Implementing a Distributed Web-based Management System in Java

Adel Ghlamallah and Raouf Boutaba
University of Montreal, IRO Department
C.P.6128, succursale Centre-ville,
Montreal (Quebec) H3C 3J7
phone: (514) 343-6111 Ext. 3504, fax: (514) 343-5834
e-mail: {ghlamall, boutaba}@iro.umontreal.ca

Abstract - The distribution requirement of current and future networked environments has led to a shift from traditional centralized towards distributed management approaches. In this paper, a distributed architecture for network management is presented. The architecture implementation is based on an extensive use of emerging client/server, Web and Java technologies, the pursued objective being to promote flexibility, openness and portability.

1. Introduction

The operation of most of today's organizations lies on the services provided by the employed networking and computing facilities. The management of network and system resources is thus of paramount importance so as to ensure that these resources will provide their services efficiently. However, current and future networked systems are more likely to be composed of large numbers of heterogeneous software and hardware components. This make their management a complex task. Such complexity is often reduced by subdividing the management system into subsystems and by distributing management responsibilities over these subsystems, this way reducing management traffic, avoiding bottleneck situations and taking better advantage of system resources distributed in the network. This principle has led to the shift of traditional centralized management towards distributed management approaches based upon new concepts such as management domains [2], management by delegation [4] and others.

This paper presents an implementation of a distributed management system where autonomous software agents are delegated by a manager and act on behalf of this one to monitor and control the resources in the domain they are responsible for. The designed architecture is based on a four-tiers client/server model integrating a client side (the management interface), a server side which is the central part in the system implementing the management application logic. The server controls a set of intermediate agents delegated to perform specific management tasks. To allow for openness and portability, Web technologies and the Java language have been selected to implement the designed architecture.

The paper is subdivided into 5 main sections. Section 2 overviews a number of approaches to design network management applications using Web technologies. The third section briefly describes the Java language and highlights its capabilities to support the development of distributed network management applications. Section 4 introduces our design architecture and describes its components. Section 5 presents some implementation issues.

2. Web-based management architectures

There are two main architectures to develop web-based management applications: Two tiers and three tiers client/server architectures.

In the two tiers model [14], network managers access directly to each managed resource individually to carry out configuration, monitoring and controlling operations. Two parties are involved in this model: a client side allowing to access and display management information using a Web browser as an interface and HTTP as the communication protocol; and a server side where a Web server is embedded in each managed device to process client requests and to notify results in the form of HTML pages. In this case, each device is represented by an URL_address used by clients to access this device. Such access scheme makes network management easier, friendly and cost effective compared to current network management platforms.

In the three tiers model [5], three parties are integrated to provide management applications combining traditional platforms and a Web-based management. In this case, a web-server is added as an intermediate component accessed by the manager on one hand and accessing managed resources on the other hand. The first tier is the client side represented by a standard browser communicating with the server using the HTTP protocol. The second tier hosts the Web-server and the management application and access the management agents (third tier) representing the

0-7803-5030-8/98/$10.00 ©1998 IEEE

managed resources through a given management protocol (SNMP, CMIP, DMI, etc.). The advantage of this approach is the ability to build on existing management applications to offer a Web-based management interface providing this way an incremental solution to companies that wish to integrate a Web access to their management products.
The two tiers architecture is adapted to manage small networks and requires a separate URL address to each managed resource in the network. However, the management of large scale networks and systems requires more sophisticated tasks for the collection and processing of management data. For such networks and systems, the three tiers model seems more adapted since a server part communicates with all managed resources and provides a general view of the network state and enables a flexible access to the management application using one URL address.

3. Java and Web-based network management

Java is an interpretive programming language with interesting characteristics which can facilitate the implementation of network applications. It is a simple and a portable language as it can be executed on different platforms. The portability feature of the Java language is a considerable asset with respect to current heterogeneous networked environments. Using Java allows network products developers to reduce development and maintenance costs by providing a single product regardless of the target system environments and hardware platforms.
Java also allows interfaces to be dynamic by integrating so-called applets in HTML pages. This is an interesting feature for network management applications as it allows to display dynamic management information to users. Examples of such information are real-time state values of managed resource attributes or network traffic graphics.
Java applets can be downloaded from any server in the network and executed in standard browsers. However, some security constraints prevent Java applets to write or even to read information on the client host.
In addition Java programming facilitates the transport of objects through the network and hence satisfies the distribution transparency requirement. This way Java objects can be distributed dynamically at any time of the application life cycle. Such feature is of particular importance towards the distribution of network management functions.

4. Design Architecture

Our architecture for distributed network management is based on the notion of delegated agents [4] and the intermediate agent concept using a scripts MIB as proposed to the IETF [6]. The intermediate agents delegated by a management authority receive scripts to be executed and are supplied with special MIBs that allow them to control the execution of these scripts. A script execution leads to the invocation of management operations within SNMP agents.
Furthermore, our management system integrates a Web-based management interface. The overall management architecture is based on a four tiers client-server model: A client which is basically the Web browser; A server which is the central part in the system acting as a Web server and implementing the management application logic; a set of intermediate agents; and finally the abstract representation of the network managed resources provided in our system by SNMP agents. These components of the architecture are depicted by figure1. They communicate via standard Internet protocols which are HTTP, TCP and SNMP respectively.
Access to the management application is initiated by the client through the Web browser. The server, located on an intermediate host, continuously listens on a given communication port to receive HTTP requests from clients. The server handles these requests locally or directs them to the network using SNMP and/or TCP protocols.

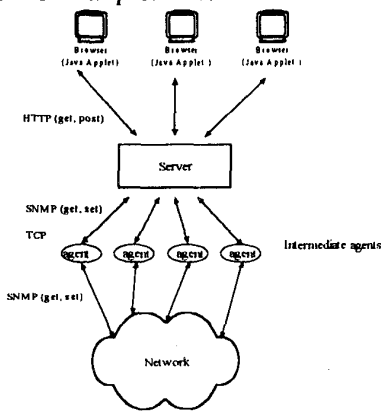

Fig. 1. Four tiers architecture

The following sections details the components involved in the management architecture.

4.1. The Client

The client is represented by a standard Web-browser where the interfaces will be displayed. These interfaces change during the evolution of the application by downloading HTML pages and applets from the server.
To access the management application, the client needs to know only the URL address of the station hosting the server and possibly the communication port number of the server. The latter is not necessary if

the server is assigned port number 80 which is the default port for Web servers [10]. Once the connection is established, the client is authenticated by a password in order to control the access to the application and thus prevent unauthorized accesses. This capability is particularly important in the open Internet context to protect critical management data.
Once authorized, the client firstly launch a discovery mechanism which displays in an HTML page the intermediate agents active in the network. It can then carry out different operations on the intermediate agents such as browsing scripts MIBs, discovering SNMP agents and delegating management tasks. Management tasks delegation can be performed according to two modes (direct or indirect modes) as explained in the subsequent sections.

4.2. The Server

The main module of the server is a Web server listening continually on its communication port for the requests issued from the browsers. This module acts as a simple web server accepting HTTP requests [11], processing these requests and sending responses before closing the connection. The first operation performed by this server module is the authentication of clients. In order to reinforce the security, the basic client authentication message (login and password) is encoded in Base64 [8]. The next requests issued by clients are handled differently depending of HTTP request types : Get or Post. The distinct modules involved in the server are illustrated in figure 2.

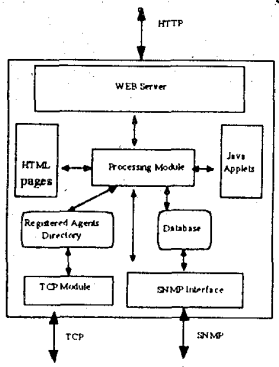

Fig. 2. Server architecture

4.2.1. Web server

The Web-server module constitutes the core of the server. It deals with the client requests according to two schemes. In the simplest scheme, it creates HTML pages dynamically by integrating applets or using the existing HTML pages and sending them back in response to client requests. In the second scheme, the server calls upon the processing module to perform the necessary operations. The Web-server module is a Java program based on multithreading and hence can handle several requests simultaneously.

4.2.2. Processing module

This module is launched by the Web server module when a request requiring a particular processing is received. Its main task is to interpret client requests and translate them into SNMP operations to be forwarded to the agents through the SNMP interface module.
When the request concerns the delegation of a script, the processing module uses the services of the TCP module to transport the script code or the URL address from where to download the script. In both cases, the script code or its URL address are transmitted to the target intermediate agent. Management information related to intermediate agents are stored within a database and are used by the processing module to build dynamically HTML pages which are then returned to the web server module. A typical example is the situation when the web server requests the processing module to discover the intermediate agents active in the network. In such a scenario, the processing module builds an HTML page based on the information available in the intermediate agents directory.
The processing module is similar to the CGI (Common Gateway Interface) concept but it is based only on the Java language whereas the CGI programs are generally written with scripting languages like Perl. In addition to the portability advantage of our Java based processing module, it is more efficient than CGI in that it uses the Servlet concept recently proposed by Javasoft [12]. The efficiency issue will be more detailed in the implementation section.

4.2 .3. SNMP interface

The SNMP interface module represents the interface between the processing module and the intermediate agents. It is invoked by the processing module to handle SNMP requests. Because, SNMP is the protocol used between the different levels of intermediate agents, the only commands communicated between theses levels are of type Get, GetNext, Set and Trap [13]. The information received from the agents are stored in the database and/or sent directly to the processing module.
To trigger the execution of a script located at the intermediate agent level, the SNMP interface module sends a Set command to this agent pointing to the «execute script OID» described subsequently.

4.2 .4. TCP Module

The TCP module carries out two main tasks. The first one consists in recording the information related to new registered intermediate agents. Indeed, once an intermediate agent becomes active, it informs the server and communicates some relevant information such as its IP address, the number of its communication port, and the managed domain (i.e., the set of managed resources) it is responsible for. The other main task, consists in ensuring the delegation of management scripts by transferring them to the appropriate intermediate agents.

4.3. Intermediate Agents

The intermediate agents are flexible enough to perform management tasks defined at application initialization or during application execution. They can be dynamically expanded with new functionality without stopping the management application. This flexibility is important with respect to the dynamic behavior of the managed network and to facilitate management applications evolution by dynamically integrating new management functions.

The main advantage of intermediate agents is to reduce the overall management task complexity by separating management concerns and distributing management responsibilities over these agents. Another advantage is their vicinity to the managed resources which allows to reduce the management information traffic in the network. The traffic generated by the polling of SNMP agents is mainly located at the level of the intermediate agents whereas the traffic with the server is considerably reduced and limited to significant reports generated by the intermediate agents after processing the information collected from the SNMP agents.

The intermediate agent is responsible of the management of a set of resources, commonly known as a managed domain. The grouping of resources into managed domains can be done according to logical or physical criteria [2]. In our management application, every network segment is a domain managed by a single intermediate agent. The resources attached to a network segment are endowed with SNMP agents.

The intermediate agent is a Java program based on multithreading which enables the handling of several requests simultaneously. It runs as a background process and is supported on both UNIX and Windows platforms. Once launched, it loads the scripts MIB and proceeds to its parsing (conformity to ASN.1 standard) which may lead to the generation of errors message. If not, the intermediate agent registers at the level of the server (in the corresponding directory entry), and waits for SNMP requests. The internal structure of the intermediate agent is shown in figure 3.

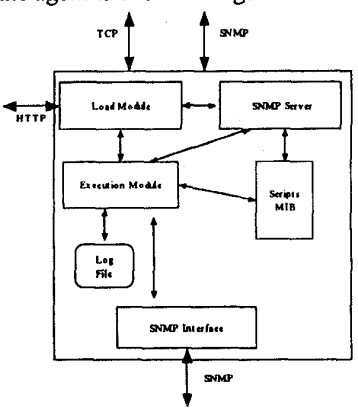

Fig. 3. Intermediate Agent architecture

4.3.1. The SNMP Server

When a new request is received at its communication port, the agent starts a new thread to process this request. A Get command leads to read the value of the variable identified by the OID similarly to standard SNMP agents. On a Set request, the agent can update the value of the indicated variable, starts the execution of a management operation, download a script object from a remote location, or open a connection to receive the script from the server.

4.3.2. The Scripts MIB

The Scripts MIB contains the list of scripts known to the intermediate agent and holds control information for these scripts execution management. The following attributes have been defined for scripts execution control:

- ScriptRef: contains the name that identifies each script known by the agent.
- ScriptStatus: indication of script state (wait, execution, or error).
- ScriptSrc: indicates the source of script. It contains the address of the remote site if the script is downloaded, and left empty if it is local.
- ScriptExec: indicates how many times this script has been executed by the agent.
- ScriptExeARGS: a list of arguments required for the script execution.

In addition the MIB contains references to executable objects which allow when invoked to receive, download or execute scripts. These object references are defined as follows:

- ExeClass : allows to load and execute a script.
- ReceiveClass : allows to open a connection and receive a script from a manager.
- DownLoadClass : allows to download scripts from

remote sites.
Each of these object references has an OID in the MIB known by the manager. When receiving an SNMP Set command corresponding to one of the OIDs, the corresponding object reference is activated and the appropriate operation is executed.

4.3.3. The Execution Module

If the received request corresponds to the execution of a management script, the agent checks first if this script belongs to the list of known scripts. If the invoked script is unknown, the intermediate agent notifies the manager requesting the script which can either downloaded or sent directly. Otherwise, the intermediate agent loads and execute the script using the ClassLoader Java object.

4.3.4. The Load module

The Load Module is a Java program which performs the copying of a script file from a remote server given its URL address. The latter is sent by the manager as a parameter of the Set command. A script can be also transferred directly by the manager. In the latter case, the intermediate agent opens a TCP connection on a given port and closes it after receiving the script file. In both cases, the load module is initiated by the execution module. Moreover, it is the load module that is responsible, at the starting time, of registering the intermediate agent in the agents directory maintained by the server.

5. Implementation Issues

We have implemented a web-based management system and application using the standard set of Java classes [3] and SNMP API provided by the Advent product [1]. The choice of Java as a programming language has been motivated by the networked facilities and cross-platforms interoperability it offers. Advent SNMP API allows to build SNMP-based management applications and provides support for synchronous and asynchronous SNMP communication between the manager and the agents. The following sections point out some issues related to the implementation of the server and the intermediate agent.

5.1. Server Implementation

The Server is implemented by a Java program supporting multithreading. Each request processed by the server is linked to thread which execute it. The threads are launched within the main program. To implement a flexible behavior of the server, servlet classes are used. servlets are protocol and platform independent server side components written in Java to realize specific tasks.
As part of the server functioning, the processing module is invoked by the Web server module whenever an information request is received from the manager. In this case, a process is loaded dynamically by the server as a servlet. Mainly this servlet will construct dynamically new HTML pages containing the requested information. As an example, when a discovery command is received, the servlet will automatically retrieve the information about the registered intermediate agents from a local directory and generates an HTML page within an applet at the client Web interface.

5.2. Intermediate Agent

The intermediate agent is implemented by a Java program supporting multithreads and designed to run on any host that has a Java Virtual Machine. It processes SNMP requests whenever received from the server. For example, when it receives a SET command pointing to the OID corresponding to the execution class ExecClass, it invokes the execution module with the name of script received as a parameter in the Set command. This example is illustrated in figure 4.

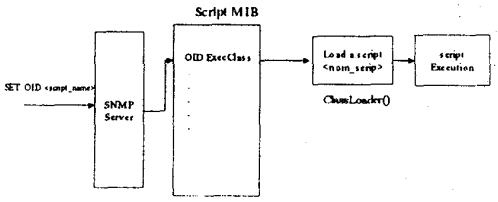

Fig. 4. Script execution

Before to proceed with the execution, the execution module checks first if the script name is in the list of known scripts located in the intermediate agent scripts MIB. After, the loading operation is performed using a subclass of the Java Load class. The defined loading subclass checks if the invoked script has been already loaded in the system. For security purposes, it also checks if the byte-codes incoming from the network are legal Java class file. Finally, it instantiates the invoked scrip object. The loading of a script is done at run-time when the script is to be executed. This way, the intermediate agent don't has to know in advance about the script (e.g., the script name). This dynamic capability allows for a high level of flexibility in overall the management process.

5.3. Scripts Transfer

The transfer of scripts between the server and the agents can be achieved according to one of the following schemes:
1) Direct transfer mode: once the manager through

his/her Web browser decides to transfer a management function to an intermediate agent. As the management function is located at the server side, an SNMP request is built by the server processing module and sent to the target agent through the SNMP interface module. The SNMP request encapsulates the OID corresponding to the ReceiveClass object reference which in turn will initiates the script loading at the level of the agent load module. The latter opens a TCP connection and sends a READY message to the server for starting the transfer of the script object. When the script file correctly received, the agent close the connection. Figure 5 shows the steps involved in this script transfer scheme.

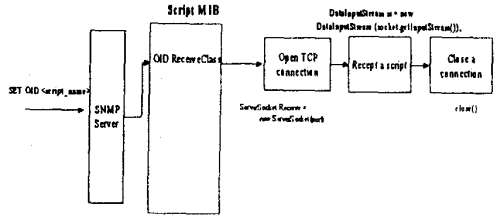

Fig. 5: Script direct transfer mode

2) Indirect transfer mode: The server sends only the URL address of the server where the script is located. When receiving the SET command encapsulating the OID corresponding to the DownLoadClass object reference, the intermediate agent initiates the control operations depicted in figure 6.

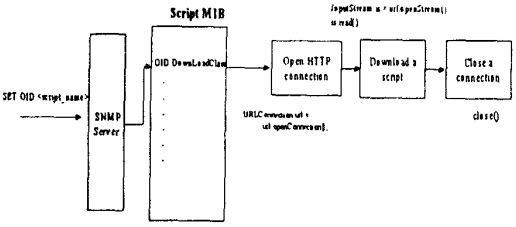

Fig. 6: Script indirect transfer mode

6. Conclusion

This paper presented a practical architecture for building management applications based a four tiers client/server model and emerging Web-technologies. The architecture integrates intermediate agents delegated by the manager to perform specific management tasks at the level of a managed domain. A Server is introduced to coordinate the agents activity and to provide a gateway between the agents and the Web-based management interface. The distributed feature of the proposed architecture allows to separate management concerns and hence to reduce the complexity inherent to the management task of large scale networks and distributed systems.

This paper has also presented an implementation of this architecture fully based on the Java language. This has allowed for a high degree dynamicity at three levels : the client level (Web browser) using Java applets to display dynamic management information; the second level, is the Server using Java Servlets loaded dynamically to perform specific management tasks such as building HTML pages encapsulating appropriate Java Applets; and the Intermediate Agents level using a defined sub-class of the standard Java Class Loader to dynamically load and execute management script objects.

The Java implementation choice has been motivated by the language portability feature to reduce development and maintenance efforts by allowing the same management application to run in all environments supporting a Java Virtual Machine. This implementation experience has confirmed the network programming facilities of Java in the specific area of network management applications development.

References

[1] Advent Network Management, Inc. [On-line], URL http ://www.adventnet.com

[2] R. Boutaba, "A Methodology for Structuring Management of Networked Systems", In Proc. Int. Conf. on Advanced Information Processing Techniques for LAN and MAN Management, IFIP Transactions, pp. 225-242, 1994.

[3] G. Cornell, C. S. Horstmann, "Core Java ", Prentice Hall, 1997

[4] G. Goldszmidt , "Distributed Management by Delegation", PHD thesis, 1996.

[5] M. Jander, "Welcome to the Revolution", Data Communications, November 1996

[6] D. B. Levi, J. Schoenwaelder "Definition of Managed Objects for the Delegation of Management Scripts", Internet Draft, 1997

[7] G. Pavlou, G. Mykoniatis, Jorge-A. Sanchez-P "Distributed Intelligent Monitoring and Reporting Facilities", in DSEJ, special issue on Services for Managing Distributed Systems. 1996

[8] RFC 1113 "Privacy Enhancement for Internet Electronic Mail", August 1989

[9] RFC 1213 "Management Information Base for Network Management of TCP/IP-Based Internets: MIB-II", 1991

[10] RFC 1700, "Assigned Numbers", October 1994

[11] RFC1945 "Hypertext Transfer Protocol-HTTP/1.0", May 1996

[12] P. Sridharan, "Advanced Java Networking", Prentice Hall 1997

[13] W. Stalling: "SNMP, SNMPv2 and RMON-Practical Network Management", Addison Wesley, 1996

[14] The Simple Times Volume 4, Number 3, 1996.