INTERNATIONAL JOURNAL OF NETWORK MANAGEMENT
Int. J. Network Mgmt 2000; **10**:261–275

An efficient and lightweight embedded Web server for Web-based network element management

By Hong-Taek Ju,[*] *Mi-Joung Choi and James W. Hong*

An Embedded Web Server (EWS) is a Web server which runs on an embedded system with limited computing resources to serve embedded Web documents to a Web browser. By embedding a Web server into a network device, it is possible to provide a Web-based management user interface, which are user-friendly, inexpensive, cross-platform, and network-ready. This article explores the topic of an efficient and lightweight embedded Web server for Web-based network element management. Copyright © 2000 John Wiley & Sons, Ltd.

Introduction

As the World-Wide Web (or Web) continues to evolve, it is clear that its underlying technologies are useful for much more than just browsing the Web. Web browsers have become the *de facto* standard user interface for a variety of applications. This is because Web browsers can provide a GUI interface to various client/server applications without a client application. An increasing number of Web technologies can also be applied to network element management.

Web-based network element management gives an administrator the ability to configure and monitor network devices over the Internet using a Web browser. The most direct way to accomplish this is to embed a Web server into a network device and use that server to provide a Web-based management user interface constructed using HTML,[5] graphics and other features common to Web browsers.[4] Information is provided to the user by simply retrieving pages, and information is sent back to the device using forms that the user completes. Web-based management user interfaces (WebMUIs) through embedded Web servers have

Hong-Taek Ju received his BS degree in computer science from Korea Advanced Institute of Science and Technology (KAIST) in 1989 and MS degree in Computer Science and Engineering from Pohang University of Science and Technology (POSTECH) in 1991. From 1991 to 1997, he worked at DAEWOO Telecom. Currently, he is a PhD candidate in the Department of Computer Science and Engineering, POSTECH. His research interests include distributed processing and network management.

Mi-Joung Choi received her BS degree in computer science from Ewha Womans University. She is currently a graduate student in the Department of Computer Science and Engineering, POSTECH. Her research interests include Web-based network management and policy-based network management.

James W. Hong is an associate professor in the Department of Computer Science and Engineering, POSTECH, Pohang, Korea. He has been with POSTECH since May 1995. Prior to joining POSTECH, he was a research professor in the Department of Computer Science, University of Western Ontario, London, Canada. Dr Hong received BSc and MSc degrees from the University of Western Ontario in 1983 and 1985, respectively, and PhD degree from the University of Waterloo, Waterloo, Canada in 1991. He has been very active as a participant, program committee member and organizing committee member for IEEE CNOM sponsored symposiums such as NOMS, IM, DSOM and APNOMS. For the last several years, he has been working on various research projects on network and systems management, which utilize Web, Java and CORBA technologies. His research interests include network and systems management, distributed computing and traffic engineering and planning. He is a member of IEEE, KICS, KNOM and KISS.

[*]*Correspondence to: Hong-Taek Ju, DPNM Laboratory, Department of Computer Science and Engineering, Pohang University of Science and Technology, San 31, Hyojadong, Namgu, Pohang, Korea.*
Email: juht@postech.ac.kr

many advantages: ubiquity, user-friendliness, low development cost and high maintainability.

Embedded Web Servers (EWSs)[1–3] have different requirements, such as low resource utility, high reliability, security and portability, for which general Web server technologies are unsuitable. Above all, due to resource scarcity in embedded systems it is important to make EWSs efficient and lightweight. There are also design issues such as HTTP[6,7] and embedded application interface. In embedded Web server usage, Java applets can play an important role for making embedded Web servers truly useful for management applications.

In this paper, we present our research to develop an efficient and lightweight EWS for Web-based network element management. We first propose the architecture of an embedded Web server that can provide a simple but powerful application interface for network element management. We then present the design and implementation of POS-EWS, an embedded Web server that we have developed for Web-based network element management. Finally, we present the results of POS-EWS's performance and EWS optimization methods for making an efficient and lightweight EWS. There are many commercial EWS products on the market for Web appliances, but our work is a good example of making an efficient EWS suitable for Web-based network element management.

The organization of the paper is as follows. In the second section we present an overview of EWSs, and describe the EWS-WebMUI and EWS requirements. In the next two sections we present the EWS design and implementation of our proposed EWS architecture, respectively. In the fifth section we evaluate POS-EWS's performance and explain our methods for optimizing POS-EWS. In the sixth section we briefly investigate the available offerings of EWS products focusing on their features and the approximate code size needed. In the final section we summarize our work and discuss possible future work.

Embedded Web Servers and Web-based Management User Interface

In this section, we briefly overview embedded Web servers, comparing them with general Web servers. Also, we describe the EWS-WebMUI and EWS requirements that we must consider during development.

—Embedded Web Server—

General Web servers, which were developed for general-purpose computers such as NT servers or Unix and Linux workstations, typically require megabytes of memory, a fast processor, a pre-emptive multitasking operating system, and other resources. A Web server can be embedded in a device to provide remote access to the device from a Web browser if the resource requirements of the Web server are reduced. The end result of this reduction is typically a portable set of code that can run on embedded systems with limited computing resources. The embedded system can be utilized to serve the embedded Web documents, including static and dynamic information about embedded systems, to Web browsers. This type of Web server is called an Embedded Web Server (EWS).[1–3]

EWSs are used to convey the state information of embedded systems, such as a system's working statistics, current configuration and operation results, to a Web browser. EWSs are also used to transfer user commands from a Web browser to an embedded system. The state information is extracted from an embedded system application and the control command is implemented through the embedded system application. In many instances, it is advisable for embedded Web software to be a lightweight version of Web software. For network devices, such as routers, switches and hubs, it is possible to place an EWS directly in the devices without additional hardware.

—EWS-WebMUI—

WebMUI and EWS-WebMUI—The rapid proliferation of Web-based management makes it clear that schemes using HTTP and standard Web browsers provide benefits to both users and developers. Most Web-based management applications provide an interface to the status reporting, configuration, and control features of managed objects. Several such Web management approaches have been proposed thus far. Sun Micro-systems is pushing its Java Management eXtension (JMX)[8] and Microsoft, Compaq and Intel are touting Web-based Enterprise Management (WBEM).[9] However, both approaches are sufficiently complex that many small network devices would find it very difficult to implement them.

By embedding a Web server, Web documents and management applications into an embedded system, a Web-based Management User Interface (WebMUI) can be provided directly to system administrators (an EWS-WebMUI). Therefore, an EWS-WebMUI is the direct result of embedding a Web server, Web documents and management applications into an embedded system. The Web documents give a display form of management information, a collection of manageable data that is monitored or configured for managing an embedded system.

By embedding a Web server in a network device, the device can serve up Web documents to any Web browser.

Advantages of EWS-WebMUI—By embedding a Web server in a network device, the device can serve up Web documents to any Web browser. These Web documents become the GUI interface to the device. Consequently, few techniques need to be learned for management interface of the new device. Because Web documents can be displayed directly from files that may be edited with either ordinary text editors (for HTML) or specialized authoring tools, it is easy to quickly prototype the look and feel of a WebMUI. Alternatives can be explored and reviewed without ever actually embedding the interface into the system. If the mechanisms used to embed the interface are properly designed, changes made to the Web documents can be quickly imported to the embedded system with little or no change to the management application code. This translates into the potential for better, more useful interfaces in less development time.

EWS-WebMUIs also have the advantage of a platform independent graphical user interface. The SNMP[10] management scheme usually consists of an SNMP based Network Management System (NMS). Most NMSs give users the option of using a graphical interface based on MS-Windows or X-Window as opposed to the command line interface. Most NMS users demand specific platforms, such as OS, or computer hardware in order to install and execute the NMS. By contrast, an EWS-WebMUI does not demand any specific platform because Web browsers are available for virtually all computers.

While the EWS-WebMUI concept appears straightforward and perhaps even commonplace, the implications are deeper than first appears. By placing the GUI within the device itself, the device is now self-contained and need not be matched with a corresponding version of a user management application program; the problems inherent in providing separate user interface software disappears; there is no risk of the user having an old version of the user application software that does not support all the features of latest devices; and users can upgrade some systems to the latest release without having to change the management software they use because the necessary part of upgrade is only the EWS-WebMUI. Consequently, there are no porting or distribution efforts for the user application program.

Additionally, it is usually possible to upload Web documents to the embedded system so that a device can receive an upgrade to its management interface from a remote location on the network. This feature makes it possible for developers to upgrade all devices over the network from the one point. High maintainability for EWS-WebMUI is a direct result of ease of Web document development and one point upgrade.

Design

In this section, we present our design result that includes a functional architecture and a process structure of EWS.

—EWS Architecture—

We have designed an EWS that consists of five parts: an HTTP engine, an application interface module, a virtual file system, a configuration module, and a security module. The design architecture of our EWS is illustrated in Figure 1.

The most important part of the EWS is an HTTP engine, which serves a client's request. The

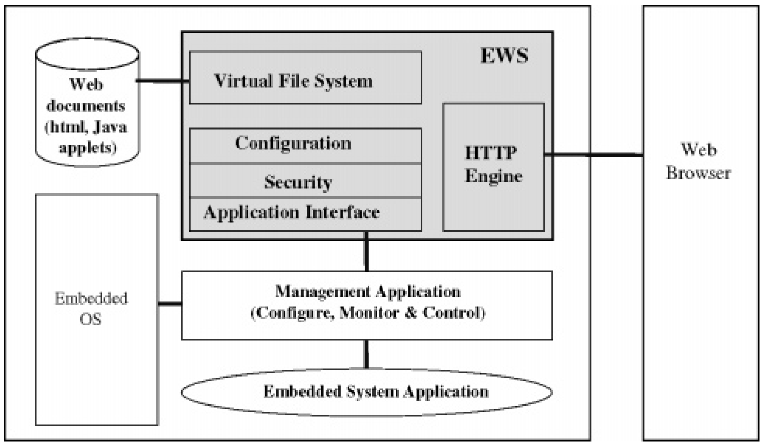
Web documents (html, Java applets)
EWS
Virtual File System
Configuration
Security
Application Interface
HTTP Engine
Web Browser
Embedded OS
Management Application (Configure, Monitor & Control)
Embedded System Application

Figure 1. EWS architecture

minimum requirement for an HTTP engine is that it must be compliant with HTTP specifications. Unlike general Web servers that start a new thread or process whenever a new connection is made, normally an HTTP engine supports multiple simultaneous users while running as a single process. The number of processes that the server requires can impact on both RAM usage, due to the stack space per task, and CPU usage. Next, we explain an HTTP transaction process using a state transition diagram.

In an EWS, the application interface module enables developers to add new management functionality. With any off-the-shelf Web authoring tool, it can merge Web documents with management application programs to generate specific dynamic management information. This module provides mechanisms for interacting with the embedded application. Embedded Web server software must provide mechanisms for the embedded application to generate and serve Web pages to the browser, and to process HTML form data submitted by the browser. One possible solution is modeled after the Common Gateway Interface (CGI)[15] found in many traditional Web servers. In this model, each URL[16] is mapped to a CGI script that generates the Web page. In a typical embedded system, the script would actually be implemented by a function call to the embedded application. The application could then send raw HTML or other types of data to the browser by using an interface provided by the embedded Web server software.

Another solution is to use Server-Side Include (SSI).[5] With this approach, Web pages are first developed and prototyped using conventional Web authoring tools and browsers. Next, proprietary markup tags that define server-side scripts are inserted into the Web pages. The marked-up Web pages are then stored in the device. When a marked-up Web page is served, the embedded Web server interprets and executes the script to interface with the embedded application. In order to offload substantial Web server processing from the embedded system at run time, a preprocessor tool can be used. The preprocessor enables sophisticated dynamic Web-page capabilities by performing complex tasks up front and generating an efficient and tightly integrated representation of the Web pages and interfaces in the embedded system.

The virtual file system (VFS) provides the EWS with virtual file services, which are *file_open* for opening the file, *file_read* for reading the file, and *file_close* for closing the file after reading. The file system has a data structure storing file information such as file size, last modified date, etc. The data structure for an HTML documents file needing dynamic information must store the pointer of the script and the function name called by the script. To construct this VFS we need a Web compiler. The Web compiler supports any format, such as Java, GIF, JPEG, PDF, TIFF, HTML, text, etc. It compiles these files into intermediate C-codes and then compiles & links them with the Web Server codes. The resulting structure does not require a

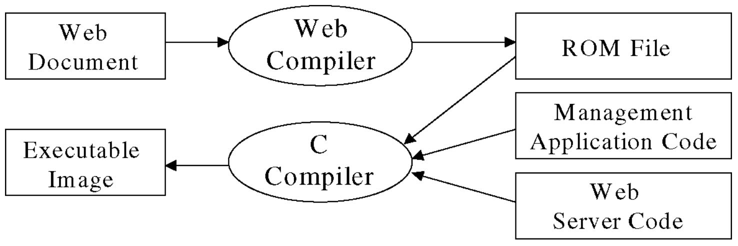

Figure 2. Process of a web server making a virtual file system

file system, yet the files are organized like in a file system—a virtual file system. The Web browser traverses this virtual file system just as if it were an actual file system. Figure 2 illustrates the process of a Web server making a virtual file system.

Security is an important concern in network management. Therefore, an EWS generally has a security and/or configuration module. Security is accomplished by defining security realms on a server and username/password access to each realm. When a request comes in for an object in a protected realm, the server responds with a response code of 401 (Unauthorized). This will force a browser to prompt the user for a username/password pair. The original object request will be resubmitted with the username/password, base-64 encoded, in the request header. If the server finds the login correct, then it will return the requested object, otherwise, a 403 forbidden response is returned. The configuration module provides the administrator with the functionality to set the embedded Web server configuration from any standard Web browser. The configuration environment variables passed at startup define the number of concurrent connections, socket port, own host name, root file path, default 'index', inactivity timeout and time zone. Common usage of Web browsers makes it a more important matter to protect abnormal access to the sensitive information of network devices, especially those that involve equipment configuration or administration.

—EWS Process Structure—

We designed an EWS as a finite state machine (FSM), which processes an HTTP request in a sequence of discrete steps. Figure 3 shows the state transition diagram of the HTTP engine. In order to support multiple connections in a single thread environment, multiple finite state machines are run by a scheduling system which uses a lightweight

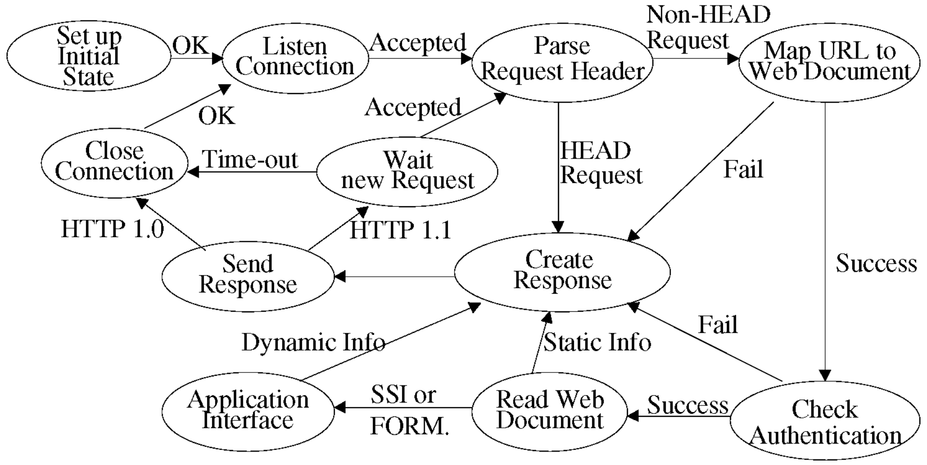

Figure 3. EWS finite state machine

Explore Litigation Insights

Docket Alarm provides insights to develop a more informed litigation strategy and the peace of mind of knowing you're on top of things.

Real-Time Litigation Alerts

Keep your litigation team up-to-date with **real-time alerts** and advanced team management tools built for the enterprise, all while greatly reducing PACER spend.

Our comprehensive service means we can handle Federal, State, and Administrative courts across the country.

Advanced Docket Research

With over 230 million records, Docket Alarm's cloud-native docket research platform finds what other services can't. Coverage includes Federal, State, plus PTAB, TTAB, ITC and NLRB decisions, all in one place.

Identify arguments that have been successful in the past with full text, pinpoint searching. Link to case law cited within any court document via Fastcase.

Analytics At Your Fingertips

Learn what happened the last time a particular judge, opposing counsel or company faced cases similar to yours.

Advanced out-of-the-box PTAB and TTAB analytics are always at your fingertips.

API

Docket Alarm offers a powerful API (application programming interface) to developers that want to integrate case filings into their apps.

LAW FIRMS
Build custom dashboards for your attorneys and clients with live data direct from the court.

Automate many repetitive legal tasks like conflict checks, document management, and marketing.

FINANCIAL INSTITUTIONS
Litigation and bankruptcy checks for companies and debtors.

E-DISCOVERY AND LEGAL VENDORS
Sync your system to PACER to automate legal marketing.

WHAT WILL YOU BUILD? | sales@docketalarm.com | 1-866-77-FASTCASE

fastcase® Smarter legal research.