A standards-based network monitoring system

Javier F. Díaz – Laura A. Fava – Alejandro J. Sabolansky
Computer Science School, National University of La Plata
50 and 120, 2nd floor, La Plata, Buenos Aires, Argentina.
{jdiaz, lfava, asabolansky}@info.unlp.edu.ar

Abstract— **This article describes a system meant to provide monitoring services using Simple Network Management Protocol (*SNMP*) and the Java 2 Platform, Standard Edition (*J2SE*). The architecture of the system is based on JMX (*Java Management eXtensions*), and is composed of agents and managers. Each Java agent is a mediator between a set of SNMP agents of a network and the management applications.**
When the number of administered SNMP agents is large, assigning a Java agent to handle each one of them would be not only complex, but unfeasible if the device subject to monitoring does not support Java or if the device is unreachable for security matters. To avoid this problem without interfering with the security policies of a network, we propose an architecture in which a Java agent or master concentrates and sends the management application all the monitoring information of the SNMP sub-agents. This proposal simplifies the process of administration of the devices, increases the internal security of the monitored network, allows monitoring of not java-enabled devices and diminish the network traffic considerably. Additionally, the system provides a manager with a simple but high quality user interface to facilitate the remote administration of the Java agents.

Index Terms— **Software/Resource Management, Agents, Java Management Extensions, Simple Network Management Protocol**

I. INTRODUCTION

In the field of network and device management, SNMP[1] has been the dominant protocol for many years. A large amount of vendors have distributed many devices with SNMP management capabilities. In addition, Sun has extended the Java platform to provide JMX management and monitoring of applications and services, encouraging the development of non-proprietary solutions.
Considering, then, that most of the devices have SNMP and exploiting the new resource management characteristics of the Java platform, we designed an architecture which simplifies the process of secure network monitoring applying these technologies.
This article is structured in the following manner. Section II presents typical scenarios signaling the main management needs. Section III briefly describes SNMP, the protocol for standard management and summarizes the most significant characteristics of JMX[2], the standard Java specification to implement applications and agents. Section IV presents the JMX-based architecture and a prototype called Cafetín which implements it. This paper is ended with conclusions.

II. SCENARIO

The proliferation of networks in organizations, together with the geographical distribution of their dependencies, generates a scenario in which the network devices, servers and services are not concentrated in a single location. If we add to this the importance of having availability in 7X24 services, this typically leads to administrators using many monitoring tools and techniques to provide immediate answers to potential anomalies, both in the services and in the infrastructure in which they are mounted.
It is true that many manufacturers provide the user with monitoring tools, but most of them use proprietary protocols, which impedes their integration and correlation with the information generated by equipment manufactured by another company.
There are also tools based on standard protocols such as SNMP for the management of remote devices, which facilitates resource monitoring, although many of them generate a great amount of SNMP traffic and require access to the internal network structure. Some of these have management applications with very textual and unfriendly user interfaces.
In light of these reasons and having SNMP available in most of the devices, we propose a system to improve these deficiencies and facilitate the network management with a great number of SNMP agents.

III. APPLIED TECHNOLOGIES

A. Simple Network Management Protocol (SNMP)

SNMP is the dominant technology in the management field. It defines an architecture made up of a set of management stations or managers and network elements or agents. The management stations execute applications which enable monitoring and control of the network elements, whereas the network elements are devices with agents responsible for the execution of the management tasks required by the said stations. Lastly, the management protocol called SNMP is

used to communicate management information between the stations and the network elements[3,4].
Basically, the management application can obtain data from the device and send data to the device, and the agent can send unsolicited data (called trap) to the said application. Furthermore, each device contains in its memory very well defined information with a tree-like structure, called MIB[1]. An MIB is a complete database which defines the information available from a device for the network or the management application, which is accessed using the SNMP protocol.
SNMP has evolved and there are three versions of it: SNMP version 1 (SNMPv1), which defines the structure of the management information [5] and the SNMP protocol, SNMP version 2 (SNMPv2), which offers improvements in the types of packages and structures of the MIBs [6,7] and its last version (SNMPv3), which has undergone significant changes in relation to its predecessors, above all in security matters [8,9]. SNMPv3 was launched to cover the deficiencies of its antecessors while improving privacy, authentication and authorisation aspects.

B. *Java Management Extensions (JMX)*

The JMX technology was developer through the Java Community Process[2] as two related specifications: Java Management Extensions Instrumentation and Agent Specification (JSR3 3) and Java Management Extensions Remote API (JSR 160), plus JSR 255 of 2007, which updates the previous ones. The first implementations of these specifications were included as of version 5.0 of the standard Java platform. The latest version of the standard J2SE platform is 6.0, which includes JMX 1.4.
The JMX specification defines a management architecture and a set of APIs which describe the components of this architecture. The JMX 1.4 specification specifies an architecture composed of three levels, which are described below [15].

Instrumentation Level

The instrumentation level provides a specification to implement manageable JMX resources. A manageable JMX resource may be an application, a service implementation, a device, a user, etc. This is developed in Java and may be managed by JMX applications.
The instrumentation of a given resource is provided by one or more managed beans, known as MBeans. The instrumentation of a resource allows it to be managed by means of the Agent Level described below. MBeans do not require information regarding the JMX agent they will operate with. Additionally, this level specifies a notification mechanism. Agents compatible with JMX can manage JMX resources automatically.

Agent level

The agent level provides a specification to implement agents. The agents control the managed resources directly and make them available for the remote management applications. According to SUN's documentation, the agent is commonly located in the same machine as the SNMP agent.
A JMX agent is an entity which runs in a JVM and acts as a mediator between MBeans and the management application. It is composed of an MBean server, a set of MBeans representing the managed resources, one or more implemented services such as MBeans and at least one Connector or Protocol Adapter for communication.

Distributed Services Level

This level provides connectors which allow communication between JMX agents and management applications to occur. Each connector provides the same remote management interface through a different protocol. When a remote application uses this interface it can connect to a JMX agent in a transparent manner through the network, no matter which protocol it is using.

IV. CAFETIN: A MONITORING JMX-BASED SYSTEM

The architecture proposed is based on JMX and is composed of, as was said before, an agent and a management application, both compatible with JMX. The Java agent is the one responsible for exposing the management information of a set of SNMP agents to the remote management applications.

A. *The Cafetín Agent*

Because the agent is implemented in Java, it must be executed in a device that has a JVM. It can be installed in the same device on which the SNMP agent is running if it also has a JVM, or in any other machine Java enabled. The most common procedure is to install the Java agent in the device to be monitored (this is the case of agents monitoring firewalls, web servers, J2EE servers and even the JVM itself.) However, there are situations in which the Java agent must be in a different device than the one that is executing the SNMP agent, either because the device to be monitored does not have the capacity to execute a JVM –as is the case of most routers and switches- or because the device cannot be accessed on account of filtering or routing policies. Both situations are present in the typical scenarios presented in section II where commonly, in addition, a great number of SNMP agents must be handled. It would be difficult to install a Java agent in each device (which has the SNMP agent) to be monitored. For the reasons mentioned, we considered it appropriate to define a hierarchical structure of main agent or master and SNMP

[1] Management Information Base (MIB): is a collection of information organized hierarchically. These are accessed using a protocol such as SNMP.
[2] Java Community Process (JCP): is an organization which guides the development and controls the technical specifications of Java technologies.
[3] Java Specification Requests (JSRs) are final specifications and proposals for the Java platform.

sub-agents, in which the master agent concentrates and sends information about its sub-agents to the management application. The management application only communicates with the master agent and accesses the SNMP sub-agents transparently, as if the information resided in the master agent. As can be observed in Figure 2, a Cafetín agent can monitor one or many SNMP agents, thus, only one Java agent needs to be installed in a machine to monitor *N* SNMP agents. Moreover, the Cafetín agent reads the IPs of the devices to be monitored when it starts, which makes it adapt well to be executed in the same machine of the SNMP agent and monitor only one SNMP agent.

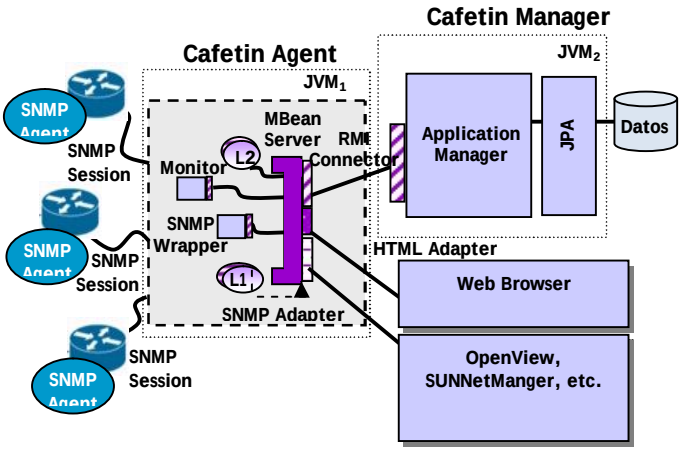

Figure 1: Cafetín Architecture

As can be observed in Figure 1, the agent has an MBeanServer which functions as an object container, which model a resource or service. These resource or management services, in the shape of MBeans, can be registered and removed dynamically from the management application. The proposed architecture has defined, among others, the following MBeans:

SNMP Access Service (WrapperSNMP): this MBean works as a wrapper of the SNMP agent. It provides the services of an SNMP agent.
What is the advantage of having a wrapper instead of connecting directly through a SNMP protocol adapter? Minimizing the network traffic. Without this functionality, we should establish a connection directly from the manager application to each of the SNMP agents. In this way, the Cafetín agent is the one doing the pollings to an SNMP agent or groups of SNMP agents belonging to a network and reports to the management application only in special situations.
To achieve this, many different monitors are used and given the task of monitoring a given attribute. This MBean is dynamic, i.e. the attributes to be monitored can be defined in run-time, through a screen from the manager which allows for the specifications of the monitor parameters.

Monitoring Service: MIBs attributes can be monitored using the monitoring services offered by JMX. The monitoring services facilitate the observation of the time variation of the attribute values in the MBeans representing the MIBs and issue notifications when faced with certain situations - changes in the attribute values, attribute values which reach a threshold, etc.-. The monitors send notifications when the observed values gather certain conditions, mainly when they match or exceed a threshold. These conditions are specified when the monitor is started or dynamically from a management application using the corresponding graphics window for defining monitors as shown in Figure 3.
The value observed by the monitor is called a "derived gauge", and it can be the exact value obtained from an attribute at a precise time or the value difference between two consecutive observations. The time interval during which an attribute is monitored is called "granularity period" and is measured in milliseconds. There is also a data, called "offset" which can be configured, and indicates when the threshold increases once the top value has been reached.

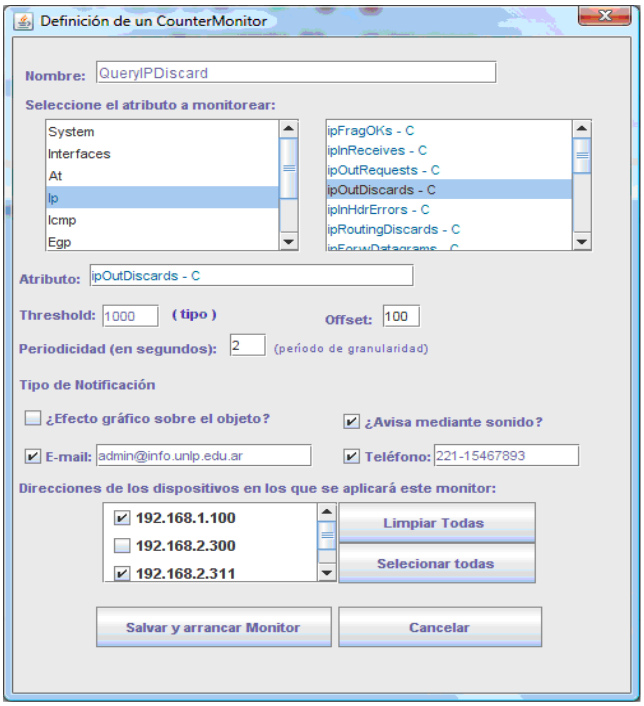

Figure 2: Windows for definition of a CounterMonitor from the Management application

The management application provides graphics windows (Figure 2) to define different types of monitors according to the type of data of the MIB attribute. These windows also allow to specify whether all the network devices monitored by the JMX agent or only some network IP addresses will be monitored.
The IP addresses are automatically loaded when the administrator to selects a widget that represents a remote java agent (Figure 3). Each IP address in the list corresponds to an SNMP agent that such Java agent handles.
This functionality shows an aspect of the extensibility of the system. The manager application instructs the agent to run a new MBean (monitor). As of then, the person in charge of the network will receive e-mail or telephone notifications from

this new monitor, and in addition will be able to visualize graphically the anomaly on the corresponding User Interface widget.

Broadband Controlling Service (BandWithMBean): this MBean measures the use of a WAN connection, with the goal of limiting its use. WANs use full-duplex connections, i.e. point-to-point connections where both devices can transmit and receive at the same time, because both know there is only one other device sharing the said connection. As MIB-II variables are stored in counters, we can take two polling cycles and calculate the difference between them.
We can use the difference between two polling cycles of the ifInOctets attributes (represents the amount of incoming traffic cycles), the difference between two polling cycles of the ifOutOctets attribute (represents the amount of incoming traffic octets) and the value reported by the snmpifSpeed attribute (which represents the interface speed.)

The agent implemented is extensible; it can add new MIBs or services in run-time if the remote applications instruct it to do so. It can also monitor attribute values and issue notifications of undesirable situations by different means, or longer, to perform an automatic action to improve such situations.

B. The Cafetín manager

Other main goal of Cafetín was providing a collaborative, web-based and high-quality user interface. For this reason we decided to use Java Foundation Classes[9] for the construction of the UI. The manager application Cafetín can be used in two manners, as an **Applet Swing** or a **Swing desktop application**. Both have the same GUI, the difference is in how they work. In the case of the applet, the administrator needs to work in the browser, whereas with the application the administrator has to access the main page of Cafetín one time and click on a link. This will cause the Java Web Start[10] to download the application to work in an independent browser window. Figure 3 shows an image of the management application GUI with the JMX agents and a small window with the information of the SNMP agents it is monitoring.

The JFC, is a set of technologies which facilitates the development of GUIs for desktop applications and for applets. It is part of the Java platform as of version 1.2 and contains a set of GUI components with pluggable look & feel, known as Swing components, which, among other things, manipulate 2D advanced graphics and images, interact with accessibility technologies and create applications with automatic internationalization. Although the design of the GUI was an important step in the development of the management application, the main goal was the construction of a portable, extensible and dynamic tool to assist administrators in the processes of configuration, performance administration and network troubleshooting. As regards ***configuration***, the manager application allows the construction of a visual network map and the indication of device IP address and other data necessary for communication with the JMX agent. Regarding ***network faults***, the GUI provides a permanent network visualization, allowing any user to see the topology and its status. A set of colors and visual effects show the status of each device, allowing a simple observation to provide information on the general status of the network. The monitors are responsible for communicating critical states by means of significative icons, as well as by e-mail or telephone. Additionally, it has a history fed by monitoring, which allows the administrator who solved the problem to input the mechanism he used to do so. This information can be consulted by date, type of error, device, etc., and can be useful to solve similar future problems.

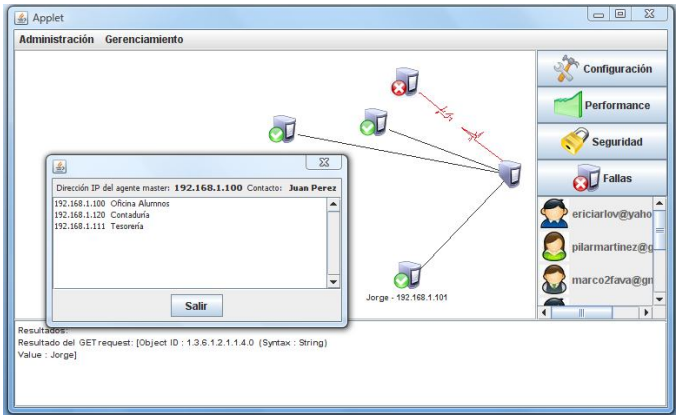

Figure 3: User Interface of the management application

C. Communication between agents and managers

Cafetín makes use of JMX's distributed services level, i.e. of the protocol adapters and connectors for the different communications. The manager application and the JMX agents use RMI to communicate, however, the JMX agents can be accessed through HTTP or SNMP as well, because of the protocol adapters described below.

The agent has an *RMI Connector* which makes the information of the said agent available to the remote management applications or clients. The agent has an RMI server connector registered and the management application makes use of the client part of the RMI connector to establish the communication. The client connector exposes a remote version of the MBeanServer interface. Each client connector represents an agent to which the manager wishes to connect. In order to do this, the manager must know the host and RMI port. The RMI server connector listens for application connection requests, creates a connection for each request and replies everyone simultaneously. It also has an *HTML Protocol Adapter* which acts as an HTML server. It allows any browser to access the agent through the HTTP protocol, to manage all the MBeans in the agent. In the case of Cafetín, it was especially incorporated into the agent with a tool to accelerate the development of the agent and the testing. This

adapter is implemented as a dynamic MBean. When the HTMP protocol adapter is started, it generates a TCP/IP socket, listens for managers connecting to the agents, and waits for requests. It also includes an ***SNMP Protocol Adapter***, in charge of translating data from an MBean to an SNMP MIB and using the SNMP protocol to transport the monitoring information to the interested listeners. This adapter does not interact with MBeans in the same manner the other adapters and connectors do. This is because the SNMP data are in the MIBs, and only MBeans representing MIBs can be managed through SNMP.

D. Communication with the Data Base

For the persistence of the Cafetín information in the data base, we used Java Persistence API or JPA[11,12], the persistence management and object/relational mapping standard interface of the Java Enterprise Edition 5.0 (JEE 5) platform. JPA is supported by most of the J2EE container providers. However, the new standard establishes that the JPA engine must be capable of running outside an EJB 3.0 execution environment. For this reason, any J2SE application, as is Cafetín, can make use of JPA without the need for an EJB 3.0 container or Java EE server.

To access the database we implemented a data access layer, commonly referred to as DAO, essential in any application architecture. This layer provides a separation between what regards object persistence and the data access logic of any particular persistence mechanism or API, allowing flexibility to change the persistence mechanism without necessarily effecting changes in the logic of the application interacting with the DAO. Currently, the persistence is being implemented with JPA, but could be modified to use JDBC or any other API without major changes.

V. CONCLUSION

We have proposed an architecture and implemented it with the J2SE standard, both for the master agent and the management application. We have obtained an extensible, dynamic and portable agent/manager application. These characteristics are the result of using Java and its JMX, JPA and JFC standards. The use of JMX made possible the development of the agent and facilitated the run-time remote resource and services addition, JFC facilitated the construction of a high-quality GUI and JPA permitted the implementation of object relational mapping.

The agent/manager application in its whole facilitates the administration of devices of any brand and type and reduces the network load, providing an intermediary JMX agent between the management application and the SNMP agents. It also makes its management easier, especially when we consider a great number of SNMP agents.

Finally, the management application to allow programmers to focus on the management functions rather than the underlying SNMP operations.

REFERENCES

[1] Managing Internetworks with SNMP, Mark A. Miller, ISBN 1-55851-561-5, second edition, 1995.

[2] Welcome to the JMX Technology Home Page, http://java.sun.com/javase/technologies/core/mntr-mgmt/javamanagement/

[3] J. Case, K. McCloghrie, M. Rose, Protocol Operations for Version 2 of the Simple Network Management Protocol (SNMPv2), RFC 1905, enero 1996.

[4] J. Case, A Simple Network Management Protocol, RFC 1157, mayo 1990.

[5] M. Rose, Structure and Identification of Management Information for TCP/IP-based Internets, RFC 1155, mayo 1990.

[6] J. Case, K. McCloghrie, M. Rose, Introduction to Community-based SNMPv2, RFC 1901, enero 1996.

[7] J. Case, K. McCloghrie, M. Rose, Transport Mappings for Version 2 of the Simple Network Management Protocol (SNMPv2), RFC 1906, enero 1996.

[8] B. Wijnen,R. Presuhn, K. McCloghrie, View-based Access Control Model (VACM) for the Simple Network Management Protocol (SNMP), RFC 2275, enero 1998.

[9] Project Swing (Java Foundation Classes), http://java.sun.com/products/jfc/

[10] Java Web Start Technology, http://java.sun.com/javase/technologies/desktop/javawebstart/

[11] Java Persistence API, http://java.sun.com/javaee/technologies/persistence.jsp

[12] Java Persistence with Hibernate, Christian Bauer, Gavin King. Manning, 2007.

[14] SNMPv3: "A Security Enhancement for SNMP", William Stallings. http://www.comsoc.org/livepubs/surveys/public/4q98issue/stallings.html

[15] Java Management Extensions (JMX) spacification 1.4, http://java.sun.com/javase/6/docs/technotes/guides/jmx/JMX_1_4_specification.pdf