Abstract

The present work is motivated by an interest in exploring the PUSH model in Web-based management, with a special look at the network management architecture, called JAMAP (Java Management Platform), proposed by Martin-Flatin in [4,5,6]. Before we present a prototype, called Java Push-based Network Management (JPNM), based on the same theoretical background, we describe how to implement the asynchronous transmission over Hypertext Transfer Protocol (HTTP) with PUSH model and introduce the use of Enterprise Java Beans (EJB).

KeyWords : PUSH model, Web-based management, HTTP, JAMAP, JPNM, EJB.

1. Introduction

Through the Web-based management we can exploit all the possibilities opened up by using HTTP instead of the traditional Simple Network Management Protocol (SNMP) [8,9,13].

A closer look reveals that the SNMP and HTTP are not mutually exclusive. However, one of the limitations of the first Web-based agent demonstrations that used HTTP was that the current version of the transport protocol did not support the notion of sending asynchronous alerts, like trap messages under SNMP [9]. This happens because the HTTP is strictly based on request-response paradigm [6].
Our primary focus on using PUSH model with Web-based management is the transmission of asynchronous traps via HTTP and later the use of it in the regular and ad-hoc management on the Web.

We describe the JAMAP architecture and compare its implementation with the system called Java Push-based Network Management (JPNM).

2. **PUSH vs. PULL Model**

Through the PUSH model the servers can send asynchronous messages to specified clients without the client requesting the information. The opposite of PUSH model is the more traditional client/server model called the PULL model [9].

The PULL model is based on a request-response paradigm, like polling in the SNMP management platform. The client requests and then receives the data from the server. The transfer is always initiated by the client [2, 6].

The Web is essentially a PULL resource. By typing in a URL address or clicking on a hypertext link, information is received from a Web server exactly as requested [15].

The PUSH model defines the transmission of asynchronous information to the client without having to request the information every time it is sent. There are two paradigms for the implementation of the PUSH model [6,9]:

- Publish/Subscribe/Distribute
- Broadcast
In the first paradigm, the client registers his purpose in receiving certain information from the server. The information is created on the server and is periodically sent to the client.

During a Broadcast, the server sends the most up-to-date information to all of its clients. In spite of being simpler than the former, it is less efficient and does not apply in all situations.

The Publish/Subscribe/Distribute paradigm is the most suitable implementation for the PUSH model in network management. That way, the manager communicates with each agent only once, during the subscription to an MIB (*push data definition*) variable, where it specifies the frequency (*push frequency*), with which the agent must send his values (*push data*) in the distribution phase [5].

The PUSH model, instead of the PULL model, decreases management information traffic. This happens because after the subscription phase, there is no additional traffic from the manager to the agents, except in cases where the manager desires to alter a subscription. Additionally, in the PUSH model, some of the CPU load is transferred to the agent since the data transfer is initiated by the agent without a previous request from the manager [2].
3. JAMAP Overview

The JPNM (Java Push-based Network Management) system is completely based on JAMAP (Java Management Platform) described in detail in [4] and [2] (see Figure 1).

The system is composed of Java applets and servlets that communicate in HTTP by means of persistent Transmission Control Protocol (TCP) connections and transmit data in Multipurpose Internet Mail Extensions (MIME) types. MIME allows the browser to deal with pages that are not in HTML.

The manager is divided into two elements: the centralized management server and any remote computer equipped with a web browser and functioning as a management station,
from which the administrator configures and queries the data. The management server is responsible for storing the MIB data in the database via JDBC (Java Database Connectivity).

The agents possess a schedule database defined by the administrator at the management station and at each cycle they send their MIB variable values to the management server.

In the Publish/Subscribe/Distribute paradigm, the system has three phases. The publication and subscription phases are run from the remote management station with a Station Applet for the MIB Data Subscription and Event Notification Subscription modules, where MIB variables and rules for trap notification supported by each agent are viewed and selected.

From there, the subscription information is forwarded to the agents, where it is stored in a schedule database. At each frequency interval the Agent Servlet through the Distributor module sends the MIB variable values to the Manager Servlet on the management server. The rules configured in the two initial phases are used in the Data Collector and Event Control modules.

In the case of regular management, the management data is sent to the data server and remains in a database that can be queried later on from the remote management station. In ad-hoc management the data is sent directly to the remote station and viewed in the MIB Values Reception module. Since the traps are asynchronous, it is not necessary to report the frequency, as they are only sent when their occurrence is detected and are viewed through the Event Reception module.
In this new architecture, the MIB is the most important element of the SNMP model still being used. With the protocol migration from SNMP to HTTP the MIB data are coded in MIME types that can still be compressed and encrypted. The persistent (keep-alive) connections available since HTTP 1.1, allow the transmission of varied information over the same connection, thereby reducing network overload or possible latency. The distribution of version updates and the flexibility of access to management data are facilitated by the inherent characteristics of the Java language.

4. Comparing JAMAP with JPNM

The [2] and [4] references presents the JAMAP project and the technologies used for its realization. In a nutshell:

- It was entirely written in Java using the Linux port of JDK (Java Development Kit) 1.1.6 by Blackdown [WWW04];
- The graphical interfaces was implemented using the NetBeans Developer 2.0 (now called Forte by SUN Microsystems®). It used SWING classes;
- The applets was tested with Netscape Communicator 4.5 under Linux 2.1.125 (glibc), Microsoft Internet Explore under Windows NT 4.0 and the applet viewer of SUN’s JDK Linux port;
- For development, the servlets was supported by Jigsaw Web Server 2.0.1 from the W3 Consortium [WWW05] and later by Jserv 1.0 module for Apache Web Server [WWW06];
- It reused some classes of the AdventNet SNMP suite [WWW03] like MibTree, MibNode, SnmpTarget, SnmpVar and SnmpTable classes. Also, it used the Util class of the HTTP
Client package written by Tschalär [WWW07]. Finally, it used the SMTPConnection class from IBM’s AlphaWorks SMTP package [WWW08] and the sun.tools.javac.Main class of SUN’s JDK Java Compiler to implement the dynamic compilation of the rules.

The JPNM prototype uses the following infra-structure:

- The graphical interfaces was implemented using the AdventNet Management Builder version 3.0 for Microsoft Windows 95 [WWW03], compatible with JDK 1.1.6 by SUN Microsystems™ [WWW02];
- We executed the applets under Netscape® Navigator 4.07 for Microsoft Windows 95, Windows NT 4.0 and the applet viewer of AdventNet Management Builder. We use the AdventNet Management Builder code generation facility to implement the applets;
- The servlets was supported by the Java Web Server 1.1.3 by SUN Microsystems™ [WWW02];
- Through the AdventNet Management Builder we can reuse Snmp2, Mibs, Ui and Beans packages from AdventNet to implement the prototype.

Now, we are trying the Weblogic Server 5.10 from BEA as the application server to include the Enterprise JavaBeans in the JPNM architecture. It is possible because the currently Servlet specification 2.2 support the newly introduced Enterprise JavaBeans (EJB) components, described later. WebLogic Server 5.1 supports the following J2EE (Java 2 Platform Enterprise Edition) standards: EJB 1.1, Servlets 2.2, JDBC 2.2, Java RMI (Remote Method Invocation) 1.0 and SNMPv1. On the back end, its support for XML (Extensible Mark-Up Language) will enable disparate data exchange across applications [WWW09].
5. Applied Technologies

According to [9], the principal characteristic of applications based on Web technologies is the use of HTTP servers to make information available over the network.

Since an HTTP server’s function can be summed up as the finding and sending of files to HTTP clients, the creation of dynamic documents is accomplished by other programs that communicate with the HTTP server. In many cases, these programs are part of Web-based applications. These new technologies make the Internet more attractive by making greater interactivity possible between the client and the HTTP server [11].

5.1 Java Applets and Servlets

Applets are Java programs transmitted from a server to a client over the Web and are run remotely by the client’s browser. Applets allow a client to concurrently receive content and the necessary code to interact with the content.

Servlets work as an extension of the server. Actually, they can best be understood as a dynamically loaded Java class that expands the functionality of any server. Servlets are usually used to extend HTTP servers [10].

SUN Microsystems™ has defined a group of basic classes for servlets in the Javax.servlet and Javax.servlet.HTTP packages. The former, for generic protocol-independent applications, while the latter is specifically for the HTTP protocol.
Besides the classes, a servlet engine is necessary. The servlet engine works together with a Web server to guarantee request and response processing, MIME types decoding, and response formatting. We use an autonomous servlet engine since it is included with an HTTP server. In our case, the Java Web Server [WWW02].

The Java class **HTTPServlet** exchanges the HTTP GET and POST protocol operations with the user-implemented methods **doGet()** and **doPost()**. When a request arrives for a servlet it runs the **service()** method which calls **doGet()** or **doPost()** (Figure 2), depending on the desired method.

For example, when the browser invokes a servlet by specifying the name of the servlet in a Uniform Resource Locator (URL), this will cause the **doGet()** method of the servlet to be executed. This practice is suitable for commands that don’t require any user input to be passed to the servlet. However, when the browser user enters form data from a Hypertext Markup Language (HTML) file which in turn invokes the servlet, this causes the **doPost()** method to be executed [12].

### 5.2 Enterprise JavaBeans

Enterprise JavaBeans technology defines a model for the development and deployment of reusable Java Server components (see Figure 3).
Components are pre-developed pieces of application code that can be assembled into working application systems. Java technology currently has a component model called JavaBeans. The EJB architecture logically extends the JavaBeans component model to support server components that run in an application server. EJB technology supports application development based on a multitier, distributed object architecture in which most of an application’s logic is moved from the client to the server. The application logic is partitioned into one or more business objects that are deployed in an application server [16].

We, particularly, want to use the EJB to replace the Management Server (see Fig. 1). In this new configuration, the EJB access the Database Server via JDBC.

5.3 The Server PUSH

With a Server PUSH, an HTTP server is able to send, or push, a sequence of responses to an HTTP client over the same connection. This mechanism, applied in the implementation of the PUSH model, allows a Java servlet together with an HTTP server to return several pages whether asynchronous or not.
The server PUSH connection between the client and the server stays open until the last page has been sent. That way, the HTTP server can send several pages, update them quickly and precisely control the sending of the information [10].

The multipart/mixed is a standard MIME type used in HTTP to encapsulate the server’s responses to a request. On a Server PUSH we use a variant of the multipart/mixed called multipart/x-mixed-replace, where the x indicates that this type is experimental. The replace indicates that each block of new data overwrites the previous block, in other words, the newer data is viewed in place of the older rather than being appended to it [WWW01].

In the multipart/x-mixed-replace type, messages are composed using a separator (boundary) to delimit each block’s data. A message of multipart/x-mixed-replace type does not have a defined end. In other words, the server can keep the connection open and send the data it wants [WWW01].

We use the MultipartResponse class \(^1\) to deal with the details involved in using Server PUSH:

```java
import java.io.*;
import javax.servlet.http.*;
import javax.servlet.лож;

public class MultipartResponse {
    ServletResponse res;
    ServletOutputStream out;
    boolean endedLastResponse = true;

    public MultipartResponse( ServletResponse response ) throws IOException {
```

\(^1\) The MultipartResponse class is part of the Java Servlet API. It provides methods for handling multipart requests, including Server PUSH.
res = response;
out = res.getOutputStream();

res.setContentType( "multipart/x-mixed-replace;boundary=End" );
out.println();
out.println( "--End" );
}

public void startResponse( String contentType ) throws IOException {
// End the last response if necessary
if ( !endedLastResponse )
   endResponse();

// Start the next one
out.println( "Content-Type: " + contentType );
out.println();
endedLastResponse = false;
}

public void sendResponse( String data ) throws IOException {
   out.println( data );
}

public void endResponse() throws IOException {
// End the last response and flush the content, so the client can see it
out.println();
out.println( "--End" );
out.flush();
endedLastResponse = true;
}
public void finish() throws IOException {
    // sends a code telling the client there will be no more responses
    out.println( "--End--" );
    out.flush();
}

6. Software Implementation

Following, we presented some of the tools used in this work.

6.1 AdventNet™ Management Builder

Is a development environment from AdventNet™ based on JavaBeans for building network management applications. The program itself is totally written in the Java language. It includes a graphic development environment, an embedded Web server, an SNMP Applet Server (SAS) to facilitate the installation of applets and a text editor that allows the creation of applets, applications, panels and frames [WWW03]. The most important packages are:

- **SNMP2**: classes that implement SNMP communication and MIB variable types in accordance with Abstract Syntax Notation One (ASN.1).
- **MIBS**: classes that allow MIB manipulation.

For the Application Interface (API) implementation (for example: Figure 4) we used AdventNet™ Management Builder, compatible with Sun’s Java Development Kit (JDK) [sun], using Microsoft Windows 95.
6.2 Java Web Server

The Java Web Server, besides working as an HTTP server, includes all the necessary classes for the development and running of servlets. For this reason, it is classified as an autonomous servlet engine.

1. We reuse the `MultipartResponse` class from [10], page 193, to implement the Server PUSH. It is also available in http://www.servlets.com.
Because we use the HTTP/1.1 persistent connections to implement a PUSH communication, the control of the life-time connections is essential. The Java Web Server provides the persistent connection time out configuration using its administration tool.

6.3 Java Plug-In

Java Plug-in is a software product from SUN Microsystems™ that allows enterprise web developers to direct Java applets and JavaBeans™ components on their intranet web pages to run using Sun’s Java Runtime Environment (JRE).

The Java Plug-In enables the development and deployment of Java™ applets on Internet Explorer and Netscape Navigator browsers and assures that they will run reliably and consistently in both browsers. See more in [WWW02].

6.4 BEA Weblogic Server

The BEA WebLogic Server is an application server, a runtime environment that provides infrastructure services such as database access, transaction coordination, and a component framework for distributed applications. WebLogic Server also provides administrative features such as configurable security, management and application deployment tools, and clustering to promote high availability and scalability.

WebLogic Server operates at the center of a multitier architecture. Clients can be very lightweight, greatly simplifying application deployment. Complete enterprise applications can be built with nothing more than WebLogic Server and a web browser on the client [WWW09].
7. Conclusions

The main objective of this work was the introduction and the analysis of the JAMAP and the JPNM systems for the Web-base network management, which make use of the PUSH technology in the transmission of management information and the sending of asynchronous traps from the agents to the management server and remote station.

We would like to point out some interesting aspects in the application of the PUSH model in network management via the Web:

- The PUSH model reduces network management traffic. Through the subscription phase, the manager does not repeat requests to the agents in each polling cycle.

- A part of the processing is transferred from the manager to the agent. Since the agent himself sends the data to the manager at each PUSH cycle, an increase in the scalability of management systems occurs.

Some problems with a PUSH technology:

- It has not yet been standardized. So, some web browsers don’t support the PUSH Server mechanism. As we present before, we use Netscape® Navigator 4.07 [WWW01]. The testes under Microsoft Internet Explore was not favorable.

- PUSH technology is more suitable in Intranet management.
The Java language, besides guaranteeing better portability and flexibility to the management system, facilitates the distribution of new versions of MIBs, applets, and management servlets. The database for management information is less dependent on the application, as happens now in SNMP management. In this Java context, we can introduce the use of EJB facilities like security, naming, database transactions.

Suggested future work:

- **Extensible Mark-up Language (XML)**

  Another possibility is the exploration of the XML language for representing the management information contained in the MIB. There is a trend to adopt XML, since it was adopted as a standard by Web-Based Enterprise Management (WBEM), an initiative of several companies such as BMC Software Inc., Cisco Systems Inc., Compaq Computer Corporation and Microsoft Corporation to propose a standard model for management based on Web technologies [13].

- **Remote Network Monitoring (RMON)**

  The major purpose of adding RMON capabilities to network management is to increase network efficiency, monitor performance and proactively manage the network. Without RMON a network administrator that was interested in network performance would have
difficult constructing a profile of the network activity using standard tools for the internetwork as a whole [9].

The integration of RMON with the Web-Based management is an interesting idea.

8. References


Akemi Adachi received the M.S. in Electrical Engineering from the Federal University of Ceará (UFC), Brazil. She works as system analyst at Federal Service of Data Processing (SERPRO). Correspondence should be directed to: Akemi Adachi, Federal Service of Data Processing (SERPRO), 832, Pontes Vieira Avenue, São João do Tauape, CEP 60130-240, Fortaleza – CE, Brazil. Phone: +55 85 216 28 91 Fax: +55 85 227 08 05 E-mail: adachi@fla.serpro.gov.br

Marcílio Fernandes de Mendonça currently heads the Java development team at Radix, http://www.radix.com.br, Pernambuco, Brazil. He earned M.S. in Computer Science from the Pontific Catholic University of Rio de Janeiro (PUC – Rio), Brazil. From 1997 to 1999, he was Java instructor at Federal University of Ceará, Brazil. E-mail: marcilio@radix.com.br

José Neuman de Souza is an Associate Professor in the Department of Computer Science at the Federal University of Ceará, Brazil. He is the brazilian representative at the IFIP TC6 (communication Systems) and vice-executive-director of the Computer Network National Laboratory in Brazil. His research interests include but are not limited to network management, quality of service, high speed networks to support multimedia applications and their management. He holds a Ph.D. degree in computer science at the Paris VI University. E-mail: neuman@ufc.br