Web-based Messaging Management Using Java Servlets

G. Jones, E. Zeisler, L. Chen
The MITRE Corporation
1820 Dolley Madison Boulevard
McLean, VA 22102-3481
gbjones, ezeisler, lichen@mitre.org

Abstract

This paper explains the function and design of a prototype messaging management station based upon leading-edge Java servlet technologies and the world-wide web. Web technologies bring valuable cost improvements, flexibility, and security enhancements to the fault management and performance management of electronic messaging systems. Web technologies can also be used to provide an integrated management function. This paper describes MITRE alpha-release software to deploy in an operational environment, and is not an experiences paper as of yet.

Keywords

Distributed Systems, Applications and Messaging Management**,** Java, SNMP

1. Introduction

This paper explains the function and design of a lightweight messaging management station based upon leading-edge Java servlet technologies and the world-wide web.

In the past, the service management of electronic messaging systems has been accomplished either manually, using dedicated mini-computer "platforms", or through purely proprietary means. Today service management must acknowledge current, state-of-the art web technologies such as Internet web browsers, web servers, and the Java programming environment.

Web technologies bring valuable enhancements to messaging management: not only do these technologies provide security solutions, but they also come at little or no up-front cost, and provide greater flexibility in implementing specialized functions. Since secure, inexpensive, extensible web-based solutions are available today, the reasons for web-based management are more than justifiable:

Cost: Free-use Java-based software libraries provide management-specific support, including the Internet Simple Network Management Protocol (SNMP), topological map display, performance management, and fault management.

Security: Public-key security mechanisms can be incorporated directly into management applications, providing access control, confidentiality, and peer authentication. In the case where web protocols are trusted, it becomes possible to issue management operations across security perimeters called "firewalls".

Flexibility: Software development environments and APIs are readily adaptable to suit custom requirements.

Evolution: There is a general industry migration trend towards secure, web-based management. Web browsers are ubiquitous and have become a common user interface to both the Internet and to management information; the tools for developing web-based applications have likewise become abundant and inexpensive. Management applications can evolve in concert with web-based management solutions developed by individual messaging component vendors.

Performance: Information transfer over an unreliable network using web protocols is superior in performance and reliability to the transfer of that information using the SNMP protocol. Thus, connectivity between management domains can be improved.

2. Function

What does it mean to manage a messaging system? Exactly which management requirements are met by the web-based management system? The following needs are basic to managing a messaging system: (1) the percentage availability of the messaging service over a unit of time, (2) an indication of whether the messaging service is operating, and the ability to alarm when it is not, (3) the volume of queued message data held by the messaging system, (4) the delivery status and end-to-end delivery time for a specific message, and (5) the total message throughput per unit time of the messaging system

Now we must explain where and how these requirements are fulfilled within the larger system architecture. In Figure 1, users employ user agents (UAs) to compose

electronic messages that are transferred to their destinations by a network of Message Transfer Agents (MTAs). Therefore, management of the MTAs is critical in meeting the above list of requirements as the MTAs collectively implement the messaging service. A human system manager uses a commercial web browser and a management-enabled web server to access management information at the MTAs.

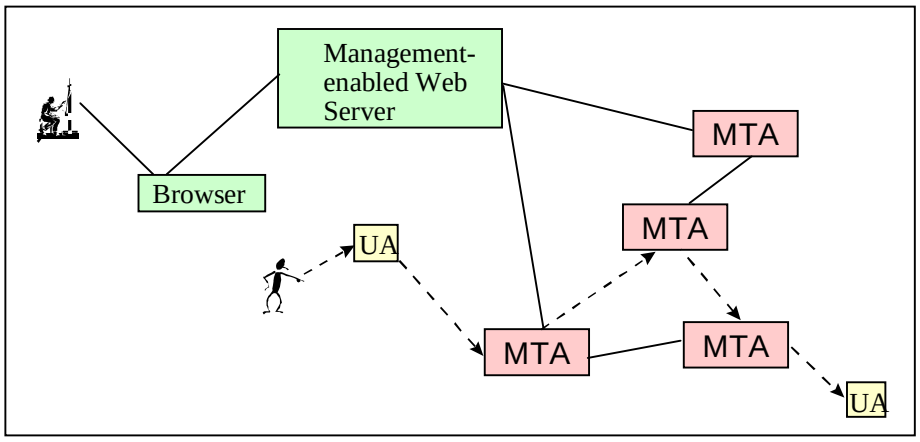

Figure 1. MTAs, UAs, and Managers

In Figure 2, management operations are exchanged using the Internet Hypertext Transfer protocol (HTTP). Web server hosted *instrumentation* in Java provides access to management applications or *agents* at the MTAs using protocols such as SNMP or the Internet Transmission Control Protocol (TCP). The agents execute management-specific operations and return results to the instrumentation at the server. In the case of SNMP, management information at the MTAs is stored in standard Management Information Bases (MIBs) according to Internet Request for Comments (RFCs) 1565 and 1566.

The Java instrumentation at the server implements the following four groups of functions. MITRE selected varying functions that would illustrate Java's flexibility in addressing messaging service management requirements:

Component Status Report. The component status report displays the current outage status of each messaging component or MTA. Components that are ‘up’ or functioning properly are highlighted green on the browser display. Components that are ‘down’ or not functioning are highlighted red and blink in an ‘alarm’ fashion. When a component is down, the report distinguishes between a network (or platform) failure and a component failure (the inability to establish network connectivity to a platform is different from the inability to contact an application running on that platform).

Performance Statistics. This function displays management data at the MTAs in human-readable fashion: standard SNMP MIBs contain current queue sizes in bytes, and contain the amount of message traffic for both inbound and outbound directions.

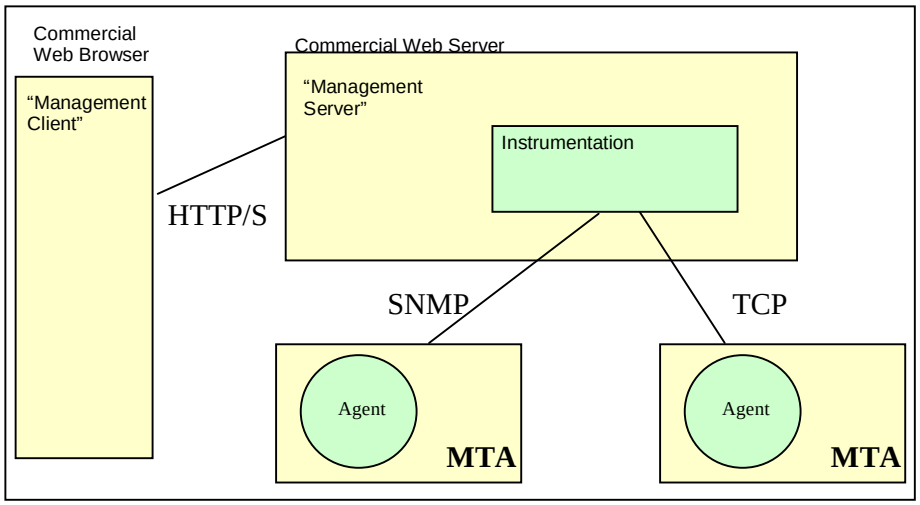

Figure 2. Management of MTAs Using the Web

Outage History Report. The outage history report displays the percentage of time that messaging components were unavailable, due to network connectivity failures or application failures. There are two types of displays: a histogram and an outage record. The outage record displays the time and length of each outage up to the most recent 50 outage events. The histogram displays the cumulative percentage down-time in bar graph fashion for each application on a scale of zero to 100 percent.

Message Tracking Query and Response. Message tracking provides the ability to inquire about the status of individual messages and their attributes. The status of the message might be 'delivered' or 'rejected' and the attributes might include 'originator', 'recipient' or 'message identifier'. This function accepts the name of a target MTA along with some of the previous attributes and returns a trace report of that message for one MTA that includes the status of the message and the name of the next MTA to take responsibility for the message, if any. The selection criteria include any one of message identifier, originator, recipient, time range, or combination of these. When multiple criteria are specified, the 'logical AND' of the criteria is used to calculate the result. If the message was transferred onward to another MTA, the user would issue a subsequent query against the next MTA.

3. Design

Let's step back from the detail for a minute and look at the goal, scope, and related project work, which provide context for our design.

3.1 Goals

The goal of a WEB browser that can extract network and application performance statistics is primarily to validate intelligent management of WEB-centric

environments for DMS with a practical, field-able solution. The server building blocks that contribute to the JAVA COTS solution work together to provide the messaging management service itself so that e-mail as a distributed service can be monitored for the state of the service.

Reliance on highly dynamic ("movable objects") technologies. Interoperability is being driven by the growth of WEB-centric components that can be understood as packages of data and software that are no longer required to reside on any one node of a network, but rather, can be moved around as needed to help optimize network and systems performance. This flexible model of application objects, as entities that can be "executed" at the locations where they are most needed, affects interoperability by changing the scope of what is being exchanged between messaging platforms. Technology examples are seen in Java, which can move entire objects around the network as byte code.

Impacts for Scalability. As DMS expands it's subscriber community to encompass external interfaces, a highly dynamic solution will enable growth, especially because the solution enables measurement of service levels for multi-party agreements.

3.2 Scope and Related Work

The work for a mail message transfer agent Web Browser can be extended for directory (directory service agents) as well as for access management for DoD Public Key Interchange (PKI) (see Figure 3). Although these distributed services are becoming widely available in the DOD community, management of these services end-to-end has just begun to be addressed. Furthermore, external domains will benefit from any future interoperability agreements to use the Defense Message System for either mail or directory interfaces, like the DMS NATO allies, ad-hoc coalitions for non-NATO countries, Combined Communications Electronics Board (CCEB), treaty-based alliances with other countries, inter-agency in CONUS. Other projects like electronic data interchange are subjects for far-term (3 to 5 years) integration with DMS.

User service links, as distinguished from middleware processes that are served by the JAVA servlet and protocol translation software, must distinguish roles as an extension of the concept prototype and filter the summarized performance statistics for various consumers, such as a DMS database administrator, DMS directory systems administrator, DOD or NATO security administrators, DMS and non-DMS network operators and other external operations support systems. A Meta Model is being developed which can address mapping to end-users by role; a significant portion of our solution is identified as MIB meta-data (super-class structures).

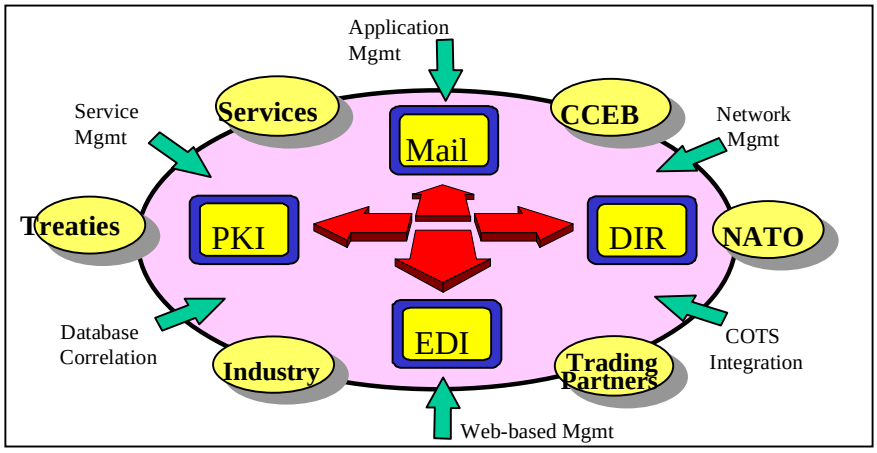

Figure 3. Design Context

Perhaps more importantly, difficulties arise with the potential deadlock between threads for syntactic reasons if the system is designed without an ability to detect deadlock at runtime. In general, large and distributed organizations, such as the DMS virtual enterprise, typically have system environments with projects where work must be managed so that people can do their work in parallel, share resources and collaborate. A standards-based state model for the status checker is one means to control concurrent access among multiple users competing for the same resources.

3.3 Technical Approach

The technical approach for a web-based management architecture consists of SNMP, HTTP, a web browser, a web server, the Java Management Application Programming Interface (JMAPI) SNMP Application Programming Interface (API), the Java Server Development Kit (SDK), and the Java Web Server. The instrumentation at the web server is manifested in a Java program called a *servlet*.

In Figure 4, a servlet runs inside the server and implements messaging management functions there. One might think of the browser and the server as Java-based operating systems running management applications that are servlets (servlets often run continuously inside the server and spawn multiple threads of execution). The browser and the server communicate with one another using the HTTP protocol. Using the freely available Java Servlet SDK or "Servlet API" and JMAPI SNMP API, it is possible to communicate with existing SNMP-based and TCP-based management agents: e.g., an SNMP agent might provide key performance and fault monitoring functions and a TCP agent might perform message tracking functions.

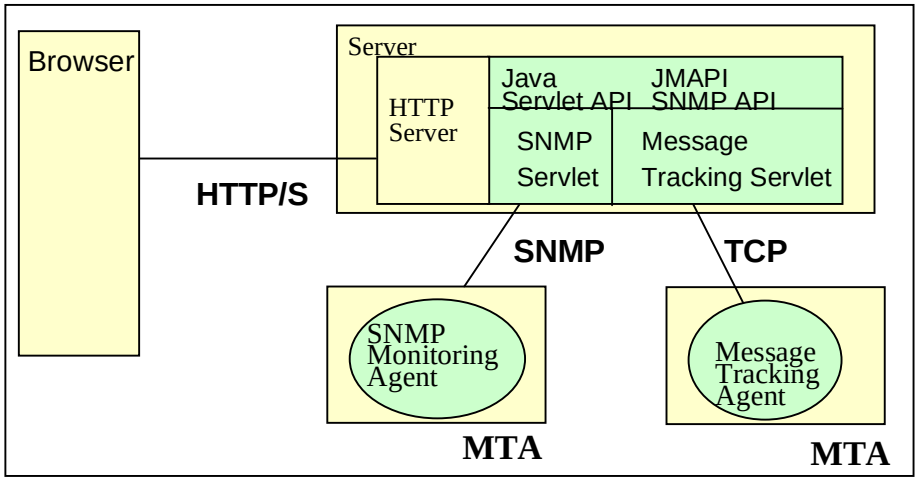

Figure 4. Servlets

The web server utilizes both the SNMP and TCP protocols to communicate with the agents and then provide this information to a web browser using the HTTP protocol. Any customization done for messaging is almost entirely contained within the servlet and HTML files.

The Java class `HttpServlet` exchanges HTTP GET and POST protocol operations with the browser client. Inside class `HttpServlet`, the method `doGet` implements the HTTP GET operation, and the method `doPost` implements the HTTP POST operation. For example, when the browser invokes a servlet by specifying the name of the servlet in a 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.

The following Java code example demonstrates the use of `doGet` to execute an SNMP operation. `HttpServletRequest` objects deliver the HTTP request from the browser to the servlet. The server includes its response within `HttpServletResponse` objects that are returned to the browser for display. The classes `HttpServletRequest` and `HttpServletResponse` provide more generic methods for the extraction and encapsulation of request and response data in ASCII or binary form.

```
public class StatusServlet extends HttpServlet {
…    public void doGet (HttpServletRequest req,
                        HttpServletResponse res)
      throws ServletException, IOException {
      // load the messaging specific SNMP MIB variables
      String mibSupplement [] [] = {
         (“applName”, “.1.3.6.1.27.1.1.2”, “S”),
```

```
      ("applOperStatus", ".1.3.6.1.27.1.1.6", "I") };
   MibStore.loadMib(mibSupplement);
   SnmpVar applName = new SnmpVar("applName");
   SnmpVar applOperStatus = new
      SnmpVar("applOperStatus");
   varBindList.addVariable(applName);
   varBindList.addVariable(applOperStatus); …
   // issue the SNMP get operation to the agent
   SnmpRequest aRequest =
      session.snmpGet(agentInfo, varBindList);}}
```

The following code fragment demonstrates the use of `doPost` to execute an SNMP operation only after the user inputs the IP address of the platform to be queried from an HTML form:

```
public void doPost(HttpServletRequest req,
                   HttpServletResponse res)
                   throws ServletException, IOException
   // parse the HTTP request into a set of fields
   table = parseMulti(boundary,
req.getInputStream());…
   // look at each field type in the HTTP request
   String IPAddr = null;
   for (Enumeration fields = table.keys();
        fields.hasMoreElements(); ) {
         String name = (String)fields.nextElement();
         Object obj = table.get(name) ; {
         // look at the value of the field
         String[] values = (String[]) obj;
         for (int i =0; i < values.length && i < 1;
   i++) …
   // we are looking for a value of "IPAddr"
   // as this contains an IP adddress
               if (name.compareTo("IP Addr") == 0){
   // save the value of the IP address
                     if (values[i].length() > 0)
                           IPAddr = values[i];}}}
   // now connect to that IP Address and get the
   // RFC1566 information as in the previous code
example
   String mtas[][] = new String[MAXMTAS][MAXATTRS];
   int numMtas = getMtaTable(IPAddr, "161", mtas);}
```

The following sections describe servlets that implement specific messaging management functions.

Browser and Status Servlets. The browser and status servlets both access standard SNMP MIBs for messaging. The status servlet displays both the current outage status

(up or down) of messaging components obtained using the SNMP variable `applOperStatus` from RFC 1565. Components that are down blink in an "alarm" fashion. If the SNMP agent is inaccessible, the `StatusServlet` distinguishes between this case and the case where the agent is accessible but the application is down.

The MIB browser servlets format and display SNMP information to the management user in human-readable fashion. These are SNMP-intensive and implement the major tables in RFCs 1565 and 1566. These servlets will provide on-demand polling and presentation of SNMP variables specific to messaging and directory as a precursor to a reporting capability. For example, the SNMP variable `mtaVolumeStored` contains the MTA's queue sizes.

In Figure 5, the Java classes `StatusServlet` and `MtaInfoServlet` inherit from class `HttpServlet`. The `doGet` method of the `StatusServlet` receives an HTTP GET command which instructs the servlet to obtain the operational status of all applications from the relevant SNMP agents. `doGet` returns HTTP-encapsulated HTML data to the browser user.

The `doPost` method of the `MtaInfoServlet` receives the name of a host from the browser user. `doPost` then queries the agent at that host to obtain MTA-specific information from the MIBs such as the MTA's queue sizes and recent message throughput. `doPost` returns HTTP-encapsulated HTML data to the browser user.

Outage History Servlet. The outage history servlet displays the cumulative outage status (up or down) of messaging components as obtained from SNMP MIBs. This includes percentage down-time histograms for each process and summary of recent state changes with the date and time of the last change. The outage history servlet uses the variable `applOperStatus` to obtain the component’s status, but unlike the status servlet it does not distinguish between the inability to contact the SNMP agent and the component malfunctioning, since both events constitute an inability to contact the component.

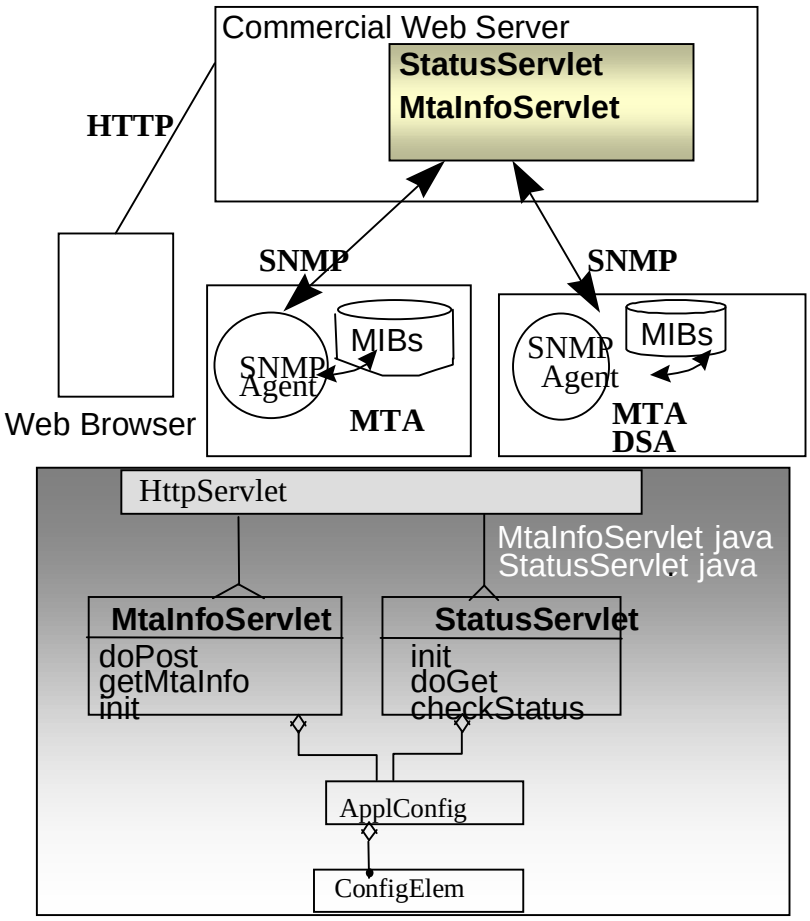

Figure 5. Browser Servlet, Status Servlet and Corresponding Booch Diagram

In Figure 6, the Java class `OutageReportServlet` inherits from class `HttpServlet`. A background process is implemented by class `StatusChecker`, which inherits from class `Thread`. `StatusChecker` periodically records the outage status of individual applications using SNMP, records state changes in a formatted outage log, and maintains an ongoing tally of the percentage uptime for each application. Later, when the `doGet` method receives an HTTP GET command, `doGet` returns both the 50 most recent events and the percentage downtime for each application to the user in HTTP-encapsulated HTML data form to the web browser. OutageReportServlet uses synchronized blocks of code to moderate between `doGet` and the `StatusChecker run` method in case these two methods try to access the outage log simultaneously.

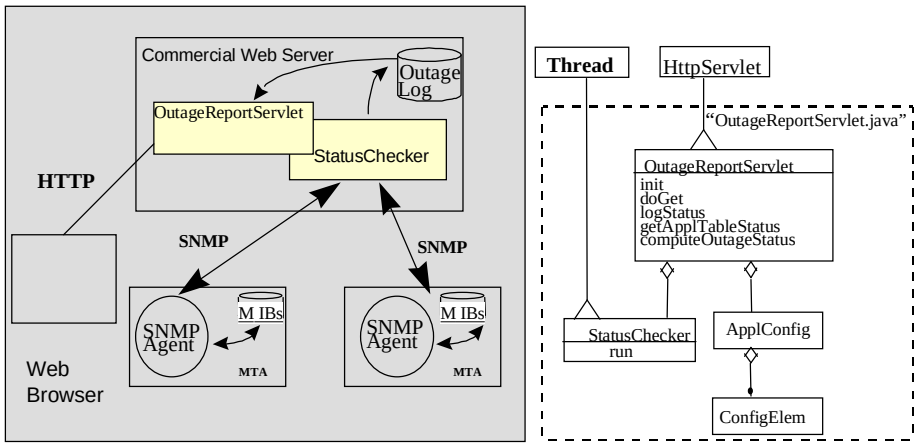

Figure 6. Outage Report Servlet and Corresponding Booch Diagram

Message Tracking Servlet. The message tracking servlet provides a user interface to TCP-based message tracking. This servlet accepts query criteria as input (message identifier, originator, intended recipient, time range, or combination of these) and the name of a target MTA. The output returned to the browser user includes the message delivery status, time of delivery, and next-hop MTA in addition to the message ID, originator, and recipient.

In Figure 7, the Java class `TrackServlet` inherits from class `HttpServlet`. The `doPost` method of `TrackServlet` receives an HTTP POST command, which causes the servlet to execute message tracking requests over TCP to query the TCP-based message tracking agent at the MTA. When the TCP agent responds to `TrackServlet`, the `doPost` method of `TrackServlet` returns HTTP-encapsulated HTML data to the browser user.

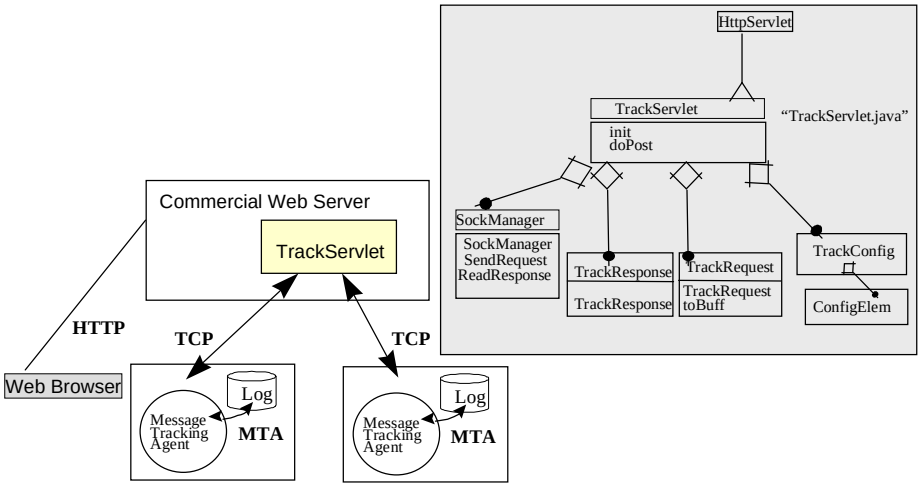

Figure 7. Message Tracking Servlet and Corresponding Booch Diagram

In summary, `TrackServlet`, `OutageReportServlet`, `StatusServlet`, and `MtaInfoServlet` are examples of web-server-based instrumentation providing an integrated management function that includes HTML-based user interfaces, back-end SNMP or TCP protocol solutions, and legacy implementations.

4. Secure Management and Synchronization

Secure functions are provided by the Netscape Certificate Server (CS) and the Netscape Enterprise Server (ES) which implement a public key infrastructure and the Hypertext Transfer Protocol over Secure Socket Layer (HTTP/SSL or simply HTTP/S). Server authentication, client authentication, and public key security features have been implemented in the prototype using X.509-based RSA encryption technologies. When integrated the above capabilities together provide a secure management service. In Figure 1, the web browser and web server provide secure management via HTTP/S; the transmission of management information between the managed components and the commercial web server is unsecured. The connection between the browser and server is secure and encrypted using HTTP/S. This provides an effective “domain management” approach where firewalls protect the domain from outside attacks and HTTP/S provides security between domains.

4.1 Server Authentication

Server authentication occurs when the web server negotiates a key exchange with the web browser. The server creates a cryptographic token that is used to encrypt the HTTP traffic between the server and the browser. This does not require the browser user to provide his or her own credentials. Table 1 shows steps that are required in order to install and configure a Netscape secure management server for server authentication.

Once these steps are complete, the server will encrypt all HTTP traffic by generating an encryption key on a session-by-session basis. As long as the CA is trusted by both the client and server, this guarantees that information received from the server is indeed from that server, has not been modified in transmission, and has not been viewed or manipulated by outside parties.

In addition to the above encryption features, if the ES wishes to authenticate each client on a client-by-client basis, the following additional steps are necessary:

Table 1: Client and Server Authentication

SERVER AUTHENTICATION
1. Identify a Certificate Authority (CA). If certificates are to be generated on-site, this will require installation and configuration of the Netscape CS or equivalent. The configuration of the CS includes defining the CA. 2. On the Netscape ES, define a secure server. Each ES can define multiple web servers using separate ports. 3. On the server platform, generate a symmetric private key and public key for the server. This requires defining a password that will thereafter be required whenever the secure server defined above is administered (e.g., started or stopped). 4. When logged into the ES, request a certificate from the CS. This will automatically cause the ES to exchange its public key with the CS. The CS will notify an arranged recipient later via E-mail. 5. When logged into the CS, approve the certificate request and send the certificate to the electronic mail address specified earlier by the ES. 6. When logged into the ES, extract the certificate from the electronic mail message and import it into the secure server previously defined. 7. Enable encryption at the ES for the specified secure server.
CLIENT AUTHENTICATION
1. A browser user must request a personal certificate from the CS. It is easiest if the same CA that issued the server certificate for server authentication issues the certificate. 2. When the browser user requests a certificate, the browser will generate a public key and private key for the current user who is making the request if no keys had been generated previously for that user. 3. The CS will inform that browser user via electronic mail that the user's key is ready by providing in the message a Uniform Resource Locator (URL) where the key may be found. 4. The browser user connects to the indicated URL to import his or her key into the browser.

From this point on, whenever the user connects to the ES the ES will request a key from the browser for the user signed by the same CA. The browser will either inform the user that a key is required, requesting that the user transmit his or her key, or the key may be transmitted automatically subsequent to the first time it is requested. In this approach, not only is the interaction between the web browser and the web server encrypted, but also the same CA as the server legitimizes each individual user. Using this approach, an organization can issue certificates for those users it deems legitimate, and only those users. It will soon be possible in the MITRE prototype to provide access control on a user by user basis, but this item is left for future study. Access could be used to assign different privileges to different levels of users.

4.2 Node Management

For the purpose of synchronization management, a state may be decomposed into optional compartments for status as shown in Figure 8 state and status notations. At the top of figure, the variable names are declared for passing special types of transition status (queued and delayed) that are internal to the state machine for management of multi-threaded operations. Status is a finer grained sub class of state

that reflects different situations for application and network faults. In the lower part of the figure, an entry and exit state automatically tag the thread: the pseudo event names are entry and exit. A nested state is used for the do operations that are continuous, and are alternately performed as stand-alone sessions or in concert. A nested state allows multiple status conditions to be partitioned, enables control to ensure a thread completes without interference from other threads, and is a lightweight approach for handling complexity. A do operation executes when the JAVA application intercepts an incoming request. Detailed operational, administrative and usage states are identified in ITU-T standards [10165, parts I-IV].

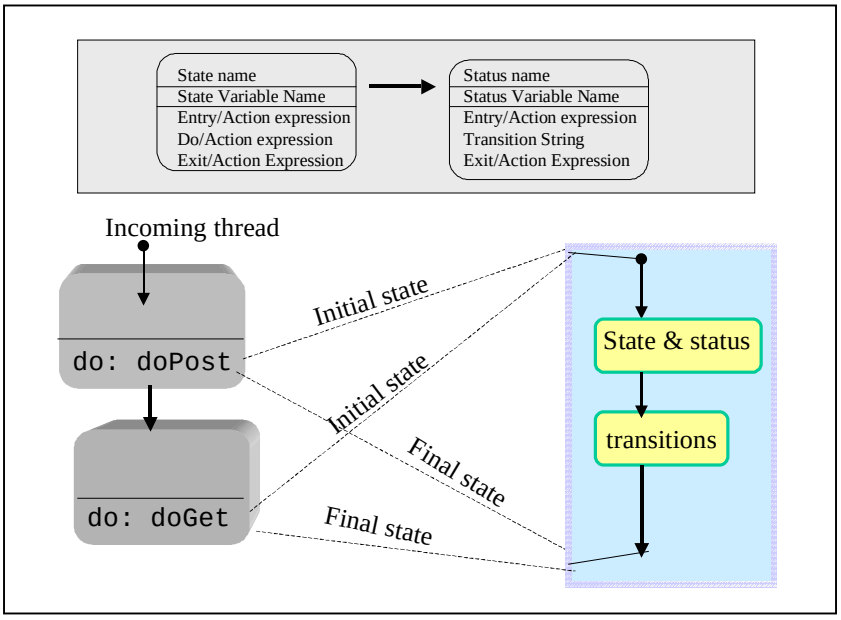

Figure 8. State and Status Modeling Notation

5. Conclusions

Not only are Java servlets an effective approach for managing messaging systems, but Java has proven to be an effective mechanism for providing an integrated management function. Java- and web-based management strategies offer flexibility in addressing custom management requirements, security functions such as data encryption, and noticeable cost advantages.

Future enhancements to the MITRE prototype might employ additional functions such as topological map display, persistent database storage, and a fault notification framework.

Bibliography

Chang, Phil Inje, "*Inside the Java Web Server: An Overview of Java Web Server 1.0, Java Servlets, and the JavaServer Architecture"*, Palo Alto, California: Sun Microsystems Inc., October 1997.

"*Java Management Programmer's Guide"*, Palo Alto, California: Sun Microsystems Inc., May 1997.

T. Berners-Lee & D. Connolly, *"Hypertext Markup Language - 2.0"*, November 1995, RFC 1866.

R. Fielding, J. Gettys, J. Mogul, H. Frystyk, T. Berners-Lee, *"Hypertext Transfer Protocol -- HTTP/1.1"*, January 1997, RFC 2068.

J. Postel, *"Transmission Control Protocol"*, September 1981, RFC 793.

Kille, S., and N. Freed, "*The Mail Monitoring MIB"*, January 1994, RFC 1566.

Case, J., et al., *"Simple Network Management Protocol"*, May 1990, RFC 1157.

Jain, N., et al., "*Messaging Management Implementor's Guide: Monitoring and Message Tracking, Version 1.0"*, Arlington, Virginia: Electronic Messaging Association, May 1995.

Jones, G. *"Managing the Message: Message Tracking"*, IEEE/IFIP 1998 Network Operations and Management Symposium, 1998: The Institute of Electrical and Electronics Engineers, Piscataway, NJ, pp. 743-747.

E. Zeisler, T. Bollinger “A Universal Service Manager Architecture for Distributed Session Management”, Ninth IFIP/IEEE International Workshop on Distributed Systems: Operations & Management, Oct 1998, The MITRE Corporation

C. Cicalese. “A Multi-Protocol Test Bed for Management of Distributed Services”, MITRE Corporation, IFIP/IEEE International Workshop on Distributed Systems, October 1994.

Paul Allen and Stuart Frost, “Component-Based Development for Enterprise Systems”, Cambridge University Press, 1998.