Module to Implement SSL

This Apache module provides strong cryptography for the Apache 1.3 webserver via the Secure Sockets Layer (SSL v2/v3) and Transport Layer Security (TLS v1) protocols by the help of the Open Source SSL/TLS toolkit OpenSSL that is based on SSLeay from Eric A. Young and Tim J. Hudson. The *mod_ssl* package provides the following features, which is the reason why we chose it.

- Open-Source software (BSD-style license)
- Useable for both commercial and non-commercial use
- Available for both Unix and Win32 (Windows 95/98/NT) platforms
- Support for SSLv2, SSLv3 and TLSv1 protocols
- Support for both RSA and Diffie-Hellman ciphers
- Clean reviewable ANSI C source code
- Clean Apache module architecture
- X.509 certificate based authentication for both client and servers
- X.509 certificate revocation list (CRL) support
- Fully integrated into the Apache 1.3 configuration mechanism
- Assistance in X.509v3 certificate generation (both RSA and DSA) [16]

Configuring *mod_ssl* with *Apache.* *mod_ssl* provides certain directives to be inserted in httpd.conf that help configure the module to suit user needs. Figure 2-5 illustrates a snippet of httpd.conf that has the relevant parameters configured. Apache has the concept of virtual hosts, in that the same interface can listen to two different addresses as configured. The *mod_ssl* configures the SSL connections to be listening on a virtual host with port number 443. Figure 2-5 shows the configuration file.

2.3 Protocols

Two standard protocols are used, namely HTTP and SSL, to transfer the mobile agents from the WebNMAgent manager to the remote server. They are explained briefly.

2.3.1 Secure Socket Layer (SSL)

This document assumes that the reader is familiar with the public-key cryptography. However a brief discussion is made to give the required background to understand the security implemented. SSL was originally developed by Netscape but later universally accepted by the World Wide Web for authenticated and encrypted communication between clients and servers.

Secure Socket Layer protocol

The TCP/IP governs the transport and routing of data over the Internet. HTTP, SNMP etc. are application level protocols that rely on TCP to deliver their data. SSL runs above TCP/IP and below application level protocols and helps the server and client authenticate each other and communicate over an encrypted channel. The SSL includes two sub protocols.

- SSL record protocol, which defines the format used to transmit data.
- SSL handshake protocol, which uses SSL record protocol to exchange a series of messages between SSL-enabled servers and clients, when they first establish an SSL connection. This is explained in the next subsection.

SSL handshake. The SSL protocol uses a combination of public-key and symmetric key encryption. Symmetric key encryption is much faster than public-key encryption, but public-key encryption provides better authentication techniques.

An SSL session always begins with an exchange of messages called the SSL handshake. During the handshake, the server authenticates itself to the client and the client optionally authenticates itself with the server using public-key techniques. Then the client and the server cooperate in the creation of symmetric keys, which will be used for rapid encryption, decryption and tamper detection during the session that follows. The handshake is as follows.

- The client sends its SSL version number, cipher settings, randomly generated data and other information the server needs to communicate with the client using SSL, to the server

- The server sends the server's SSL version number, cipher settings, randomly generated data and other information the client needs to communicate with the server over SSL, to the client. The server also sends its own certificate and, if the client is requesting a server resource that requires client authentication, requests the client's certificate.

- The client uses some of the information sent by the server to authenticate the server, which is explained in the next subsection. If the server cannot be authenticated, the user is warned of the problem and informed that an encrypted and authenticated connection cannot be established. If the server can be successfully authenticated, the client goes on to the next step.

- Using all data generated in the handshake so far, the client (with the cooperation of the server, depending on the cipher being used) creates the pre-master secret for the session, encrypts it with the server's public key (obtained from the server's certificate, sent in Step 2) and sends the encrypted pre-master secret to the server.

- If the server has requested client authentication (an optional step in the handshake), the client also signs another piece of data that is unique to this handshake and known by both the client and server. In this case the client sends both the signed data and the client's own certificate to the server along with the encrypted pre-master secret.

- If the server has requested client authentication, the server attempts to authenticate the client, which is explained in the next subsection. If the client cannot be authenticated, the session is terminated. If the client can be successfully

- Authenticated, the server uses its private key to decrypt the pre-master secret, then performs a series of steps (which the client also performs, starting from the same pre-master secret) to generate the master secret.

- Both the client and the server use the master secret to generate the session keys, which are symmetric keys used to encrypt and decrypt information exchanged during the SSL session and to verify its integrity--that is, to detect any changes in the data between the time it was sent and the time it is received over the SSL connection.

- The client sends a message to the server informing it that future messages from the client will be encrypted with the session key. It then sends a separate (encrypted) message indicating that the client portion of the handshake is finished.

- The server sends a message to the client informing it that future messages from the server will be encrypted with the session key. It then sends a separate (encrypted) message indicating that the server portion of the handshake is finished.

- The SSL handshake is now complete and the SSL session has begun. The client and the server use the session keys to encrypt and decrypt the data they send to each other and to validate its integrity. [17]

```
 ##  SSL Support
 ##
 ##  When we also provide SSL we have to listen to the
 ##  standard HTTP port (see above) and to the HTTPS port
 ##
<IfDefine SSL>
Listen 80
Listen 443
</IfDefine>

    #   SSL Engine Switch:
    #   Enable/Disable SSL for this virtual host.
    SSLEngine on

    #   SSL Cipher Suite:
    #   List the ciphers that the client is permitted to negotiate.
    #   See the mod_ssl documentation for a complete list.
    SSLCipherSuite

ALL:!ADH:!EXPORT56:RC4+RSA:+HIGH:+MEDIUM:+LOW:+SSLv2:+EXP:+eNULL

    #   server Certificate:
    #   Point SSLCertificateFile at a PEM encoded certificate.  If
    #   the certificate is encrypted, then you will be prompted for a
    #   pass phrase.  Note that a kill -HUP will prompt again. A test
    #   certificate can be generated with `make certificate' under
    #   built time. Keep in mind that if you've both a RSA and a DSA
    #   certificate you can configure both in parallel (to also allow
    #   the use of DSA ciphers, etc.)
    SSLCertificateFile /usr/local/Apache/conf/CA/server.crt

    #   server Private Key:
    #   If the key is not combined with the certificate, use this
    #   directive to point at the key file.  Keep in mind that if
    #   you've both a RSA and a DSA private key you can configure
    #   both in parallel (to also allow the use of DSA ciphers, etc.)
    SSLCertificateKeyFile /usr/local/Apache/conf/CA/server.key

    #   Certificate Authority (CA):
    #   Set the CA certificate verification path where to find CA
    #   certificates for client authentication or alternatively one
    #   huge file containing all of them (file must be PEM encoded)
    #   Note: Inside SSLCACertificatePath you need hash symlinks
    #         to point to the certificate files. Use the provided
    #         Makefile to update the hash symlinks after changes.
    SSLCACertificateFile /usr/local/Apache/conf/ssl.crt/ca-bundle.crt

    #   Client Authentication (Type):
    #   Client certificate verification type and depth.  Types are
    #   none, optional, require and optional_no_ca.  Depth is a
    #   number which specifies how deeply to verify the certificate
    #   issuer chain before deciding the certificate is not valid.
    SSLVerifyClient require
    SSLVerifyDepth  10
```

Figure 2-5. Configuring Secure Socket Layer in httpd.conf in Apache

Server authentication.* As discussed before during handshake, the client requests the server to present its certificate. The binding between the public key and the server identified by the certificate is verified accordingly.

- Time validity: The client checks the server's validity period. Certificates whose times are not valid at the present time are regards as expired and rejected.

- Is the Certification authority (CA) issuing the server certificate trusted: Each SSL-enabled client maintains a list of trusted CAs. This list determines which server certificates the client will accept. If the DN on the server certificate matches the DN of a CA on the client's list, then it proceeds to next step. If the issuing CA is not on the list, the client follows the certificate chain till it finds a CA that is on the trusted list.

- The client then uses public key of the CA from the list to validate the CA's digital signature on the server certificate. If the information in the server certificate has changed since the CA signed, or the private key does not match the corresponding public key, then authentication fails.

- The client then checks if the domain name on the certificate actually matches the server's domain name, to prevent man-in-the-middle-attacks. If it matches then the server is successfully authenticated.

Client Authentication.* This is similar to the server authentication process, but for some extra steps. To start with, the server requests the client to send its credentials. It is also required for the client to create a digital signature by creating a one-way hash from the data generated randomly during handshake and known only to the client and the server. The hash of the data is then encrypted with the private key that corresponds to the public key in the certificate being presented to the server.

- The server checks that the user's digital signature can be validated with the public key in the user certificate. If so it is ascertained that the private key corresponds to the public key of the client and the data not tampered.

---

* This section is paraphrased from Netscape Communications Corporation [17]

- Then it checks the time validity, authenticity of the CA following the chains, as in server authentication.

- The server then checks if the user's certificate is listed in the LDAP entry for the user. This is to prevent revoked certificates from using the system. This completes client authentication.

2.3.2 Hyper Text Transfer Protocol (HTTP)

The Hypertext Transfer Protocol (HTTP) is an application-level protocol for distributed, collaborative, hypermedia information systems. HTTP runs over TCP/IP

Structure of HTTP transactions. HTTP uses client/server model. An HTTP client opens a connection and sends a request message to the HTTP server. The HTTP server returns a response message usually containing the resource that was requested. After delivering the response, the connection is closed and no state of the connection is maintained, making HTTP stateless.

The format of the request and response are similar and English oriented. Both contain

- An initial line
- Zero or more header lines
- A blank line
- An optional message body*

For Example.

```
<initial line, different for request and response>
Header1:value1
Header2:value2
<optional message>
```

Figure 2-6. Format of HTTP protocol request/response

It has three parts

* This section is paraphrased from Marshall [18]

<method><request URI><Version>

Methods can be GET, POST, HEAD, PUT, TRACE, DELETE and OPTIONS,

e.g., GET /path/to/file/index.html HTTP/1.0

GET implies request a resource. Even arguments can be passed along with a GET request, using query strings attached to the end of the request.

e.g., http://localhost/jsp/test.jsp?name1=value1&name2=value2

where name1 and name2 are the arguments and value1 and value2 are the parameters. This has some limitations. This transfers the arguments with the URL requested. So it is visible to the outside world and hence is not secure. Also only ASCII characters can be sent, with the maximum limit of 1024.

POST also requests a resource, but the data is obtained through a port other than the normal HTTP port and is not visible outside. Also binary data could be sent on the channel, with no limit on the data being sent. This is the most preferred method used to transfer data.

Initial response line. The initial response line also called the status line has three parts

<HTTP version> <response status code> <English reason phrase>

e.g., HTTP/1.0 200 OK

Header lines. Header lines provide meta-information about the request or response, or about the object sent in the message body.

For Example. From: <email address of the person making the request>

user-agent:<Program name/x.xx>

Message body. It may have a body of data sent after the header lines. In a response the data requested is sent as message body or the text attached with an error. If an HTTP message includes a message body then there are relevant header lines.

e.g.,

Content-type: <MIME type like text/html>

Content-length: <byte count in the message body>

Sample HTTP exchange. To retrieve the file at the URL http://www.timesofindia.com, a socket to the host www.timesofindia.com, port 80 is opened and a request similar to the one illustrated in Figure 2-7 is sent.*

```
GET /path/file.html HTTP/1.0
From: sk3@cise.ufl.edu
User-Agent: Mozilla/3.0Gold
[blank line here]
```

Figure 2-7. Sample HTTP request

The server should respond through the same socket with a response similar to the one illustrated in figure 2-8.

```
HTTP/1.0 200 OK
Date: Fri, 25 Nov 2002 20:48:00 EST
Content-Type: text/html
Content-Length: 154
<html> data……
</html>
```

Figure 2-8. Sample HTTP response

After sending the response, the server closes the socket. For more details about HTTP please refer to RFC 2616.

* This section is paraphrased from Marshall [18]

2.4 Related Work in the Area

This section discusses the related work going on in the area. There are many mobile agent platforms that serve different purposes like IBM's Aglets, Jumping Beans etc., but two platforms have come very close to the proposed solution.

- JAMES project from University of Coimbra, Portugal. [8,9,10]
- Perpetuum Mobile Procura project from Carleton University, Canada. [6,7]

Both are discussed in required detail in the following sub-sections.

2.4.1 JAMES

JAMES is a platform developed as part of the Eureka Project, by University of Coimbra, Portugal.

General architecture of JAMES. The JAMES platform provides the running environment for mobile agents. The central host runs a JAMES manager, while the rest of the nodes in the network run JAMES agency. The agents are written by application programmers and will execute on top of that platform.

Every Network element runs a Java Virtual Machine and executes a JAMES agency that enables the execution of mobile agents. JAMES agents will migrate through these machines to access some data, execute some tasks and produce some reports. The communication among different machines is done through stream sockets. A special proprietary protocol is used to transfer the agents across machines that has its own methods to control mobility.

Integration with SNMP. This architecture intercepts all the SNMP requests originating from the legacy management application (which most of the applications are) and tries to service them through the JAMES SNMP implementation using mobile

agents. If the requests are directed towards a legacy SNMP agent, then it is forwarded to the legacy SNMP agent.

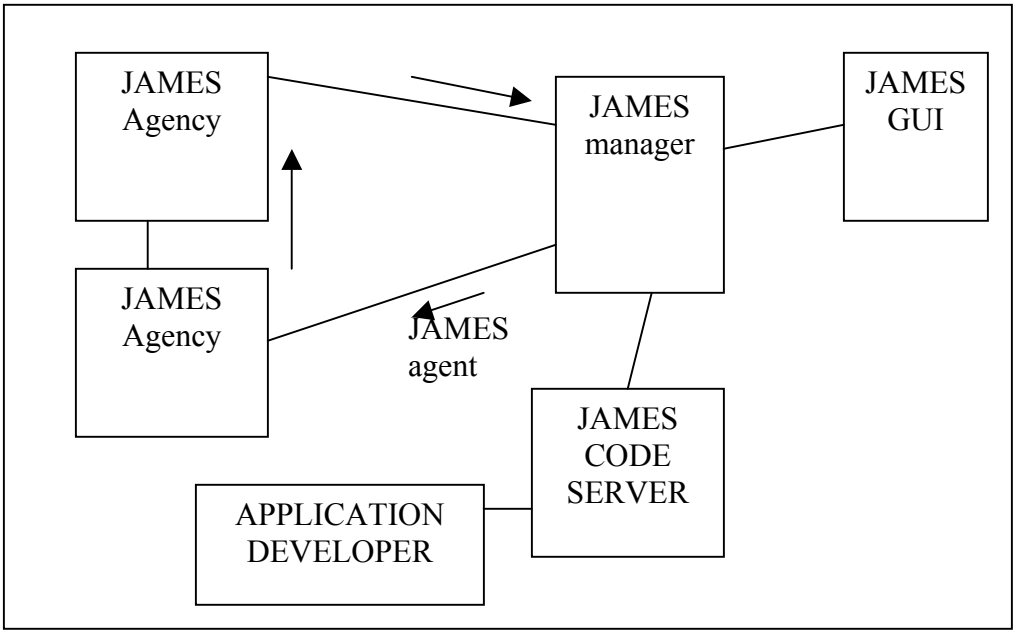

Figure 2-9 Overview of the JAMES platform

The JAMES SNMP integration module acts as both SNMP manager and SNMP Agent. This is one of the good solutions proposed and it is highly portable. It uses non-standard components, which might make its acceptance difficult. It also introduces redundancy.

2.4.2 Perpetuum Mobile Procura (PMP)

Perpetuum Mobile Procura [6,7] is a project at Carleton University, Canada, which is involved with developing network management solutions using mobile agents.

General architecture. PMP is a project at the Carleton University that aims at using mobile agents for network management. The solution integrates SNMP with mobile agent infrastructure. To achieve this, a new SNMP agent (XMS-SNMP Agent) that is extensible and compliant with the existing mobile agent framework is built from scratch. The mobile agents, which act as SNMP extension agents, register and un-register with the

main XMS-SNMP Agent using Distributed Program Interface (DPI) protocol. The mobile agents are transported across the network using a proprietary protocol. Figure 2-13 depicts the overall architecture of PMP.

2.5 Summary

This chapter gave a prelude to the concepts and technologies used to dynamically extend the network management functionality using mobile agents and HTTP. Network management is necessary to ensure the availability of the network and the applications that are dependent on it. Network management systems depend on defined standards to interface with network devices for monitoring and controlling their configuration, performance and functionality. SNMP is a standard for TCP/IP networks and LDAP (in MDS of Globus) is nowadays used for specialized networks like the Grid, which is a directory access protocol optimized for reading data. SNMP provides a set of parameters to monitor and configure the network. If extra parameters are to be configured, then protocols like AgentX exist to register the extra parameters and methods to retrieve them, without disrupting the SNMP service.

Finally the chapter discussed two approaches that use mobile agents to extend the network management service. The JAMES and PMP solutions provide this functionality with disadvantages of being very specific to SNMP and using proprietary protocols for agent transfer.

This chapter also discussed Apache web server, which will be used to receive the mobile agents transported through HTTP. It has a modular architecture and provides APIs to develop new modules. This chapter also discussed Secure Socket Layer (SSL), which provides methods for authentication and encryption using public key

cryptography. The Apache web server provides SSL support through *mod_ssl* module. *Mod_repository* is a freely available Apache module that facilitates transferring of binary

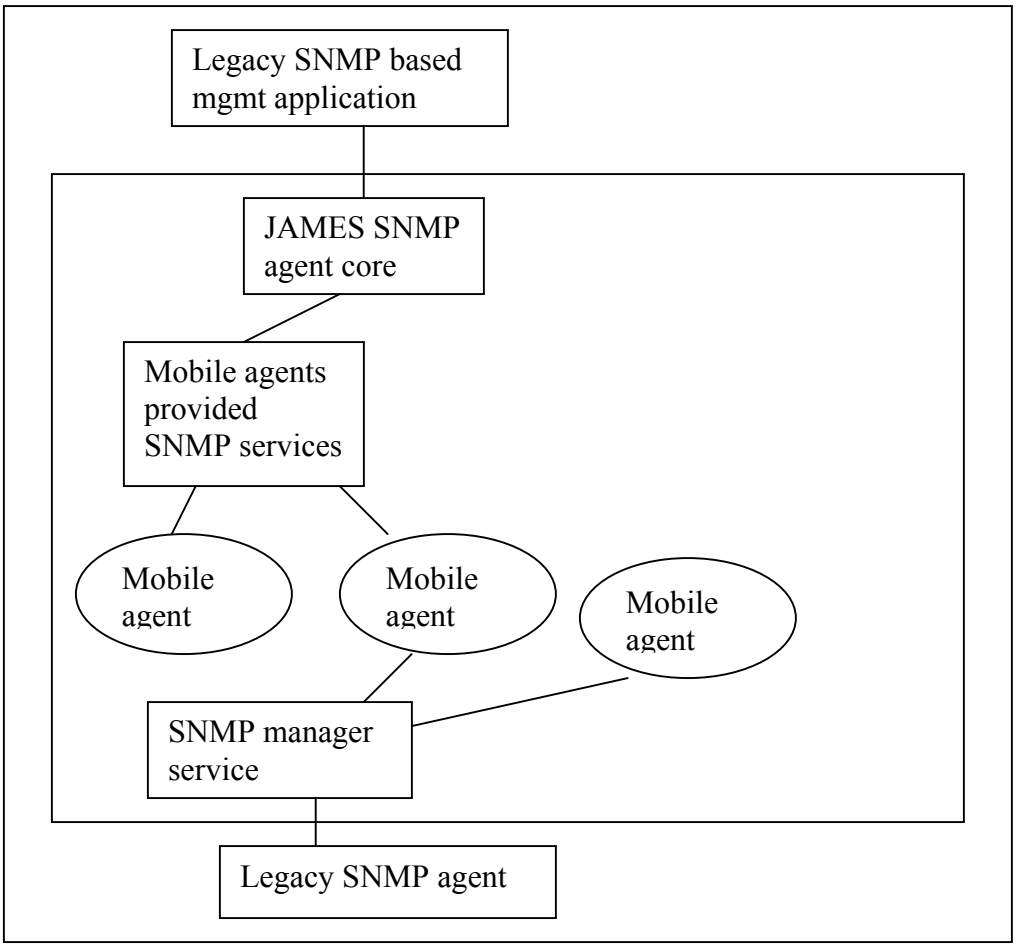

Figure 2-10 Overview of the integration of JAMES platform with SNMP

data over HTTP. The chapter also covers HTTP protocol to familiarize the user with the methods used and the structure of the HTTP transactions.The subsequent chapters discuss the requirements of the system at hand and the design that fits in the requirements and the implementation.

There are various network management services based on the type of network or type of parameters managed. They provide different means for extension. Extension mechanisms can be divided into two categories, ones that are file-based and the others

that are memory-based. In the first case, the extension mechanism uses files, which require the daemons to be stopped and started to reflect the changes made to the file. In

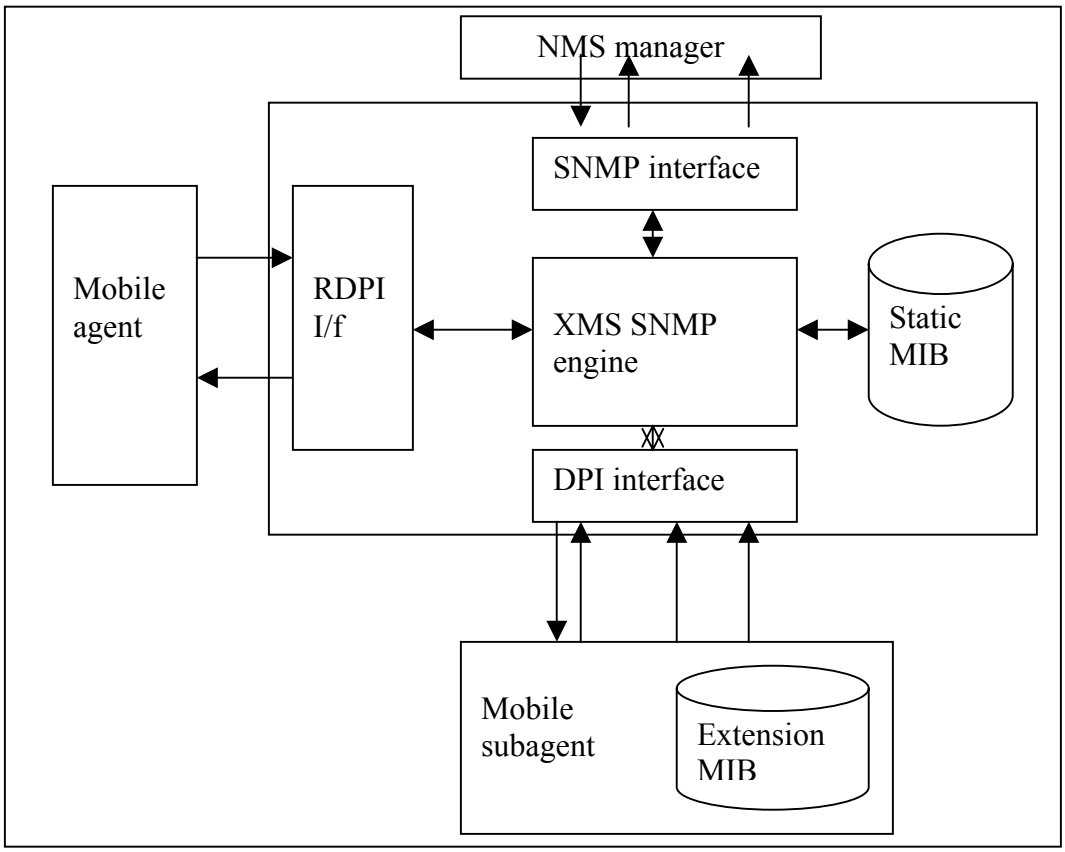

Figure 2-11 Overview of the PMP architecture

the second case, the extension data is augmented in the memory, with no need to read the configuration file and restart the daemon.

The extension mechanism proposed by the thesis depends on the second type of extension mechanism to integrate seamlessly new functionality into the system. The file-based extensions however require the network management daemons to reread the configuration files and restart. This can be achieved in Unix systems by sending the signal SIGHUP.

CHAPTER 3
REQUIREMENT SPECIFICATION

As noted in the introduction, this thesis proposes to make the extension process of network management services very flexible through mobile agents. This design could be extended to perform other network management functions in the future. However, transporting mobile code on heterogeneous systems (managed by disparate organizations) does have many issues that have to be tackled before such a system becomes acceptable and ready to use. We try to answer this question by deliberating on the processes involved in implementing and deploying such a system.

3.1 Processes

The processes can be divided into those involved on the WebNMAgent manager and those involved on the WebNMAgent server.

3.1.1 Processes at the WebNMAgent manager

This is the site from where the users can submit their WebNMAgent. It has to contain the following processes to manage the code on the WebNMAgent manager.

WebNMAgent control. The system has to have primitives or well-defined interfaces to create, delete and list WebNMAgents.

- **Create WebNMAgents:** This involves a method to build and transport the agent to the remote site.
- **Destroy WebNMAgents:** Since the agents are dispatched only when needed, there should be a mechanism to terminate them when no longer required.

- **List WebNMAgents:** Since there may be a possibility of failures on either the remote agent or the manager site, there should be a mechanism to get the agent (WebNMAgent) list from the remote site. This takes care of two things. Firstly, it helps the users to keep track of their agents, their location and their state. Secondly, it helps to guard against failures and reboots.

3.1.2 Protocol for Transporting the WebNMAgents

A protocol would be required to transport the WebNMAgents and should have the following properties.

- It should have methods to transfer data.
- A standard protocol is better as the acceptance would be higher.
- It should have security mechanisms to enforce confidentiality, integrity and authentication.

3.1.3 Processes at theWebNMAgent Server

This is the site on the remote network, where the extension agents are deployed. It should consist of the following processes.

WebNMAgent reception. Since the extension mechanism would be asynchronous, there should be a daemon listening to the WebNMAgents that will be dispatched from the manager. Since the whole process assumes a disconnected model (the functionality of the module would still be present even if the link to the manager is severed), there should be a scheme to store the dispatched WebNMAgent on the remote site.

Convention for naming and storing WebNMAgents. Since multiple users will be creating the WebNMAgents, there may be a possibility that two WebNMAgents exist in the same name space and cause ambiguity or overwriting of one by the other. Therefore it

is absolutely necessary for the WebNMAgents to exist in different name spaces. Also WebNMAgents belonging to one user should not be visible to the other user.

Resource allocation. The dispatched WebNMAgent should be executed and this consumes resources. This necessitates the inclusion of a resource manager. The resources are mainly the process space, CPU load, memory, number of open files, number of open sockets, etc.

WebNMAgent loader. There should be a module to load the WebNMAgent and spawn a thread / process running the WebNMAgent.

Network management module. This can be any module like UCD SNMP Agent, CMU SNMP Agent, LDAP Agent like the MDS by Globus or Grid monitoring tool like the MonaLisa. The Network Management Agent should be decoupled from the proposed system to allow any network management module to be configured.

As mentioned before, the mechanism for the extension agent is embedded within the extension agent. In this way, the intelligence is within the extension agent and hence makes the dynamic addition of functionality very flexible.

Error recovery and exception handling. Errors and exceptions can occur in two situations.

- Errors can occur while loading the WebNMAgent. This can happen if the path to the executable is not set appropriately or the configuration is not appropriate or the appropriate libraries are not loaded.
- Exceptions can occur while the code (WebNMAgent) is in execution. e.g., A file access is tried on a file, which is not authorized to be accessed.

There should be proper channels to conduct these errors and exceptions.

3.1.4 Security

When such a system is involved, which utilizes code transportation, there are bound to be issues with the system security. The system should take care of the possible threats that affect the confidentiality, integrity, authentication and availability of the system. Some of them are listed below.

- Someone unauthorized could use the system (affects confidentiality).
- The message (code) could be altered while transmission (affects integrity).
- A user gets access to the information for which he is not authorized to access (affects confidentiality).
- A resource may be over utilized (CPU, memory, bandwidth)(affects availability).
- A service could be disrupted (processes could be killed, data could be erased) or even crash (affects availability).
- There is room for accidents (commands like 'rm -rf *')(affects availability).

Appropriate security measures should be provided to safeguard against the above-mentioned threats. The security requirements can be broadly classified to preserve authentication, confidentiality, integrity and availability.

Requirements to preserve authenticity. Authenticity means ensuring that the origin of the message is correctly identified with an assurance that the identity is not false. There should be mechanisms to authenticate users and not allow any users not belonging to the system. Many systems do exist which provide authentication like using asymmetric cryptography or knowledge based authentication. The solution adopted should be designed for multiple users.

Requirements to preserve confidentiality. Confidentiality means ensuring that the information in a computer system and transmitted information are accessible for reading

by only the authorized entities. There should be mechanisms to enforce authorization (map right user to right privileges and right information) and encryption (to prevent unauthorized users to tap the data while transmission).

Requirements to preserve Integrity. Integrity means ensuring that only authorized entities are able to modify the data. There should be a mechanism to detect any unauthorized changes (tampering), e.g., have a digest of the information.

Requirements to preserve Availability. Availability is to ensure that the resource or service is available to be used by the authorized entities. This necessitates for a mechanism for resource control.

3.2 Summary

This chapter discussed the main requirements and the security requirements of the system. The goal is to have a secure mechanism to transfer the mobile code (extension agent) and execute it without affecting the availability of the system. It built the requirements of the system by analyzing the processes involved. Processes at the manager side consist of user-login and WebNMAgent control. Processes at the remote server consist of receiving the WebNMAgents, storing them and executing them. There also have to be modules to handle errors and exceptions. Finally, the system has to be secure to prevent any masquerading and prevent anyone unauthorized from accessing the system.

CHAPTER 4
DESIGN

This chapter discusses the high level as well as the detailed design of the system. It also explains the different modules required and their functionalities. Much care has been taken to keep the modules adhere to a single functionality. Hence they are decoupled as far as possible to interoperate with more types of network management services and in future, to interact with other systems other than dynamic extension of services.

The initial part of the chapter discusses about the design issues and the processes involved. The latter part will discuss about the modules that implement different parts of the processes required.

4.1 Design Issues and Processes

The following subsections discuss the design issues and the processes involved.

4.1.1 Choice of Protocol

The basic requirements of the protocol are

- It should have methods to transport the WebNMAgent
- It should have methods to enforce security (Authentication and Encryption)

The protocol can be proprietary or standard (non-proprietary). Considering these requirements, a proprietary protocol can be designed. Although it would serve the purpose, our supposition is that it would be difficult to be accepted, as the new protocol would again require new infrastructure and new configurations. A standard protocol that is already accepted and widely used and which satisfies the above requirements would be

ideal. There are two choices that provide the features viz. HTTP and SSH with FTP or SCP.

HTTP. HTTP is a widely used protocol to transfer data and control. It has the following merits.

- HTTP is a ubiquitous and standard protocol with built in support for security (HTTPS).
- It is well tested and much support is available.
- HTTP has built in mechanism for mobility, get and post methods. There is no need to reinvent other methods of transferring mobile code (WebNMAgents).
- There are many implementations of HTTP servers and one can be replaced by the other.

It also has the disadvantage that, it is a generic protocol. So it may contain extra features that are not required.

SSH and FTP/SCP. SSH is a protocol to transfer control commands across the network. When invoked, it transfers the standard input to the remote machine, executes it and then copies the standard output of the remote machine to the standard output of the local machine. This effectively implies that data cannot be transferred using just the SSH. However it can be combined with FTP or SCP to achieve both control and data transfer. It also offers security features like Authentication and Encryption. It has some demerits that were explained in Chapter 1. This mechanism requires another protocol to transfer data and has more security hazards. There is also no great advantage in using this mechanism. Since HTTP provides support to transfer both control and data, HTTP is chosen.

4.1.2 User Login Process

The user gets access to the system only through logging into the system at the manager side. The user has to have an account in the system to log into the system. This provides a trivial knowledge based security that may not be sufficient. The password files can be maintained at a central location or may be distributed. However greater level of security can be enforced through public key cryptography.

4.1.3 WebNMAgent Receipt and Execution

The dispatched WebNMAgent is received by the remote WebNMAgent server and stored using the scheme explained below. The scheme disallows any ambiguity that might occur due to the duplication of WebNMAgents by separating the name spaces. It also allows concurrency by writing them to different directories instead of files.
If the process reaches this stage, then it implies that the WebNMAgent was dispatched successfully and stored and ready to be executed. Before executing the WebNMAgent, it has to be checked whether the WebNMAgent sent is authorized to access the information he intends to use. If allowed, a check has to be made to decide whether the resource requirements of the WebNMAgent is within the limits allowed according to the policies of the machine being used. If all these criteria are satisfied, then the WebNMAgent is executed.

Errors and exceptions may occur while installing the WebNMAgent. Errors can be due to missing libraries or inappropriate executables. Notifications are sent to the manager, the WebNMAgent destroyed and the data structures updated reflecting the changes.

4.1.4 WebNMAgent Storage Scheme

It has to take care of disambiguating the WebNMAgents that may occur due to duplication and to allow concurrency among the WebNMAgents. A separate directory is created for each WebNMAgent and the name of the directory pre-pended by the user name to keep the WebNMAgent name spaces separate. There can be a clash in the WebNMAgents only when there is duplication that should be taken care by the WebNMAgent controller.

4.1.5 Error and Exception Handling

As discussed before, errors and exceptions might occur while installing the WebNMAgent or while executing the WebNMAgent. Notifications for errors and exceptions can be conducted to the manager either synchronously or asynchronously.

If synchronous methods are used, sessions have to be maintained for the lifetime of the WebNMAgent dispatched. This is contradictory to the model proposed, which basically assumes a disconnected model. Moreover, it would prove very costly in terms of resources used.

The best way of conducting the information is through the asynchronous channel. It can either be sent through a mail or through notifications to a daemon listening on a standard port. Since this is intended to leverage network administrator's work, a notification by mail would mean human intervention. So the best method to convey the messages is to have a daemon listen to a port, where notifications will be sent and update the relevant data structures to reflect the changes.

4.1.6 Failure and Recovery Handling

Information about the WebNMAgents is maintained on only the remote WebNMAgent server. When the WebNMAgents are successfully deployed, information about the WebNMAgent is updated on the remote WebNMAgent.

4.1.6.1 Protocol to update the state information of the WebNMAgent.

When the manager dispatches the WebNMAgent, the remote WebNMAgent server updates its table in the stable storage and sends a response message indicating the status of the dispatched WebNMAgent. Failures can occur either on the manager or the remote WebNMAgent site, which can render the system in an inconsistent state and make the information maintained on the manager and on the remote WebNMAgent site differ. We construct a state machine to illustrate all the scenarios and the actions to be taken in each case.

4.1.6.2 Fault tolerancy when manager fails

The information about the WebNMAgents is cached at the manager. When the manager comes up, it flushes the cache and builds a new image of the system using listAgents interface. Since the information at the remote WebNMAgent site is stored in stable storage before sending the status message, it will always be consistent with the image on the manager site.

4.1.6.3 Fault tolerancy when remote site fails

Whenever any remote WebNMAgent site comes up after a crash, it sends an *UPDATEALL* notification to the manager. This rebuilds the cache information of that site at the manager.

Before the message is sent and before the stable storage is updated. There can be three scenarios.

- **manager sent nothing before the crash**

  The manager and the remote WebNMAgent will be consistent. When the site comes up, it reads from the stable storage and nothing has to be done. The remote site sends an *UPDATEALL* notification that rebuilds the cache information of that site at the manager. So the manager and the remote site will be consistent.

- **manager sent createWebNMAgent**

  The WebNMAgent is spawned at the remote site, but the site fails to update the tables in stable storage and send a message. This does not update the cache at the manager. So when the remote site comes up, it reads from stable storage and the list shows only the WebNMAgents that existed before the crash. So both the manager and the remote WebNMAgent have a consistent image. The new WebNMAgent is not reflected.

- **manager sent deleteAgent**

  It is similar to the above scenario, but the deleted WebNMAgent will show up again.

Before the message is sent and after the stable storage is updated. There can be three scenarios again.

- **manager sent nothing before the crash**

  Nothing will be done. The remote site clears the cache information and updates the cache information with the latest information on the site. The manager as well as the site will have consistent image.

- **manager sent createWebNMAgent**

  The information is updated on the remote site, but is not propagated to the manager. When the site comes up it updates the its information on the manager's cache. So the

information on the manager will become consistent with the information on the remote site. When the site comes up, the WebNMAgent can be seen on the manager.

- **manager sent deleteAgent**

  This is similar to the above scenario, but the deleted WebNMAgent will not be visible at the manager after the site comes up. During the time, the site is down and the manager is running, the manager will have cache information about the remote site that is invalid, but this does not cause any serious problems.

4.1.7 User Authentication

This can be done in several ways.

User certificates based authentication. Public key cryptography is applied here to implement authentication. As discussed in Chapter 2, there can be a user-based authentication with user certificates. This lends a lot of flexibility and allows the user to log on to the manager from any network. This eliminates the malevolent user masquerading the legitimate user.

Password based authentication. There is also password-based entry into the system that still prevents illegal intruders even if they have managed to steal the public keys.

Host based authentication. If more rigorous security is desired, host-based authentication could be used with the manager also having the certificates. This requires that the user log on to the system only through the allowed machine.

Here client authentication is the only desired feature, because it is the one that is pushing data.

4.1.8 Message Encryption

This is to guard against the tapping of the data during transmission. Asymmetric cryptography is being employed to encrypt the session keys that produce the session key. The session key is later used to encrypt and decrypt data.

4.1.9 Authorization

The trickiest part is encountered in the granting of authority to the WebNMAgent. Since the application domain is a collaborative network, there are different domains administered by different entities. So the policies set by each domain might not be uniform. The trickiest part comes in specifying the policy parameters. To ease the definitions of the policies, the parameters on which they are based are discussed in this section.

The parameters can be derived from the activity that is anticipated in such a system. The anticipated activities can be broadly classified as accessing file resources, network resources, I/O resources memory resources, process resources and executing files.

- **Accessing file resources:** The file accesses can further be classified as read, write, delete, execute accesses.
- **Accessing network resources:** The network accesses can be further classified as opening sockets from <host:port> combination.
- **Accessing I/O resources:** These can be similar to file accesses with I/O devices being treated as files.
- **Accessing memory resources:** This means creation of objects in the main memory. There should be limit on this, but nevertheless cannot be enforced through policies as the memory is not referenced by names.

- **Accessing process resources:** Process resources include creation of processes or threads. This also does not have any named references and hence it is difficult to enforce through policies. However the resource controller can enforce resource limitations.
- **Executing files:** This is running executables and shell commands. These have execution rights that should be revoked to stop a user to execute it.

The policies can be expressed as the combination of one or more of these privileges. Assigning these privileges to users is an issue.

The activities involved in using the WebNMAgent system are all the operations done by a normal local user of the machine. But it becomes difficult to insure one user from the other, if all the users share the creation and deletion rights. One user can delete the WebNMAgent created by another. To avoid this, the users can be split into two types of users as in Unix based systems.

- Network normal user.
- Network super user.

The network normal user can create and destroy WebNMAgents only created by him, but the super user can destroy WebNMAgents created by other network normal users. This helps to manage the system, just as super user is required to manage things on a normal Unix platform. The privileges given for each type of user is described in the table below.

4.2 System Design

The system can be broadly split into the manager module and the Remote WebNMAgent module.

Manager module. This module is responsible for creation and dispatch of WebNMAgents. The user has access to the system through this module. It contains many sub-modules to handle different functionalities.

Agent controller module. This module provides interfaces for the creation, deletion and listing of WebNMAgents. Each of the operations is discussed below.

Agent creation provides mechanisms to develop and transport WebNMAgents across the network. It updates the data structures whenever a WebNMAgent is dispatched.

Agent deletion provides mechanisms to destroy the WebNMAgent on the remote site and update the relevant data structures.

List agents provides a mechanism to list the WebNMAgents from a remote site.

Error and exception handler module. Errors and exceptions may occur while installing WebNMAgents or executing WebNMAgents. These are relayed by the remote WebNMAgent to the manager that are handled by this module.

Remote WebNMAgent module. This module is responsible for receiving the WebNMAgent and executing it.

WebNMAgent receiver module. This module receives the dispatched WebNMAgent and checks the security constraints like authentication and authorization, resource constraints and finally launches the WebNMAgent. This module interfaces with the authentication, authorization and resource control modules.

Resource Controller module. This module is responsible for checking the resource requirements of the WebNMAgents loaded and returns a status message if the resource constraints are not met.

Table 4-1. Privileges of the system users

	Network super user	Network normal user
File resources	Read, write, delete accesses This user can delete files created by other network normal users. However this user does not have rights to access system resources and files not belonging to the WebNMAgent system.	Read, write, delete accesses This user can delete the files created only by him, but cannot delete files of other, nor read or write to files from others.
Network resources	Create client/server sockets to communicate back with the manager.	Create client/server sockets to communicate back with the manager.
Process management	Can kill processes created by any WebNMAgent in the system. The rest of the capabilities are as much as that of the local user this user is mapped to.	Can kill processes created only by that user.

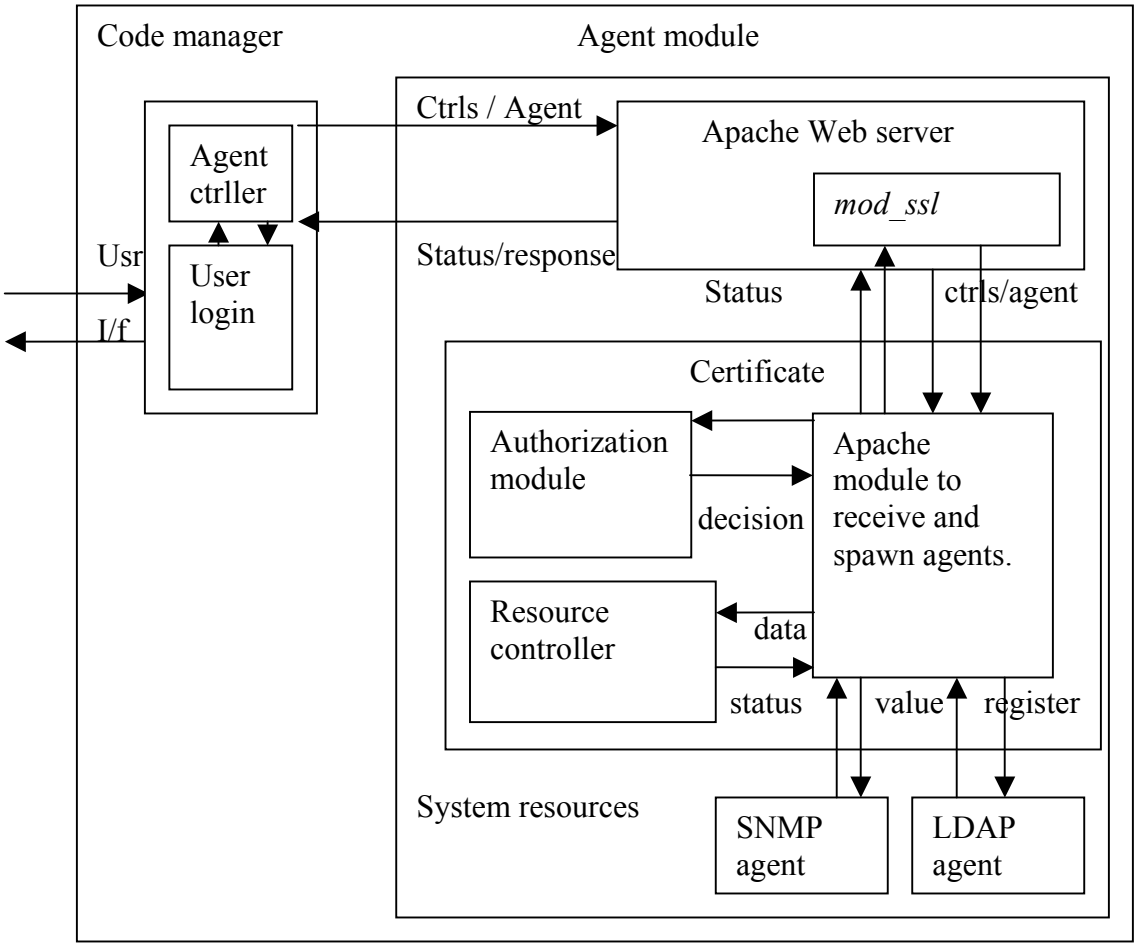

Figure 4-1. System design proposed in the thesis

Network management module. It can be any network management module like SNMP or LDAP that is extensible. It is de-coupled from the system, to make the system flexible and free to interoperate with different types of network management modules.

4.3 Security Model

Security is involved both at the manager as well as the remote WebNMAgent server.

User login module. This module implements the user login process. This module ensures that only the users belonging to the system get access to the system.

Authentication module. This module enforces the authentication process. It uses both password-based and user certificate-based authentication.

Authorization module. This module is also part of the security framework. The system assumes multiple users to be accessing the system. This module interacts with the WebNMAgent receiver module to get the user credentials like the certificates and grant permissions to authorized users based upon the policies specified at the node.

4.4 Summary

This chapter discussed the system design and the various modules involved. The major design issue is the choice of a protocol. Standard protocols lend flexibility to use any implementation and allow the developer to customize the implementation to suit the application.

The user is allowed to enter the system by providing a username/password. Greater security can be implemented using user certificates. The remote WebNMAgent server receives the WebNMAgent dispatched by the manager, stores it and executes it. To keep the WebNMAgents unique in a multi-user environment, the names are pre-pended by the username. Errors and exceptions are conducted through asynchronous means. Information about the WebNMAgents is maintained only at the remote WebNMAgent site. This guards against failures. All the communication is encrypted to preserve integrity.

The design introduced two types of users in the system to grant different authorization rights to different users, based upon the local network policies. The network normal user is granted rights to create and delete WebNMAgents. The network superuser can control WebNMAgents of other users as well. Thus an attempt is made to address the security concerns. The next chapter discusses the implementation of the design.

CHAPTER 5
IMPLEMENTATION AND TESTING

This chapter deals with the important details of implementation and the testing done.

5.1 Choice of Language

The system is intended for heterogeneous platforms in a network. This makes the language be independent of the underlying platform. According to this criterion, the languages selected would include any kind of shell scripts like Perl, or Java. Since Java provides platform independency and is very powerful and supports security, the obvious choice was Java to develop the WebNMAgents.

5.2 Choice of Web Server

Apache 1.3.27 was chosen as the web server because of the following reasons

- It is the most widely used server,
- it is open source,
- it is modular,
- it is well documented and easy to extend,
- it is ported to many platforms and
- its implementation is in C, which is very powerful and familiar.

5.3 Choice of Network Management Agent

SNMP agent from the project JAMES was used to test the system. Many other SNMP systems could be used which support AgentX protocol to extend the SNMP agent. Other network management toolkits like the MDS from Globus could also be used.

5.4 Implementation Details

The system consists of two main modules: the WebNMAgent manager and the WebNMAgent server. The functionalities of the WebNMAgent manager include:

- provide a user interface for the users to interact with the system,
- provide an environment for WebNMAgent development,
- provide mechanism to package the WebNMAgent that is consistent with the implementation on the Remote WebNMAgent server,
- provide mechanisms to receive notifications and conduct them to the user,
- provide mechanism to transport the WebNMAgent,
- provide mechanisms for authentication,
- provide mechanisms for encryption.

The functionalities of the WebNMAgent server include:

- provide mechanism to receive WebNMAgent,
- provide mechanism to store the WebNMAgent,
- provide mechanism to perform authentication checks,
- provide mechanism to perform authorization checks,
- provide mechanism to enforce resource control,
- provide mechanism to execute the WebNMAgent (rest of the activity is taken care by the WebNMAgent itself),
- do housekeeping activity like storing the WebNMAgent information in stable storage and updating it whenever a WebNMAgent gets created or destroyed,
- provide mechanism to conduct errors.

The implementation is listed module-wise and the role of each module is discussed.

5.4.1 WebNMAgent Server

This is the central module of the implementation and developed as an Apache module. It consists of an extended version of *mod_repository* [15]. This implementation incorporates all of the functionality of the WebNMAgent server in this Apache module. The idea is to make the support for dynamic extension of network management service transparent to the user and optional. The module can be included with the main module by a command like the one illustrated in Figure 5-1. The next sections describe the functionality of the module.

```
• ./configure --add-module=/home/suchin/httpd/Apache_1.3.27/src/modules/<module.c>
• make
• make install
```

Figure 5-1. Configuring and installing the Apache module

The functionality is divided into two parts. The first part deals with configuring and modifying the module to work with Apache. This includes setting up the dispatch table for handlers, setting the fields of the dispatch table, which registers functions with the Apache hooks. The second part deals with describing the implementation details of receiving the WebNMAgents, making various checks and executing the WebNMAgents.

5.4.1.1 Apache specific implementation

This part discusses the implementation that uses Apache and the supporting Apache modules.

Configuring *mod_repository*. As explained in Chapter 2, *mod_repository* provides some configuration directives to tune its functionality. Figure 5.2 lists the directives to be modified to modify its behavior to suit the requirements. This directs the Apache to store the files received in the directory "/home/suchin/Agents". The data is stored in files directly in the directory specified by the *RepositoryRoot*. This module provides methods

```
RepositoryRoot /home/suchin/Agents
RepositoryFileLevels 1
RepositoryType agent/form-data
```

Figure 5-2. Directives of *mod_repository* module

to receive the WebNMAgent, within which the request is processed. These functions are explained later in this section.

Declaration of handlers. The Apache module has a dispatch table associated with it that associates a content handler with a function name in the module. Figure 5-3.depicts the data structure from the module. The first argument "repository" is the name of the handler that is called. The second argument "repository_handler" is the name of the function.

```
static const handler_rec repository_handlers[] = {
    { "repository", repository_handler },
    { NULL, NULL }
};
```

Figure 5-3. Content handler table in *mod_repository* module

A directive is inserted into the httpd.conf to associate the handler name with the file extension. When a file object with the extension specified is requested, the corresponding module gets invoked and the request object is passed to that module. Figure 5.4 depicts the configuration snippet required.

Hooking up the module. The modules hook up their content handlers through the structure module *MODULE_VAR_EXPORT* as a call back function. So when a request comes from a client, based on the file extension, the relevant content handler is called which calls the relevant function. The structure "module *MODULE_VAR_EXPORT"* is explained below.

httpd.conf configuration
AddHandler repository agentx

Figure 5-4. Code snippet configuring the content handler.

```
/* Dispatch list for API hooks */
module MODULE_VAR_EXPORT repository_module = {
    STANDARD_MODULE_STUFF,
    NULL,               /* module initializer                  */
    create_mconfig,       /* create per-dir    config structures */
    NULL,               /* merge  per-dir    config structures */
    NULL,               /* create per-server config structures */
    NULL,               /* merge  per-server config structures */
    repository_cmds,       /* table of config file commands       */
    repository_handlers,   /* [#8] MIME-typed-dispatched handlers */
    NULL,               /* [#1] URI to filename translation    */
    NULL,               /* [#4] validate user id from request  */
    NULL,               /* [#5] check if the user is ok _here_ */
    NULL,               /* [#3] check access by host address   */
    NULL,               /* [#6] determine MIME type            */
    NULL,               /* [#7] pre-run fixups                 */
    NULL,               /* [#9] log a transaction              */
    NULL,               /* [#2] header parser                  */
    NULL,               /* child_init                          */
    NULL,               /* child_exit                          */
    NULL                /* [#0] post read-request              */
#ifdef EAPI
   ,NULL,               /* EAPI: add_module                    */
    NULL,               /* EAPI: remove_module                 */
    NULL,               /* EAPI: rewrite_command               */
    NULL                /* EAPI: new_connection                */
#endif
};
```

Figure 5-5. Content handler table in *mod_repository* module

5.4.1.2 WebNMAgent server specific implementation

This part includes implementation details of how the WebNMAgent is received, the various security checks made and the WebNMAgents executed.

Entry point. The WebNMAgent is transferred using HTTP POST request. So the entry point to the WebNMAgent server is the function that handles the POST request. This function is responsible for receiving the WebNMAgents and storing them. It creates a file in which it stores the WebNMAgent received. A special name is generated that makes sure that there is no name clash while storing. The same module checks a name clash later.

The WebNMAgents are created by including the key word “create” in the URL requested. This is parsed in this function and the method createWebNMAgent(key, r) is called. Similarly, if the keyword “destroy” is found, then destroyWebNMAgent(key,r) is called and if “list” is the keyword then the WebNMAgent list is sent to the manager. The WebNMAgent is packaged as a jar file that is stored in the file by a special name. Figure 5-6 illustrates the flowchart of the processes going on at the remote WebNMAgent site.

Error / Exception conveying mechanism. As discussed in the design, notifications are sent asynchronously. The manager runs a JSP engine. The remote WebNMAgents send a message to the manager by invoking the jsp “resolve.jsp” and passing the message as arguments. The message will be processed and parsed by “resolve.jsp” at the manager and updates the information about the remote WebNMAgent that sent the message in the cache and will be immediately reflected in the user view.

5.4.2 WebNMAgent Manager

The WebNMAgent manager is implemented on any node in the network that has the following capabilities.

- It is visible on the Internet. Since the WebNMAgent servers will be sending their traps messages and status messages to the manager, the manager should be visible to all the WebNMAgents. It implies that it has a valid IP address,
- It runs a JSP engine,
- It has a web browser that supports HTTP 1.1.

The user interface to control the WebNMAgents is implemented using HTML and displayed on a web browser, since web browsers are HTTP clients. The various processes that are involved at the manager site, are elaborated along with their implementation.

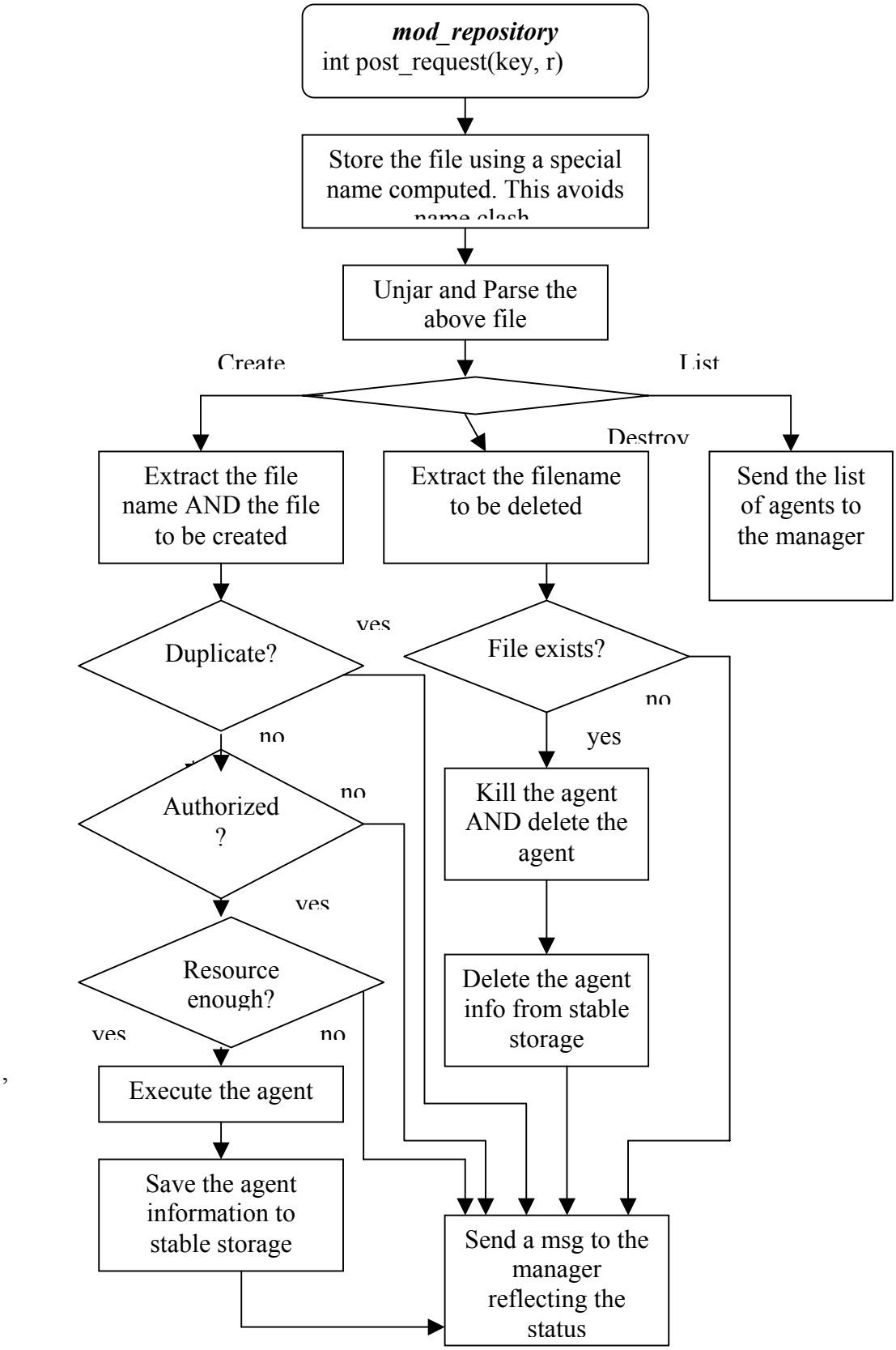

Figure 5-6. Flow chart explaining the WebNMAgent server logic

5.4.2.1 User login

The user can enter the system only through logging into the system, for which he is required to have a valid account. This serves two purposes.

- It provides security
- It provides customization for the user. So that the user is taken to the web page with only his information.

The information about each user is maintained in a separate directory so that the access rights can be enforced at the directory level. The login can be controlled by ".htaccess" mechanism provided by the Apache web server. The Apache web server maintains a file called "*.htaccess*" to enforce secure access to the directory it is in. The contents of the ".htaccess" file are shown in Figure 5-7. The ".htpasswd" file contains the username and password to enter the directory.

```
AuthUserFile ./.htpasswd
AuthGroupFile /dev/null
AuthName User login
AuthType Basic
require valid-user
```

Figure 5-7. Sample .htaccess file

5.4.2.2 User interface

The user interface has mechanisms to login, create WebNMAgent, destroy WebNMAgent and list WebNMAgents. All these operations are embedded in an HTML page. Since HTTP is the mode of communication, it is very natural to use web browsers to interact with the remote WebNMAgent system.

The user login leads the user into this page. The HTML page contains options for the user to select the operation. Each of the operation is embedded as a form action that sends or pushes the binary file selected to the remote WebNMAgent.

Interface to create the WebNMAgent. The WebNMAgent is dispatched as a result of the form action, which is of the form illustrated in Figure 5-8.

```
<form action="http://grinpc03.phys.ufl.edu/home/createagentx" ENCTYPE="multipart/form-data"
METHOD=POST>

JAR FILE CONTAINING THE AGENT <INPUT TYPE=FILE NAME=file><BR>
<INPUT TYPE=SUBMIT>
```

Figure 5-8. Interface to create WebNMAgents.

The important point to notice is the file being requested. The extension “agentx” invokes the “repository” handler and the keyword “create” invokes the “createWebNMAgent” function on the remote WebNMAgent server. The HTML input type “FILE” is used to select the jar file to be sent. This allows the user to transport files across the network.

Interface to destroy the WebNMAgent. Similarly for destroying the WebNMAgent the code snippet is given as Figure 5-9.

```
<form action="http://grinpc03.phys.ufl.edu/home/destroyagentx" ENCTYPE="multipart/form-data"
METHOD=POST>
```

Figure 5-9. Interface to destroy the WebNMAgent

Interface to list the WebNMAgents. Similarly for list WebNMAgents, the code snippet is illustrated in Figure 5-10.

```
<form action="http://grinpc03.phys.ufl.edu/home/listagentx" ENCTYPE="multipart/form-data"
METHOD=POST>
```

Figure 5-10. Interface to list the agents

5.4.2.3 WebNMAgent development environment

The WebNMAgents are all implemented in Java. The Java classes can be developed and compiled using any tool. If any packaging information is inserted in the Java code, then the *CLASSPATH* can be set in the manifest file. Even otherwise the *CLASSPATH*

can be set to include any paths. However, the *CLASSPATH* for this environment is set to the current working directory so that a class with or without any package information could be executed. To include any other path, the manifest file could be manipulated.

5.4.2.4 Mechanism to package the WebNMAgent

The WebNMAgent is packaged as a jar file. This has three advantages. It

- compresses the files,
- provides the user with a choice to include any libraries required,
- is uniform across the platforms, unlike "tar" which can be used only on Unix platforms.

5.4.2.5 Mechanism to receive notifications

The remote server WebNMAgents can send notifications to manager by invoking the *resolve.jsp* Java server page. This writes the information to the cache that is maintained as a file in the users directory guarded by ".htaccess." This has the following advantages.

- The capability is provided by the web server. This still falls within the limits of HTTP, which prevents the need to invent one more protocol to convey the messages.
- This shunts out user intervention. The lists are automatically updated.

5.4.2.6 WebNMAgent interaction with the network management agent

The implementation uses SNMP agent by the JAMES project. It is enabled with AgentX support, which enables it to extend the SNMP agent dynamically (Without stopping the service). The mechanism to interact with the network management agent can change. This information is hence always included in the WebNMAgent dispatched. The WebNMAgent server provides the mechanism to execute the code sent. The WebNMAgent handles the rest of the mechanism. The network management module is

decoupled from the WebNMAgent server. As a result, any network management module can be made to interact with the WebNMAgent server.

5.4.3 Security Model

Security includes three areas:

- authentication,
- encryption and
- authorization.

The security is implemented using public key cryptography, as well as password-based security. *mod_ssl* provides strong cryptography for the Apache 1.3 webserver via the Secure Sockets Layer (SSL v2/v3) and Transport Layer Security (TLS v1) protocols by the help of the Open Source SSL/TLS toolkit OpenSSL. This module is configured into the Apache installation to provide security.

5.4.3.1 Authentication

Authentication is based on certificates. In this case both the WebNMAgent server and the user on the manager has server and user certificates respectively.

User certificates. Certificates are encoded in different formats prescribed by the standard used. All browsers use pkcs#12 standard to store private and public keys securely. So user certificates are stored in pkcs#12 format as .p12 files at the manager. These can be imported into the browser.

Server certificates. The server certificates are installed on the server. Apache requires the server certificates to be pem encoded. Appropriate changes have to be made to the httpd.conf to reflect the changes. The paths to the server certificates and keys are specified in the httpd.conf as illustrated in Figure 5-12.

These certificates are currently produced by the server and dummy certificates from BelSign are used. For a real implementation valid certificates have to be used.

Authentication procedure. The user has to be authenticated based on the certificate he possesses. This requires that there be client authentication. Client authentication can be enforced by inserting the configuration information in httpd.conf as illustrated in figure 5-11.

How certificate validation works. Although this has been explained in Chapter 2, the required elements are reiterated according to the context. The client certificate is signed by a certification authority, which may or may not be on the list of trusted Certification authorities on the server. If it is present, then the certification authority is

```
#   Client Authentication (Type):
#   Client certificate verification type and depth.  Types are
#   none, optional, require and optional_no_ca.  Depth is a
#   number which specifies how deeply to verify the certificate
#   issuer chain before deciding the certificate is not valid.
SSLVerifyClient require
SSLVerifyDepth  10
```

Figure 5-11. Snippet of httpd.conf showing client authentication enforcement

verified, else the parent of the certification authority is contacted, which is level one up. This continues till the SSLVerifyClient limit is reached. The server and client exchange the certificates as per the SSL protocol and the client and server are hence verified and authentication method complete.

```
#   server Certificate:
#   Point SSLCertificateFile at a PEM encoded certificate.  If
#   the certificate is encrypted, then you will be prompted for a
#   pass phrase.  Note that a kill -HUP will prompt again. A test
#   certificate can be generated with `make certificate' under
#   built time. Keep in mind that if you've both a RSA and a DSA
#   certificate you can configure both in parallel (to also allow
#   the use of DSA ciphers, etc.)
SSLCertificateFile /usr/local/Apache/conf/CA/server.crt

#   server Private Key:
#   If the key is not combined with the certificate, use this
#   directive to point at the key file.  Keep in mind that if
#   you've both a RSA and a DSA private key you can configure
#   both in parallel (to also allow the use of DSA ciphers, etc.)
SSLCertificateKeyFile /usr/local/Apache/conf/CA/server.key

#   Certificate Authority (CA):
#   Set the CA certificate verification path where to find CA
#   certificates for client authentication or alternatively one
#   huge file containing all of them (file must be PEM encoded)
#   Note: Inside SSLCACertificatePath you need hash symlinks
#         to point to the certificate files. Use the provided
#         Makefile to update the hash symlinks after changes.
SSLCACertificateFile /usr/local/Apache/conf/ssl.crt/ca-bundle.crt
```

Figure 5-12. Snippet of httpd.conf where server certificates are defined

5.4.3.2 Encryption

Once the authentication is done, the rest of the communication is encrypted by the session key generated during the handshake phase of SSL. The module *mod_ssl* on the remote WebNMAgent server receives the WebNMAgent, decrypts it with the public key obtained during handshake and pass the decrypted data back to the Apache, which will be further passed on to the appropriate handler.

5.4.3.3 Authorization

This module is still not implemented and is left as future work. The idea is that, a file is maintained at the server, which contains a mapping of the certificate's DN to a local user on the node. The user at the manager is mapped on to a specific local user at the server through this file. The user will have the rights of the local user to which he is mapped to. The policies are embodied in a policy file and included at runtime while spawning the Java classloader.

5.5 Testing

A series of unit and system testing was done, the details of which are discussed below.

5.5.1 Testing User Login

The system should let only valid users into the system. Multiple user accounts were created with a ".htaccess" file for each of them. User login was successfully tested with multiple users. The system allows the users only after they present a valid username and password.

5.5.2 Testing Agent Controls

The system should display the WebNMAgent controls once the user logs in to the system.

5.5.2.1 Testing create WebNMAgent

The system should allow the user to select and transfer the WebNMAgent. Currently the system functions with only one machine. However it should be configured to send to any machine from a list of machines. As a result two things should happen. The browser gets a response reflecting the status of the WebNMAgent execution and the cache at the manager gets updated. Several WebNMAgents were dispatched to check whether the WebNMAgents get created. The status messages were successfully received.

Testing name spaces. Different users used same jar names. Jar names are effectively the WebNMAgent names. At the remote WebNMAgent site, they are stored with names pre-pended by their user names, thus separating the name spaces. This was tested successfully.

Testing valid class files. Valid class files are the ones that execute without exception. If the files are executed successfully, then the status message reflects this condition and the cache was updated accordingly.

Testing invalid class files. Invalid classes are the ones that execute with exception like "class not found," or "unable to bind." The status messages also reflected these conditions appropriately.

Testing registration with the SNMP agent. An AgentX-enabled SNMPagent was started at the test WebNMAgent server. The WebNMAgent currently does not respond to the command "snmpget grinpc03.phys.ufl.edu public .1.3.6.1.4.1.1331.1.0." This query returns with the result "object not found". The SNMP agent responds with the value configured when the WebNMAgent is successfully created. This means that the subagent dispatched by the manager successfully registered with the main agent. If there is a duplicate registration, then the agent returns with "duplicate registration" message, which is communicated to the manager through the status message channel.

5.5.2.2 Testing delete agent

This case can have the following tests.

Valid WebNMAgents deleted. Specified WebNMAgents of the user were destroyed by killing the processes related to the agent and removing all the entries from the remote WebNMAgent site. The query "SNMPget grinpc03.phys.ufl.edu public .1.3.6.1.4.1.1331.1.0" was issued. It responded with "object not found," which indicates that the WebNMAgent was indeed destroyed.

Non-existent WebNMAgents deleted. The status message reflected this condition.

5.5.2.3 Testing list agents

List-agents compels the remote WebNMAgent to clean the cache at the manager and update with its latest list of WebNMAgents. This was successfully done.

5.5.3 Testing Node Failures

This ensures the fault tolerancy of the system.

WebNMAgent manager was shut down. When the server came up, it successfully reconstructed the whole WebNMAgent list. The WebNMAgent list on the manager and those on the WebNMAgent servers were coherent. The cache is actually stored in stable storage, which was successfully retrieved.

WebNMAgent server was shut down. When the WebNMAgent server came up, it successfully started all the WebNMAgents that were stored to stable storage before the crash. It sends an update message to the manager as soon as it comes up.

5.5.4 Testing Security

Security includes authentication, encryption and authorization.

Testing authentication. The system should enforce user-name/password authentication, as well as client authentication in the form of user certificate.
The system presents username/password every time the users tries to enter the system. Users with wrong usernames or wrong passwords were disallowed from entering the system. This is enforced at the manager. The user is required to choose his user certificate from the list maintained by the browser at the manager. Wrong certificates did not allow the user to send the message. They were rejected at the server side (remote WebNMAgent site). So users with valid username/password and valid user certificates were allowed.

Testing encryption. The system was tested without encryption, i.e. http not https. The system should not allow the communication without the encrypted channel. This was tested successfully.

5.6 Summary

The whole functionality of the system can be divided into two parts, the manager and the remote server. The manager can be any node on the machine visible on the Internet or the intranet if the scope is an intranet. The manager implements the user interface using HTML as all web browsers are HTTP clients.

The server implements the functionality by extending the capability of *mod_repository* Apache module, which already has the capability to transfer binary files. The server implements important functions like receiving the WebNMAgent, making several checks and finally spawning a class loader to execute the WebNMAgent. The mechanism of a content handler is used to invoke the Apache module when a file object with the extension ".agentx" is requested.

The server also implements error and exception handling by sending notifications to the manager through invoking jsp and passing the status messages as parameters. The next chapter is the last chapter, which concludes the thesis and underlines the accomplishments made in thesis and the future work in the area.

CHAPTER 6
CONCLUSION AND FUTURE WORK

The objective of this thesis was to provide an elegant solution for dynamic addition/extension of network management services using mobile code. It successfully demonstrated an implementation of the concept of mobile code applied to the field of network management. The thesis emphasized on providing a solution that makes use of most of the existing infrastructure, thus reducing redundancy in the modules being used and making it easily deployable.

HTTP was used to transfer the extension agents and Apache web server was used to implement the module required to provide the basic functionality of receiving the mobile code containing extension agents and execute them. HTTP over SSL was used to enforce authentication and encryption. Functionality was imbedded into the Apache server by developing a module that had handlers to handle file types with a particular extension (.agentx). So the whole support is transparent. It is also optional and the functionality can be availed by just including the Apache module. The design is also modular where the network management module was completely decoupled from the agent system. This makes it interoperable with any type of network management service.

Future work. This thesis lacks some features that can be provided later. Future work can be listed as follows.

- The thesis proposes a scheme to enforce authorization but does not provide an implementation.

- The thesis emphasizes the importance of resource control, but does not provide .an implementation.
- The thesis is currently ported only to Unix platform. Work has to be done to port it to other platforms.
- The user interface provides limited barebones functionality that can be improved in the future.

Hence one of the main concerns of implementing a mobile code based implementation like resource control has to be addressed to make it robust. The current implementation uses Apache. Other web servers can be used if they provide a modular extension of their functionality.

APPENDIX
BUSINESS MODEL

With the growth of networks, the dependence on the network and the applications using them is also growing. In a system where mission critical problems are dependent on the availability of the network, it becomes necessary to have a network management system that ensures this and also centralizes the control of the network. The way this is ensured is by monitoring and often configuring certain parameters in the network.

There can be different parameters based on the type of network. Also traps or notifications can be reported to the management station in case of anomalies. Trend analysis can be done and preventive measures could be taken in time if the network and resource conditions are known before hand. If these parameters keep changing often, which is the case in networks like grids, extension of these services from a central point would greatly reduce human intervention and make the system very flexible. If the system were customizable to work with any network management service, then such a system would be comprehensive enough to include the extension mechanisms for multiple network management services in one solution. The user can also change the functionality dynamically depending on the type of management required. The thesis proposed provides all these solutions with minimal configuration.

Features. The features can be listed as

- It uses HTTP to transfer the agents. This has the advantage that HTTP is ubiquitous, standard and well accepted. This does not invent one more protocol and hence minimizes the infrastructure.

- It is optional and is included as a module.
- It is secure and has mechanisms to enforce authentication, encryption and authorization.
- It has modular design and hence works with any network management service.
- There are no installation hazards. It can just included as a module and only a configuration file has to be edited.
- It is portable across multiple platforms.
- Its functionality can be extended at runtime and the control is centralized.

Scope and limitations. The system can be useful in the following conditions

- In a heterogeneous system that is topologically distributed managed by different administrative entities and has different platforms viz. Virtual Organizations in grids.
- Memory-based extension mechanisms like SNMP/AgentX. Unfortunately file based extension mechanisms like MDS/LDAP of Globus, make the system to restart to read the modified configuration file. This can be handled in Unix systems by sending the signal SIGHUP.

.

LIST OF REFERENCES

[1] Symbol, "Network management in wireless environment", http://www.symbol.com/products/whitepapers/whitepapers_network_mgmt_in_wi.html, 2002, Accessed on 11/08/2002.

[2] Globus, "Monitoring and Discover Service", http://www.globus.org/mds/, 2002, Accessed on 09/09/2002.

[3] Wahl M., Critical Angle Inc., Howes T., Netscape Communications Corp. and Kille S., Isode Limited, "Lightweight Directory Access Protocol (v3)," 1997.

[4] Case J., SNMP Research, Fedor M., Performance Systems International, Schoffstall M., Performance Systems International and Davin J., MIT Laboratory for Computer Science, "Simple Network Management Protocol (SNMP)," RFC 1157, 1990.

[5] Daniele M., Compaq Computer Corporation, Wijnen B., T.J. Watson Research Center, IBM Corp., Ellison M., Ellison Software Consulting, Inc. and Francisco D., Cisco Systems, Inc., "Agent Extensibility (AgentX) protocol, version 1," RFC2741, 2000.

[6] Pagurek B., Wang Y. and White T., "Integration of Mobile Agents with SNMP: Why and How," Network Operations and Management Symposium, 2000. (NOMS'00). 2000 IEEE/IFIP, pp. 609-622, Honolulu, USA, 2000.

[7] Bieszczad A., Pagurek B. and White T., "Mobile Agents for Network Management," IEEE Communication Surveys, http://www.comsoc.org/pubs/surveys, fourth quarter 1998, Vol 1 No 1, 1998, Accessed on 08/08/2002

[8] Simoes P., Silva L.M. and Fernandes F.B., "Integrating SNMP into a Mobile Agent Infrastructure," Proceedings of DSOM'99 Tenth IFIP/IEEE International workshop on Distributed Systems: Operations & Management (DSOM'99), Zurich, Switzerland, 1999.

[9] Simoes P., Reis R., Silva L.M and Fernandes F.B., "Enabling Mobile Agent Technology for Legacy Network Management frameworks," Proceedings of International Conference on Software in Telecommunications and Computer Networks (SoftCOM '99), Split, Croatia, 1999.

[10] Rivalta P.C., "Mobile Agent Management," Master's thesis, Carleton University, Canada, 2000.

[11] Fielding R., UC Irvine, Gettys J., Compaq/W3C, Mogul J., Compaq, Frystyk H., W3C/MIT, Masinter L., Xerox, Leach P., Microsoft and Berners-Lee T., W3C/MIT,"Hypertext Transfer Protocol--HTTP/1.1," RFC 2616, 1999.

[12] Howes T. and Smith M., University of Michigan, "The LDAP Application Program Interface," RFC1823, 1995.

[13] Legrand I., Wu Y., Voicu R., Steenberg C., Ravot S. and Katageri S., "MONitoring Agents using a Large Integrated Services Architecture (MONALISA)," http://cil.cern.ch:8080/MONALISA/, 2002, Accessed on 10/03/2002.

[14] Apache HTTP server Documentation Project, "Apache 1.3 http server documentation," http://httpd.Apache.org/docs/, 2002, Accessed on 10/02/2002.

[15] Aker B, TangentOrg, "*Mod_repository* Apache module," http://software.tangent.org/faqs/*mod_repository*.html, copyright 1994-2002, Accessed on 09/08/2002

[16] EngelSchall R.S., "*mod_ssl* HOWTO," http://www.modssl.org/docs/2.8/ssl_howto.html, 2002, Accessed on 09/08/2002.

[17] Netscape Communications Corporation, "Introduction to SSL," http://developer.netscape.com/docs/manuals/security/sslin/contents.htm, 2002, Accessed on 10/10/2002.

[18] Marshall J., "HTTP Made Really Easy," http://www.jmarshall.com/easy/http/, 1997, Accessed on 10/10/2002.

[19] Ravichandran A., "Secure Communication services for distributed conference system," Master's thesis, Computer and Information Science and Engineering Department, University of Florida, 2002.

[20] Sun Microsystems, "Java Security Architecture," http://Java.sun.com/j2se/1.3/docs/guide/security/spec/security-specTOC.fm.html, 2002, Accessed on 10/15/2002.

[21] Karnik N.M. and Tripathi A.R., "Security in the Ajanta Mobile Agent System," Technical report, Department of Computer Science, University of Minnesota, 1999.

BIOGRAPHICAL SKETCH

Suchindra Katageri was born in India on 29th Feb. 1976. He earned his basic education from Basel Mission English Medium School, Dharwad, India, and completed his Bachelor of Engineering from Sri Jayachamarajendra College of Engineering, Mysore University, India. He pursued his Master of Science at the University of Florida, Gainesville, FL, majoring in computer science.