7

Distributed systems management on the web

B. Reed, M. Peercy, E. Robinson
IBM Almaden Research Center
650 Harry Rd
San Jose, CA 95120
breed@almaden.ibm.com

Abstract

The need to manage a multi-platform enterprise and the advancement of web technologies have made it possible to build powerful cross-platform management applications. As more network and system management tools become web-enabled, the web browser becomes a convenient point of integration.

This paper describes the architecture and implementation of a web-based management console to an existing PC management application. Justification for using an embedded web server instead of the normal general purpose web server and CGI's is given. The limitations and solutions are discussed, as well as advantages over traditional GUI's.

Keywords

web-based management, embedded web server, CGI, WWW

1 INTRODUCTION

As the number of computer systems grows in an enterprise domain, administration of all the systems in the enterprise from a single console becomes more and more of a requirement. A number of systems management packages and tools exist to integrate, on a single screen, data from several sources throughout an enterprise network. These facilities have traditionally been designed for mainframe computers and UNIX workstations, both because these machines have controlled the critical resources of the enterprise and because the

A. A. Lazar et al. (eds.), *Integrated Network Management V*

computational requirements of managing such resources demand such machines. In fact, systems management utilities often require a workstation as a dedicated platform for execution. However, as x86-based personal computers and PC servers obtain dominance in the market, the ability to monitor and administer these resources becomes extremely important.

A number of companies have developed products for the administration of PC resources. Among these products are IBM Tivoli's NetFinity, Compaq's Insight Manager, Intel's LanDesk, and Microsoft's SMS. Each of these products addresses parts of the hardware, software, and network domains of the PC computing universe. Nonetheless, prior to this research, none of them integrated all their features well into the enterprise systems management utilities being run on UNIX workstations to manage UNIX and mainframe resources.

Recently there has been a lot of interest in web-based management solutions (Wellens, 1996)(Bruins, 1996)(Mullaney, 1996). Switches and routers (McLean, 1996) are available that can be controlled and configured from a web browser. (Deri, 1996). In addition, Deri (Deri, 1996) has shown a way to access network management resources from the web. In this paper we show how to use the web for more complex systems management operations. By web-enabling systems management tools the web can provide a point of integration for enterprise management.

In this paper we discuss the problems and issues involved and describe our web-based solution. Section 2 describes the problem and our constraints. Section 3 provides background in the NetFinity product and general web HTTP/HTML serving. In Section 4 we discuss the solution, and how it was implemented. Section 5 explores issues surrounding the solution, particularly limitations and unexpected benefits. Section 6 continues with performance measurements of our embedded web server serving systems management data. We conclude with Section 7.

2 THE PROBLEM

In general, UNIX and mainframe systems and network management tools have a centralized management architecture with a heavy footprint requiring one or more powerful machines dedicated to the task of management. Many times the graphical user interfaces (GUI's) used to administer the network also have a large footprint and are intimately tied to the management machine. On the other hand, PC systems management tools such as NetFinity are designed to manage PCs in a peer-to-peer fashion without requiring dedicated management machines.

In order to integrate PC systems management into UNIX-domain systems

management packages, we would like to integrate the PC's GUI's into the UNIX-domain's. However, these PC GUI's impose significant restrictions on scalability, screen real estate, and platform. Not only is direct porting of PC GUI's to the UNIX domain unpleasant because of the cross-platform implementation, but the PC GUI's are principly designed for single-system-at-a-time management. Thus scalability as well as screen real estate become extremely limiting issues. This integration required from us a different approach.

Our goal was to enhance NetFinity so that it would meet the following criteria:

1. the manager and the GUI would retain a small footprint,
2. the GUI would be cross platform, and
3. the GUI would easy to develop, change, and integrate with other tools.

Our solution to this problem was to embed a minimal web server into NetFinity so that any platform with a web browser could manage the NetFinity machine. By minimal we mean only the basic functionality needed to process HTTP requests without the overhead of general purpose file serving and CGI-bin style processing. Thus the GUI is automatically cross-platform, at least for all the platforms that have a capable browser ported to them. Furthermore, the embedded web server is very small, consuming few resources. Finally, since HTML is significantly easier to work with than traditional event-driven GUI code, development, maintenance, and integration is quite pleasant.

A minimal web server was implemented by encapsulating the hypertext transfer protocol (HTTP) in a C++ subclass of iostream. In this paper, we describe the processes which brought us to this design, describe the design, and offer some performance numbers of the embedded server delivering real-time systems management data.

3 BACKGROUND

3.1 Introduction to NetFinity

NetFinity is a completely distributed management tool for PC's. NetFinity agents are installed on every managed PC, and managing machines do not need to be dedicated to NetFinity management. The NetFinity machines communicate over a remote procedure call (RPC) layer that sits on top of NETBIOS, TCP/IP, IPX, or serial protocols. The RPC layer allows multiple hops through multiple protocols to be used to contact a remote host. Among the services supported are screen captures, process management, remote sessions, alert management, and system monitoring. The base executables that perform these services are small; in some cases they are loaded and unloaded on demand, and therefore do not require

additional system resources. The user interfaces communicate with the bases over the RPC layer.

3.2 Introduction to the web

The World Wide Web consists of two main standards: hypertext transfer protocol (HTTP) (Berners-Lee, 1996) and hypertext markup language (HTML). HTTP is a stateless protocol used to transfer files to web browsers: a distinct HTTP request is made for each file requested. HTML is the usual file type that is transported by HTTP and is a markup language that describes the format of the document to the web browser. Along with simple document rendering, HTML also supports clickable links to other documents and conveniently arrayed forms in web pages. The standard order of events in forms processing is as follows:

1. The web browser renders the page. Included in the page are form elements allowing user entry of commands or data.
2. The user makes the entries he or she wishes and submits the form.
3. The web browser generates a uniform resource locator (URL) consisting of the destination of the form submission and a number of variable/value pairs representing the entered data.
4. The web server receives the request and services it according to the page and arguments in the request.

Furthermore, variable/value pairs may be hidden in the rendered document and thus used to maintain state between requests. Also, since all the documents we are generating change often, we send the documents to the web browser as "expired" to prevent them from being cached.

For security we used a publicly available Secure Socket Layer (SSL) (Freier, 1996) library to implement our embedded web server. Most popular web browsers support SLL. SSL uses asymmetric keys to negotiate a symmetric key that is used to serve the HTTP request. For performance the symmetric key is cached for a short period of time. SSL provides encrypted communication as well as authenticating the server to the browser using an X.509.v3 certificate. Clients are authenticated using the NetFinity authentication RPC and basic web authentication.

4 PUTTING NETFINITY ON THE WEB

Because of the simplicity of HTTP, it is possible to create a minimal HTTP server with a very small footprint. We encapsulate the HTTP protocol in a class that inherits from iostream and includes methods to retrieve variables passed, the document requested, and authentication information. By inheriting from

iostream, all of iostream's formatting and buffering are available to the server application code.

The HTTP class consists of the following:

1. methods for replying through the socket to the HTTP requestor,
2. methods for parsing the services and arguments present in the URL,
3. methods to ease formatting of URL data into the HTML forms, and
4. all the methods inherited from iostream.

The web systems management interface application listens on a socket for the HTTP request, in the form of a universal resource locator (URL), from the web browser. When it receives a request, the server spawns a thread with the URL and the socket instantiated in the HTTP class. The thread parses the request, using HTTP class methods, to determine the systems management service and the arguments desired. The thread then executes the RPC communication appropriate to perform the requested systems management service. The RPC's receive information from the backends, and the results are formatted dynamically and returned to the browser, again using the HTTP class. To avoid conflict with other web servers which may be running on the system, our server runs on a user-selected port whose default is different from the traditional port 80.

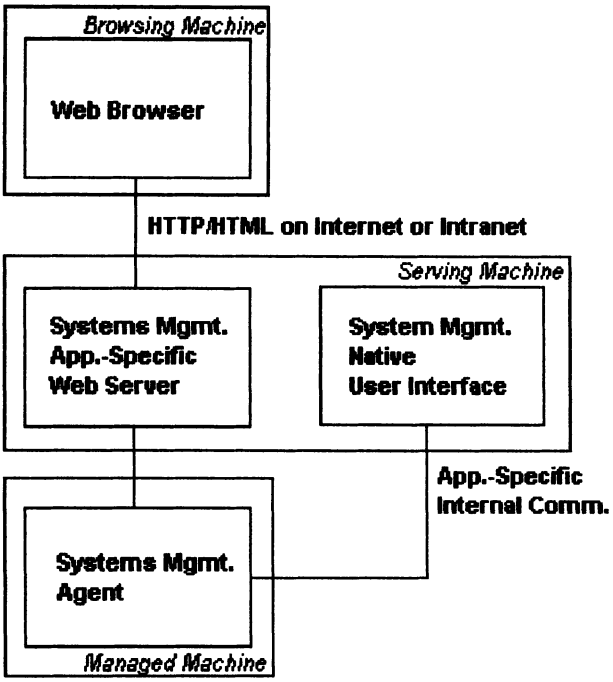

Figure 1 Architecture of web interface daemon.

The result is a daemon that communicates with a web browser as if it were a web server and communicates with NetFinity backends as if it were a NetFinity GUI. Figure 1 shows the general architecture. Through this server architecture any web browser on any platform can manage a NetFinity machine. Implementing the daemon is very easy since it is totally procedural: many of the normal GUI event driven difficulties are avoided.

5 ANALYSIS

5.1 Limitations

Of course, there are tradeoffs to using the web as a management interface. Limitations include the use of forms as a means of interaction, the lack of client form validation, and basic formatting restrictions.

Being limited to forms as the only means of user interaction turned out not to be as limiting as one might imagine. In many cases, this restriction made the user interface less cluttered and easier to use, and, in general, it is a good fit for our systems management applications. While we did not have full control of screen

layout, extensive use of tables (Raggett, 1996) improved the appearance greatly. In the end the limited control of page layout simplified the pages and allowed them to look good on a variety of platforms.

The one service that we could not do with forms was Remote Session. Remote Session offers the user a shell on the remote machine. Therefore, it must be fully interactive. We used Java to achieve this level of interactivity. Java is a very powerful yet simple language that enabled us to implement Remote Session in a few hundred lines of code. When the session is started, it makes an HTTP request to the daemon, and once the connection is made, the daemon and the Java applet use that connection as a full duplex channel.

The lack of form validation was not functionally limiting, but it does make forms somewhat more difficult for the user since he or she learns of errors in the form only after submitting it. Javascript from Netscape has the potential of fixing this. However, the lack of support, as well as security issues, prevented our use of it.

5.2 Unanticipated benefits

A number of unanticipated benefits arose from the using the web as the GUI. These include the ability to mail web pages as an alert-triggered action, the bookmark service offered by the web browser, links to other web resources, the find button of the browser, and rapid turn around of GUI modifications.

Various services in NetFinity and on NetFinity-managed machines generate alerts. These alerts can trigger pagers, e-mail, window pop-ups, etc. We constructed an alert action that sends a MIME-encoded e-mail of type "text/html" with links back to the machines involved in generating the alert. Thus, if the user's mail reader is properly configured, his or her web browser loads the page that was sent in the e-mail. E-mail is a convenient form of notification since people in general either watch, or have applications which watch, for new mail in their mail spools. Furthermore, since the e-mail is actually a web page, the links in the message allow the user to quickly access the machines causing and reporting the problem.

One of the difficulties in using any application with multiple GUI windows is finding a particular window on the screen or from one use to another. A native GUI's traditional means of changing context is by opening a new modal dialog box. In contrast, a web browser works in one window unless the user chooses to open another one. Also, when using a web browser, a bookmark can be set at any page and the user is able to return to that page at any time. In addition, instead of explaining complicated navigating instructions to another user who needs to find the window, a URL can be given or the page can be mailed. Also, pages can be printed straight from the browser. The find button is particularly useful on

information packed pages. All these features come free to the web application developer but would be very costly to implement in native GUI's.

In summary, using a web browser as the front end and implementing a web server layer to translate service activity from the web domain to the application domain offers a plethora of benefits. The browser performs ancillary tasks, as a matter of course, that are only dreamed about in traditional GUI's. Traditional GUI's do not necessarily miss these resources, but they can be leveraged to provide a valuable service to the application user. In the case described in this paper, the user is the system administrator whose job becomes significantly easier with the portable and powerful web browser/server interface.

The use of HTML made GUI prototypes as easy as editing a web page. Changes to program code that did page layout were also simplified. Through the use of rapid prototyping and the ability to make quick changes to the code, the GUI evolved more rapidly. This often resulted in a better user interface than the native GUI.

6 PERFORMANCE

Today's web servers traditionally process forms through the use of CGI (Robinson, 1996) binaries or scripts. These are spawned by the HTTP server in response to an HTTP request to a specifically patterned URL. CGI bins are often written in C or C++. Also, Perl is a favorite language for these scripts.

Our embedded HTTP server does not spawn its form processing requests to a CGI program. We handle the form processing with the same executable as the server itself. In fact, the server connection, an http object as described previously, is passed to whatever handler is indicated by the requested URL. Each request spawns a new thread, and therefore operation is extremely lightweight compared to traditional CGI.

We compared the operation of the server implemented through the http class with a server implemented through spawning. License agreements for HTTP servers generally do not permit benchmarking the servers. This fact, plus the wish to make the CGI-spawning HTTP server as comparable as possible to our threaded server, led us to write our own spawning server. The spawning server simply accepts an HTTP connection, spawns a program, and directs the output of the program back through the connection. Since this is the absolute minimum that any HTTP server must do, it provides a good baseline.

The servers were run on a 100 MHz Pentium with 32 Meg of RAM running Windows NT 3.51. The HTTP requests were generated by a UNIX machine on

the same LAN. Processor usage was gathered using the Performance Monitor built into Windows NT. SSL was not enabled. NetFinity RPC requests were made to the local machine. The page being served is the main page that lists the available services.

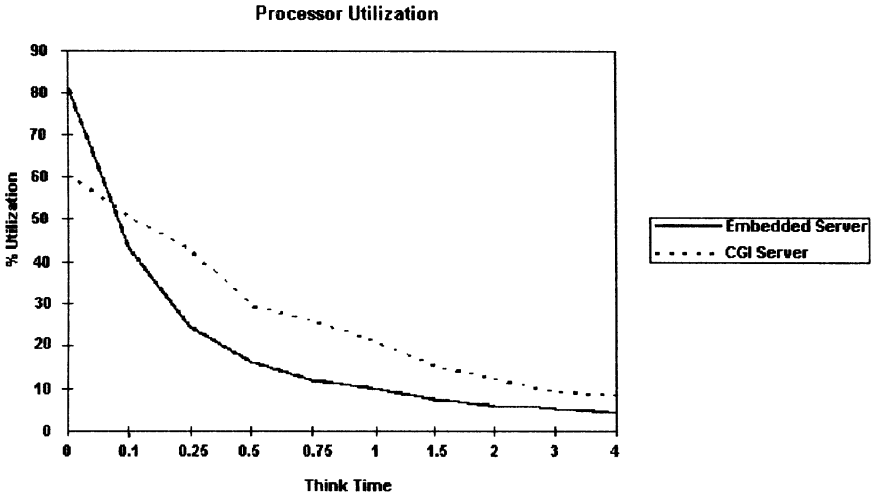

Figure 2 Processor utilization comparison of an embedded server verses a CGI server.

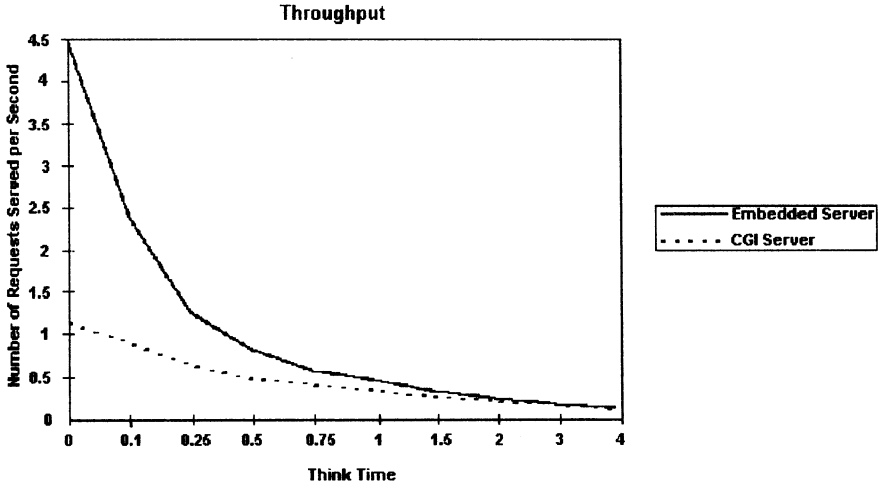

Figure 3 Request throughput comparison of an embedded server verses a CGI server.

Figure 2 shows the processor utilization in relation to the think time for the two servers. Think time is the time from the end of the previous request to the start of the next request. It does not include the time to service the request. Figure 3 shows the throughput in relation to the think time. As think time increases, think time dominates the service time, and the throughput will tend toward the reciprocal of think time.

Note that even with a think time of 4 seconds the processor utilization of the CGI server is double that of the embedded server with approximately the same throughput. The contrast is even bigger with a think time of 250 milliseconds. The embedded server uses a little more than half of the processor utilization with twice the throughput. Only near zero think time does the utilization of the embedded server pass that of the CGI server; however, the throughput is nearly quadruple that of the CGI server. This difference is simply due to the overhead of spawning the GCI process.

In normal operations the time between HTTP requests to a server will be much greater than 4 seconds since generally the page will have information that will take time to be read by the user. However, as explained above, one machine can be used to administer other machines. This is especially useful to be able to manage non-IP machines (such as IPX and NETBIOS) from the Web. If a CGI server were used, a dedicated machine would be needed if the throughput were to get above 1 request per second. The embedded server uses less than 25% of the CPU at 1 request per second eliminating the need for a separate dedicated server for all but the busiest of managers.

It should be noted that the overhead of spawning is not the only performance advantage of using an embedded server: application initialization can also be very costly. In the case of the NetFinity RPC layer, communications must be set up and taken down each time a program is run. Using an embedded server, the initialization cost is paid only once; however, using CGI's requires the initialization to be run for each request.

7 CONCLUSIONS

Platform-independent user interfaces offer powerful flexibility in administration and integration of similar or disparate systems management tools. Web-based systems management provides a large segment of the overall picture which will bring single-image systems management to reality. Furthermore, because the HTML interface is as simple at the server end as it appears at the browser end, advancement in systems management features can proceed at a much greater pace than is possible with traditional GUI's. Aided by Java for functionality

requiring a fully interactive interface, web-based systems management is definitely a step towards more powerful systems management.

6 REFERENCES

Berners-Lee, T., Fielding, R., and Nielsen, H., "Hypertext Transfer Protocol -- HTTP/1.0", RFC 1985, May 17, 1996

Bruins, B., "Some Experiences with Emerging Management Technologies", Simple Times, July, 1996

L. Deri, Surfin' Network Management Resources Across the Web, Proceedings sf 2nd Intl IEEE Workshop on Systems and Network Management, Toronto, June, 1996

Freier, A., Karlton, P., and Kocher, P., "The SSL Protocol, Version 3.0" (Internet Draft), March 1996

McLean, M., "Browsers to Simplify Remote-Device Setup", LAN Times, May 13, 1996

Mullaney, P., "Overview of a Web-based Agent" Simple Times, July, 1996

Raggett, D., "HTML Tables", RFC 1942, May 1996

Robinson, D., "The WWW Common Gateway Interface Version 1.1" (Internet Draft), February 15, 1996

Wellens, C., and Auerbach, K., "Towards Useful Management", Simple Times, July, 1996

7 BIOGRAPHY

Benjamin Reed has been working since 1995 on system and network management solutions at IBM Almaden Research Center. He is currently working on his PhD in Computer Science at the University of California Santa Cruz.

Michael Peercy received his BS in Computer and Electrical Engineering from Purdue University and his MS and PhD from the University of Illinois. He has been a Visiting Scientist and Software Engineer at IBM Almaden Research Center since 1994, working in the area of systems management.

Ed Robinson has been working for Binary Consulting, Inc. on integrating web technology with modern compiler technology. He is a PhD candidate at the University of Houston. Prior to working at Binary Consulting, he was at IBM Almaden Research Center where he contributed to the work presented in this paper.