*Journal of Network and Systems Management, Vol. 9, No. 1, 2001*

# EWS-Based Management Application Interface and Integration Mechanisms for Web-Based Element Management

**Hong-Taek Ju,[1] Mi-Joung Choi,[1] and James W. Hong[1]**

Web-based network element management provides an administrator with 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 [Embedded Web Server (EWS)] into a network device, and use that server to provide a Web-based management user interface constructed with HTML, graphics, Java and other features common to Web browsers. In this paper we present EWS-based management application interface mechanisms for use between embedded management applications and embedded Web servers. We propose a guideline for choosing an efficient interface mechanism, which is based on the characteristics of management information and Web documents. A Web-based management user interface through embedded Web servers has many advantages such as ubiquity, platform independence and user-friendliness. In order to be truly useful, a Web-based management user interface must have a low development cost and a short development time. We provide effective integration mechanisms for each interface. We validate these mechanisms by implementing them in an Internet router.

**KEY WORDS:** Embedded web server; web-based network management; network element management; management application interface mechanism; management application integration mechanism.

## 1. INTRODUCTION

Network devices can be equipped with Web servers for providing Web-based network element management. This type of Web server is called an Embedded Web Server (EWS) [1, 2]. For EWS-equipped devices, Administrators

[1] Department of Computer Science and Engineering POSTECH, San 31 Hyoja, Namgu, Pohang 790-784 Korea. E-mail: {*juht, mjchoi, jwkhong*}*@postech.ac.kr*



1064-7570/01/0300-0031$19.50/0 

point their Web browsers at the home page residing within the network devices in order to configure, monitor and control them. Administrators can manage individual devices remotely by using off-the-shelf Web browsers. This management scheme provides an administrator with a simple but enhanced, more powerful and ubiquitous user interface. The EWS is limited directly by the device resources. Since most embedded devices by nature contain small amounts of memory and storage, optimization techniques must be applied in EWS development [3–5].

An EWS communicates via HTTP to a Web browser by receiving a request which indicates the operation that the administrator wants to perform, and sending a response in the form of a Web page for viewing. The URL determines which operation to perform, and it is executed by the embedded system application. The most familiar operation is a system date and time setting with the URL specifying the current time. Further, to incorporate live data into a response, the EWS must dynamically create the Web page to return to the Web browser. The live data is extracted from the embedded system application by the EWS.

In our earlier work [5], we proposed a network element management architecture with an EWS as a core component. We have also developed an HTTP 1.1 compliant embedded Web server (called POS-EWS) that supports our proposed architecture [6]. POS-EWS is a tiny Web server (30 Kbytes binary code size and 30 Kbytes runtime memory) compared with a general Web server, such as Apache (4 Mbytes binary code size and 3 Mbytes runtime memory). In this paper we propose four basic interface mechanisms for use between a management application of an embedded system and an EWS. A developer can choose an appropriate interface mechanism depending on the management information characteristics or types of Web documents. An integration mechanism between a Web document and a management application enables developers to add new management functionalities by merging Web documents with management application programs that generate specific dynamic management information. For rapid and low cost development, an easy but powerful integration mechanism must be provided. In this paper we provide effective integration mechanisms for each interface.

In Section 2 we summarize an EWS-based network element management architecture which is the result of our previous work, and give a more detailed explanation for the role of the interface mechanism between EWS and management application of the embedded system. Section 3 describes the interface mechanisms between management applications and Web documents. For each interface mechanism described in Section 3, an effective integration mechanism is presented in Section 4. An applied example is described in Section 5. Conclusions and future directions of our work are described in Section 6.

## 2. EWS-BASED NETWORK ELEMENT MANAGEMENT ARCHITECTURE

Two principal industrial bodies are playing a leading role in contributing standardization to Web-based network management: the Web-Based Enterprise Management (WBEM) [7] and the Java Management eXtension (JMX) [8, 9]. The WBEM from Distributed Management Task Force (DMTF) is an initiative based on a set of management and Internet standard technologies developed to unify the management of enterprise computing environments. Despite extensive work on WBEM currently under way, it is still not quite ready for deployment to embedded systems. The JMX is based on Java technology so it draws on Sun's experience with Java management. The technology dependency on Java results in less generality for applying to embedded systems.

The most direct way to achieve a Web-based Management User Interface (WebMUI) is to place a Web server and Web pages directly on the network device and allow a browser to communicate one-to-one with that device. This has the primary advantage of simplicity. By embedding a Web server and Web documents into the network devices, the WebMUI can be provided directly to network operators from the device (EWS-WebMUI). An EWS-based network element management architecture for EWS-WebMUI is depicted in Fig. 1. The architecture was already described in our previous literature [4, 5]. In the rest of this section we summarize it and give more detailed description about management applications.

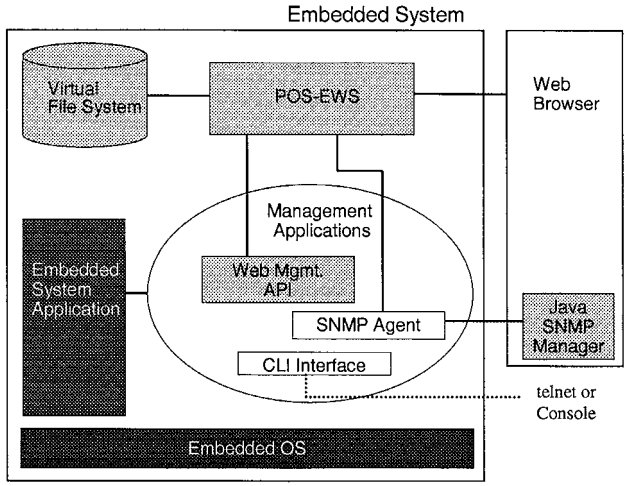


**Fig. 1.** EWS-based Network Element Management Architecture.

EWS-WebMUI provides an interface for status reporting, configuration, and control features of the network devices. This functionality requires dynamic content as well as static content because management information is either constantly changing or changed by an operator. Consequently, when a request arrives from a browser, the server determines the current value of the managed object, forms that information into an HTML document [10], and then returns it to the browser. Further, the network device must notify the user of events and possibly allow the user to perform some action based on them. Management applications, as part of embedded system applications, provide such dynamic information to the Web server. Basically, if a request contains control information, the server passes the information to the management application, which in turn issues a control command according to the information.

Web Management Application Program Interface (Web Mgmt. API), one of the management applications of embedded system, exposes the managed object to the EWS. It is specifically designed to map a managed object to the Web presentation/control object. Web Mgmt API is modeled after Common Gateway Interface (CGI) [11] and Server Side Include (SSI) [10]. In network devices such as routers and switches, the SNMP agent is for all intents and purposes required. The EWS-based network management architecture does not throw away that investment. The SNMP is used as a source of management information to be integrated with a Web page.

Where Java applets are used to implement the WebMUI, real-time data can be placed alongside static HTML in the same page. Java applets are automatically downloaded by a browser as a separate application that are used within an HTML page. Once the applet is loaded, it has control over where it gets its data and how to display or manipulate that data. A Java SNMP manager is a Java applet program that communicates with an embedded SNMP agent and plays the role of manager.

Web pages must be stored in the embedded system to be served at run time. It is stored in a form of file in the case of general-purpose computers, but a file system is not a basic requirement for an embedded system. POS-EWS supports the virtual file system in order to store Web pages, where an embedded OS does not support a file system. The virtual file system provides POS-EWS with file services, which are *file open*, *read* and *close*. All system programs require services from the embedded Operating System.

## 3. MANAGEMENT APPLICATION INTERFACE MECHANISMS

Management information can be classified on the basis of update period, direction of information flow or object of information source. From the viewpoint of update period, some management information changes dynamically, and some does not change at all. Further, some information possesses real-time char-

acteristics, for example a dynamic graphic form in a real-time update or event notification to perform some action. Regarding the direction of information flow, some information can originate from a Web browser and travel to a Web server and vice versa.

As depicted in Fig. 1, a management application includes a Web Management API that is used by EWS exclusively, and SNMP agent that has standard SNMP interface. These two elements can be management information source for providing WebMUI. Also, different types of Web documents have very distinct characteristics, especially between HTML and Java [10, 12]. All things considered, it is more desirable to provide several effective interface mechanisms between the Web document and management applications. This variety in determining the interface mechanism enables developers to choose an appropriate mechanism on the basis of characteristics of management information and Web documents. We define four interface mechanisms, which are depicted in Fig. 2: (a) CGI-Type; (b) SSI-Type; (c) SSI SNMP-Type; and (d) Java SNMP-Type. Each mechanism has its own advantages over the others. They are explained and compared below.

### 3.1. CGI-Type Interface

An EWS must provide mechanisms in order for the management application to generate and serve Web pages to be sent to the browser, and to process HTML form data submitted by the browser. One possible solution is modeled after the Common Gateway Interface (CGI) [11] found in many traditional Web servers. In this model, each URL [13] 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 an EWS. The (a) arrow in Fig. 2 shows the data flow of the CGI-Type interface mechanism. While this approach is perhaps the easiest for embedded Web server developers, it is by far the most difficult for WebMUI developers because CGI scripts are tedious to write. Once written, the display of the Web page can only be determined when the script is executed. In an embedded system, this implies building the executable code, downloading it into flash memory, and booting the device before the Web page can be viewed by a browser. Therefore, CGI solutions require a long development time and are difficult to maintain. But for management applications that process the user commands that have much parametric information and produce simple Web pages to contain only success or fail information, this mechanism is a good solution because the interface mechanism is simple. No special integration mechanism between a Web document and the management application is required—just a few useful library functions are enough to integrate them.

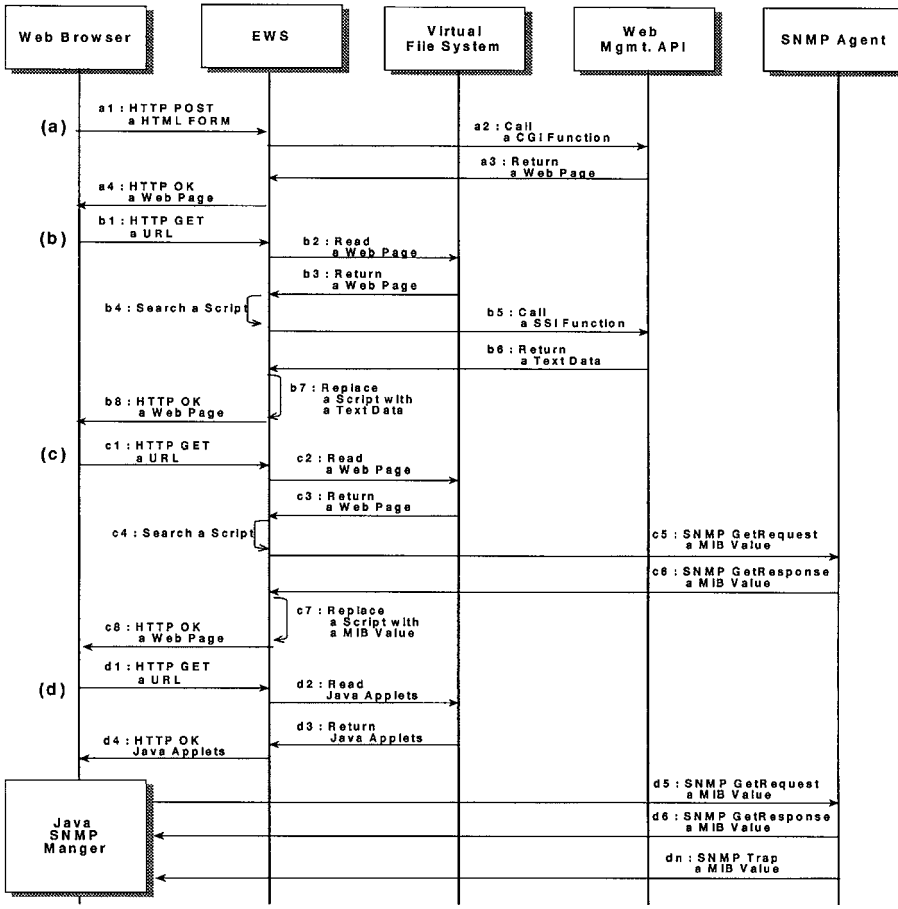


**Fig. 2.** Management Application Interface Mechanisms.

## 3.2. SSI-Type Interface

Another solution is to use Server-Side Include (SSI) [10]. With this approach, Web pages are first developed and prototyped using conventional Web authoring tools and browsers. Next, 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 of the tags in order to interface with the management application. The server maintains a database for mapping script names into management application functions. In a general Web server, parameters can be passed to the Server Side Include routine by the one of two methods: in the

URL or in the Web page. POS-EWS does not support parameter passing of SSI in the URL because of security problems [14].

For example, a scripting language on the basis of tags could define an interface to invoke application functions used to generate the dynamic part of a Web page. The (b) arrow in Fig. 2 shows the information flow of this interface mechanism. An embedded system initialization code is used to register the server-side scripts with the function calls of the management application. The code for a function call is invoked when the server interprets server-side scripts. Since SSI is a substitution mechanism, where text is inserted into a document at tagged locations and CGI is a generation mechanism, where the whole Web page is produced by a real program, CGI shows higher flexibility than SSI. But SSI requires little development expertise beyond writing the Web page and also has a lower development time than CGI.

However, interpreting scripts at runtime in an embedded system may impact system performance. Moreover, a significant amount of memory is required to maintain a database for mapping script names into embedded software functions and variables. In order to offload such substantial Web server processing from the embedded system at runtime, a special integration mechanism, a preprocessing, is used. Further details about the preprocessor tool, Web compiler for POS-EWS, are provided in Section 4.2. Where a Web document displays status information in the form of a table or other formatted style, this interface mechanism is an ideal solution because complicated display formats can be separated from the management information production method, which is almost system dependent. Conversely, information production logic is independent of its usage for display. A typical and proper use of this interface mechanism is with state report generation. The marked-up tags used by POS-EWS are contained within standard HTML comment elements. These elements are blocks of text surrounded by the <!--# and --> sequences.

### 3.3. SSI SNMP-Type Interface

Simple Network Management Protocol (SNMP) [15, 16] is the most widely used management method for managing network devices on the Internet. Its simplicity enables it to be implemented on small platforms without much difficulty. Presently, most network devices are equipped with an SNMP agent. With integration of SNMP and EWS-WebMUI, the advantages of EWS-WebMUI are preserved with no loss of SNMP agent functionality.

The SSI-Type interface mechanism does not restrict the form of management applications; it is just a set of function calls. When the management application is an SNMP agent, a more elegant and automated method can be introduced. The SSI SNMP-Type interface mechanism is a specialized case in limiting the form of management application to an SNMP agent. Basically, the devel-

opment process and integration mechanism are the same as those of SSI-Type mechanism, but the management application functions called by the script interpreter are specialized to the interface provided by an SNMP agent.

There are two ways of interfacing with the SNMP agent; one of them is to use the local loop interface of an SNMP stack and the other is to use the SNMP MIB implementation directly, which is more efficient in resource usage. The direct interface to the SNMP MIB implementation depends on the implementation of the SNMP agent. But the local loop interface of the SNMP stack provides a standard interface just like a remote access. Therefore, this method has higher portability and reusability than the direct interface to the SNMP MIB implementation. The SNMP protocol itself is simple and light so the performance overhead of the local looped SNMP stack method can be negligible, especially in case of SNMPv1. POS-EWS uses the local looped SNMP stack method in order to interface with an SNMP agent using the SSI-Type mechanism.

Because the SNMP agent is used as a management application, the management information is also defined in SNMP SMI [17]. This well-defined management information makes it easy for a developer to understand management information to be appeared in Web pages, while formalized specification method is not supported in CGI-Type or SSI-Type interface mechanism. All network systems deployed at this time are equipped with an SNMP agent, so the use of a legacy SNMP agent can reduce the development time of a WebMUI.

A Web-based SNMP MIB browser is the most common use of SSI SNMP-Type interface mechanism. The browser gives the traversal functionality on the defined MIB treefor investigating the up-to-date values of SNMP MIB variables. The Web interface for the MIB browser takes the typical form of a user interface. Accordingly, the Web pages of the browser can be produced automatically from SNMP MIB definition.

The MIB2HTML compiler [18] reads SNMP MIB definitions and generates Web pages of the MIB browser. Marked-up tags used by POS-EWS in order to retrieve or set MIB variables at run time are inserted into the generated Web documents. A detailed explanation about MIB2HTML is provided in Section 4.3. The management application interface is identical for all MIB definition by use of local loop interface of a SNMP stack. POS-EWS supports the interface program in a form of library code. Therefore, Web-based MIB browser of the network device embedding an SNMP agent can be developed without an additional management application program or any other sort of program. Only Web pages generated by MIB2HTML is added to the embedded system.

### 3.4. Java SNMP-Type Interface

HTTP/HTML is a client driven scheme [19]. One side effect is that once a page is served from a Web server to a Web browser it becomes static and

does not change even if management data has changed on the server side. For dynamic management information, this is not very useful. There are several methods to provide dynamic interfaces. Refresh buttons can be placed on pages with dynamic information so that the user can press them to reload the page from the server with updated information. This is obviously very cumbersome and not very efficient.

A method to solve this problem is client pull [20]. The http-equiv meta tag, REFRESH, can be used in page response headers to force the browser to re-request a page after a certain number of seconds defined by the variable CONTENT=*x*, where *x* is the numberof seconds between refreshes. This method can be employed to periodically update the page a user is viewing by continuing to re-request and display it. The disadvantage of this method is that the entire screen will go blank as the page is reloaded. Since communication can be slow, this is usually fairly noticeable. Another method is server push [20]. Once a page that contains multi-part content has been loaded, it is possible to maintain a connection through which the server can continue to send updated data. For handling continuous data, a browser uses a plugin for displaying graphics. The main disadvantages of a plug-in is that the software must be loaded and installed on the client computer. Server push is generally more efficient than client pull, since there is no need to connect and tear down TCP/IP session frequently. Since a connection is held open over time, the server must be willing to accept dedicated allocation of a TCP/IP port, which may be an issue for servers with a sharply limited number of TCP/IP ports.

To be truly useful for management applications, pages will have to be constructed dynamically so that real-time data can be placed alongside static HTML in the same page. For common types of real-time data, such as traffic monitoring and CPU load, users want to see data displayed in a dynamic graphic form in a real-time update. This is the function of Java applets [12], which are automatically downloaded by a browser as a separate application that gets used within an HTML page. Once the applet is loaded, it controls from where it gets its data and how to display or manipulate that data. Java applets by nature are cross-platform and act the same within any browser.

Another situation we must consider is when the network device needs to notify a user of an event and possibly allow the user to perform some action based on the notification. The event notification is another issue since asynchronous communication method is not supported in HTTP. A Web browser would have to refresh a page or the page would require client-pull refresh periodically to determine if there is a new event. Fortunately, it is a straightforward task to design Java applets to handle this kind of situation if Java applets are programmed on the basis of the Java security model as a mobile code from a managed element to Web browser. Normally, the browser must be active with that applet at all times and a server, not a Web server, for sending an event

notification or responding query for a new event must be introduced into the network device. It is commonly accepted that Java applets have a small footprint while the Java Virtual Machine (JVM) required to execute the byte code of a Java applet is typically very large. Because Java applets containing only byte code are fairly compact, the code can be stored in an embedded system without a large increase in memory requirement. Since Java is not actually executed on the network device, a JVM is not required in the device.

Java implementation of an SNMP manager mediates between an SNMP agent and a Web browser. The Java SNMP manager source code is written and compiled to produce a Java SNMP applet. This applet is stored in a network device and is transferred by the EWS to the browser over the network at run time. After loading on the JVM of a browser, the Java SNMP manager applet communicates with the SNMP agent in the network device and enables the administrator to control and monitor the network device through the browser, using SNMP messages. In addition to the Java SNMP applet, the network device in this scenario must store at least one HTML document containing a reference to the applet. The HTML document would be loaded in advance into the Web browser, which would automatically request the Java SNMP applet that is referenced by the previous HTML.

Therefore, an SNMP stack is included in the Java applet as well as the manager function. The code size of a Java SNMP applet is small enough that it can be embedded into the network device (30 Kbytes in compressed format) because SNMPv1 is simple (three basic message types and simple message format) and light (uses UDP as its transport protocol, and thus does not have connection setup or acknowledgement overhead). SNMP defines traps that can be directed toward one or more trap receiver station. If a trap management application is implemented as a Java SNMP applet and loaded from the network device, traps can be collected and viewed together with a Java SNMP applet, and appropriate responses taken.

### 3.5. Comparison of Application Interface Mechanisms

In this subsection, the characteristics of previously described application interface mechanisms are compared. We assume that the SSI SNMP interface mechanism implements the SNMP MIB browser. Table I summarizes a comparison of the interface mechanisms. Unlike programming support on a general-purpose computer, programming support on an embedded system is so limited that in most cases programmers cannot use advanced shell or script programs such as csh, Perl and Tcl/Tk. Building blocks of system program are written in C or C++.

Each interface mechanism has a different development method for Web documents. For the CGI-Type mechanism, the management application program

**Table I.** Comparison of Interface Mechanisms

| | CGI-type | SSI-type | SSI SNMP-type | Java SNMP-type |
|---|---|---|---|---|
| Web documents development method | Management application program | Web authoring tool + marked-up tags insertion | MIB2HTML compiler | Java applet program |
| Web documents development cost | High | Low | Very low | High |
| Management application programming | Necessary | Necessary | Unnecessary (Library code) | Unnecessary (SNMP Agent) |
| Management information source | Web interface | Web interface | SNMP Agent | SNMP Agent |
| Network load/ Web page | 1 HTTP requests | 1 HTTP requests | n-SNMP & 1-HTTP | 1-HTTP & Continuous SNMP |
| CPU Load | Small | Medium | Large | Medium |
| Code size | Management application program | HTML + management application program | HTML + Management application program | Java class |
| Portability | Low | Middle | Middle | High |
| Event Support | No | No | No | Yes |

for the Web interface generates Web documents; consequently, its development cost is high. For the SSI-Type, Web documents are developed using a Web authoring tool and marked-up tags are inserted into the Web pages. Compared with the CGI-type, its development cost is low because Web authoring tools are used. For SSI SNMP-Type in the case of SNMP MIB browser implementation, MIB2HTML compiler generates whole HTML pages for the MIB browser, so its development cost is negligible. For Java SNMP-Type, Web documents are developed in Java applet classes on the supported Java SNMP stack. Its development cost is high.

In the case of wanting to add new management functionality using the CGI-Type and SSI-Type, it is necessary for a developer to add a new Web interface module of management application program that generates or supports HTML, respectively. However, when using SSI SNMP-Type and Java SNMP-Type it is unnecessary to add a new management application on the assumption that a legacy SNMP agent supplies the necessary information.

With resource usage, the CGI-Type uses the least amount of CPU, memory and network resources because an EWS only calls on a CGI function just once and sends the whole result text of the function call without any processing.

Compared with CGI-Type, SSI-Type uses more CPU resources because it needs to search the script of marked-up tags and function calls and text replacement on each marked-up tag. Further, SSI-Type uses a file system for storing HTML documents while the CGI-Type does not use any type of file system. SSI SNMP-Type uses the largest amount of CPU, memory and network load because it needs to use an additional local looped SNMP stack for the SSI style. A Java SNMP-Type mechanism does not require script parsing, but the Java SNMP manager that is downloaded from the embedded system to a Web browser issues SNMP requests continuously. Compared with CGI-Type, Java SNMP manager downloading uses less computing resources than generating Web page by CGI-Type, but continuous SNMP communication uses computing resources after downloading.

Management application of CGI-Type interface is programmed by system programming language, not well-known script language, and it uses EWS interface library functions that are supported by the embedded Web server, not a well-known interface mechanism such as standard input/output in UNIX. Its portability is marginally low because the CGI-Type management program depends on system environment and library functions of the embedded Web server. In the SSI-Type interface, at least portability of Web pages is guaranteed, but SSI script programs to be executed by EWS have the same portability as CGI-Type interface programs. The SSI SNMP-Type management application has the same level of portability with the SSI-Type management application because two interface mechanisms are the same. The Java SNMP manager is just stored at the embedded system, runs on the Web browser and communicates via SNMP protocol. Its portability is high because there is no dependency on the Web server or on the embedded system. Since only Java SNMP-Type supports asynchronous communication, it is possible to implement event notification with only Java SNMP-Type interface mechanisms.

## 4. MANAGEMENT APPLICATION INTEGRATION MECHANISMS

An integration mechanism between Web documents and management applications enables developers to add new management functionality by merging Web documents, which provides a user interface format with management application programs that generate specific management information. Through the integration mechanism, Web document development can be separated from management application development. Consequently, the management application developer no longer needs to consider the user interface of Web documents through program logic.

Also, Web document prototyping, involving different user interfaces with a deployed Web document, no longer requires code changes and recompilation

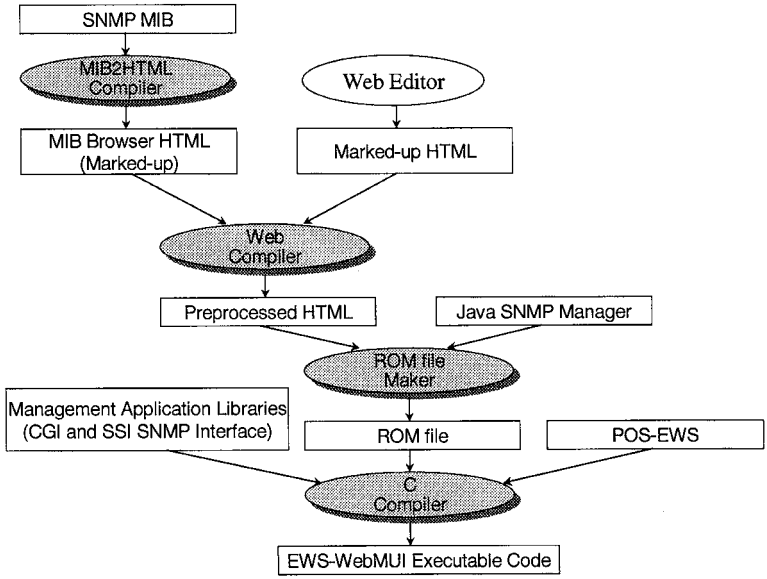


**Fig. 3.** Management Application Integration Mechanisms.

of the management application program if the same management application interfaces are used and if a Web page can be uploaded dynamically. For rapid development, an easy but powerful integration mechanism must be provided.

The integration mechanisms that are used in POS-EWS are depicted in Fig. 3. There are two compilers, MIB2HTML [18] and a POS-EWS Web Compiler [3], and a C cross-compiler for the embedded system. The compilers are used for accelerating Web document development time. The MIB2HTML compiler generates marked-up HTML files from the SNMP MIB definition for the SNMP MIB browser and the POS-EWS Web compiler processes the marked-up HTML file to be used by POS-EWS at runtime. The ROM file maker generates binary coded ROM files for supporting the virtual file system, where embedded OS does not support a file system. Binary coded ROM file consists of character arrays and the virtual file system is a limited set of read-only functions built into the ROM. The character array is the result of simple conversion and compression from plan source file to C language array.

The management application includes user-programmed function for CGI-Type and SSI-Type interface and library functions supported by POS-EWS for SSI SNMP-Type interface. This management application program is combined with POS-EWS source code in order to generate an EWS-WebMUI executable code. The EWS-WebMUI is executed in a single thread process in the embedded system [5].

### 4.1. CGI Library

In the CGI-Type application interface mechanism, the management application program generates a Web document directly. Therefore, the presentation logic on the Web browser is combined with management information production logic in the management application program. The best way of integration is to provide the management application developer with an easy method for Web document generation. As mentioned before, a typical application of the CGI interface mechanism is user command processing such as setting system time, and simple display of the processing result through Web interface. Frequently used functions related to WebMUI are to parse the parameters and to generate Web documents.

POS-EWS offers frequently used functions in the CGI interface in the form of a library. The supported library functions are listed here:

- CgiGetVar: Look variable up in the table of HTTP tags. Return the value of a variable.
- CgiGetCookie: Get value of cookie from the HTTP tags.
- CgiError: Return an error message to the requesting browser.
- CgiHeader: Output the common HTTP header and HTML tags to begin request response.
- CgiWrite: Write formatted data back to the user's browser.
- CgiSetCookie: Issue cookie to the browser.
- CgiFooter: Outputs the standard HTML tags to finish a web page.
- CgiDone: Close the connection to the browser when completing a request.

### 4.2. Web Compiler

In the SSI-Type interface mechanism, while Web documents can be quickly prototyped with readily available desktop authoring tools, the prototype must then be integrated into the system software. POS-EWS works with a fixed set of integrated Web documents that is usually frozen at the time the embedded system is manufactured. Next, it incorporates dynamic information of management application at runtime in such a way of interpretation and execution of the script. Since POS-EWS has no information about the location of script in the HTML file, it scans the whole HTML file to find script tags. It also has to look up the script function table in order to search for the function pointer to be executed.

The POS-EWS Web compiler [3] can offload POS-EWSs scanning of whole HTML file and script table lookup. The POS-EWS Web compiler preprocesses the marked-up HTML file and records the position of tags and its function pointer with the HTML file. At runtime POS-EWS reads and sends the HTML file before the starting point of a tag, continuously calls the script function, and sends the

return result of the called functions, and then proceeds repeatedly with the same procedure for the remaining HTML. POS-EWS does not search for script tags or script functions in HTML file and script function table, respectively. The Web compiler extracts script information from Web pages and generates C code. The C code is then compiled and linked with POS-EWS and the management application program to produce a tightly integrated executable code. The Web compiler 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.

### 4.3. MIB2HTML Compiler

When the SSI SNMP-Type interface mechanism is used for implementing SNMP MIB browser, the MIB2HTML compiler [18] converts MIB definitions into a set of HTML files that are used for SNMP MIB browser. The HTML pages automatically include the marked-up tags used by POS-EWS to display and traverse SNMP information on the basis of the MIB tree structure. HTML files are created for every MIB table. The root file generated contains hyperlinks to all the HTML files for the MIB table. This compiler interprets MIB tables, and automatically detects and fills in enumerated types. It extracts the MIB descriptions and inserts them as online help automatically.

When POS-EWS receives an HTTP request from a browser to display the HTML pages of the MIB browser, it loads the HTML page into memory from the corresponding ROM file. This file is used by POS-EWS to produce the HTML containing up-to-date value of SNMP MIB variable. It does this through the POS-EWS script handler. The handler calls the appropriate function for each markup tag, which is inserted by the MIB2HTML compiler. Once all the marked-up tags have been replaced, the resulting HTML is returned to the browser. This interface mechanism is the same as the SSI-Type, but functions called by the script handler have been made for exchanging management information with the SNMP agent by the use of a local SNMP stack.

The general format for a script is

<!--#*snmp command args*-->

where ***command*** is a command and ***args*** are optional space separated arguments to the command. Here are some examples of scripts:

- <!--#snmp community-->: Display the current community name used for interfacing with the SNMP agent.
- <!--#snmp set oid type -->: Return an HTML FORM structure including button and input text box for setting SNMP MIB variable.
- <!--#snmp get oid -->: Display the up-to-date value of the SNMP MIB variable.

### 4.4. Java SNMP Manager Library

In the Java SNMP-Type interface mechanism, the Java applet is a Web document and SNMP agent is a management application. The Java SNMP interface enables developers to add new functions on top of the SNMP stack in the Java program. The Java SNMP manager sends a SNMP request to the managed element. Strictly speaking, the added function is a subset of the manager role from the viewpoint of network management. The management function of the Java applet manager has limitations in performing complex tasks because the code size of the Java applet must be small so that the code could be downloaded in a reasonable time. Another limitation of the Java applet program is that Java applets are restricted in accessing local disks or executing another program, moreover communicating with the Web server except HTTP by the Java applet security model (sandbox). We used code-signing mechanism for the Java SNMP manager applets to remove those restrictions. Real-time data displays without logging and alarm notifications without history are typical tasks of Java applet manager.

Similar to the CGI interface mechanism, the best method of integration is to provide developers with frequently used library functions when programming the manager in Java. As mentioned before, a typical and suitable application of Java SNMP interface is a real-time data display in a graphic chart or an alarm display with a simple alert. POS-EWS offers a useful library for line chart and alarm display of SNMP variables. An example of implementation is as follows. Java SNMP manager sends a request for interface in/out octets of interfaces group to SNMP agent, and traffic flow graph can be drawn using a library. Like the Java SNMP stack, the code size of this library is so small that the upload time from the Web server is negligible: a total of 30 Kbytes.

## 5. VALIDATION

We developed EWS-WebMUI for a commercial Internet router (Hana Systems' Rustle router 4501 [21]) that had been equipped only with a CLI interface and SNMP agent. At first, we developed and embedded POS-EWS into the router and mapped the CLI interface into EWS-WebMUI. All CLI interfaces were classified into two classes: command processing and state reporting. For the command processing class, we applied the CGI-Type interface mechanism; and for the state reporting class, we used the SSI-Type interface mechanism. The CLI interface management application program was converted to the Web interface management application program for exclusive use by POS-EWS. The text output functions of the CLI interface were converted to formatted write functions using a CGI library provided by POS-EWS, and the input text parsing functions of the CLI interface was converted to Web parameter parsing functions using a CGI library.

By the virtue of the Web compiler, Web interface design for the SSI style

was completely separated from management application development. Bruins's work [22] persuaded us not to map commands to URLs. Each CLI command was mapped to menu item and HTML links in Web pages. After converting the CLI interface to WebMUI, we added an SNMP MIB browser WebMUI without additional programming by use of MIB2HTML compiler and SNMP agent interface library. Only the memory footprint of the code was increased to the extent of storing the generated HTML file. When SNMP MIB-II and a 40-variable private MIB was added, the memory footprint was increased by 40 kbytes. After adding MIB browser, we could perform all management functions to the router through WebMUI. But a real-time display for traffic monitoring and alert action on notification was not supported. So a Java SNMP manager library was used to compensate for that shortcoming.

Figure 4 shows the developed WebMUI examples. Figure 4 (a, b) are a converted WebMUI from CLI by CGI-Type and SSI-Type interface mechanism, respectively; (c) is an example of WebMUI for the MIB browser and (d) shows a real-time traffic monitoring example. Only in embedded environment, we implemented WebMUI and validated the effectiveness of its four interfaces. In a general-purpose computer environment, you must consider other interface mechanisms because you can use Java Virtual Machine, advanced shell tools and advanced script languages, etc.

## 6. CONCLUSIONS

Web servers are already being built into many embedded systems today. In the future, we can expect this trend to grow even further. Embedded Web servers for Web-based network element management provide an administrator with a simple but enhanced and more powerful user interface without additional hardware [4]. However, without effective interface mechanisms and integration mechanisms between management applications and Web documents, these advantages cannot be maximized. In this paper, we have proposed a Web-based network element management architecture having four interface mechanisms with their own unique advantages. Effective integration mechanisms were also introduced for interface mechanisms. Finally, we demonstrated effective use of these application interfaces and integration mechanisms using POS-EWS in a commercial Internet router.

We plan to enhance our proposed architecture so that it supports proactive management and negotiation capabilities. Embedded CORBA [23, 24] may provide an enabling platform for the success of such a management paradigm. Another possible extension to our work is to include a Java server page [25] as another interface mechanism. We are also currently working on extending our POS-EWS so that it can be used not only for element management but also for network management.

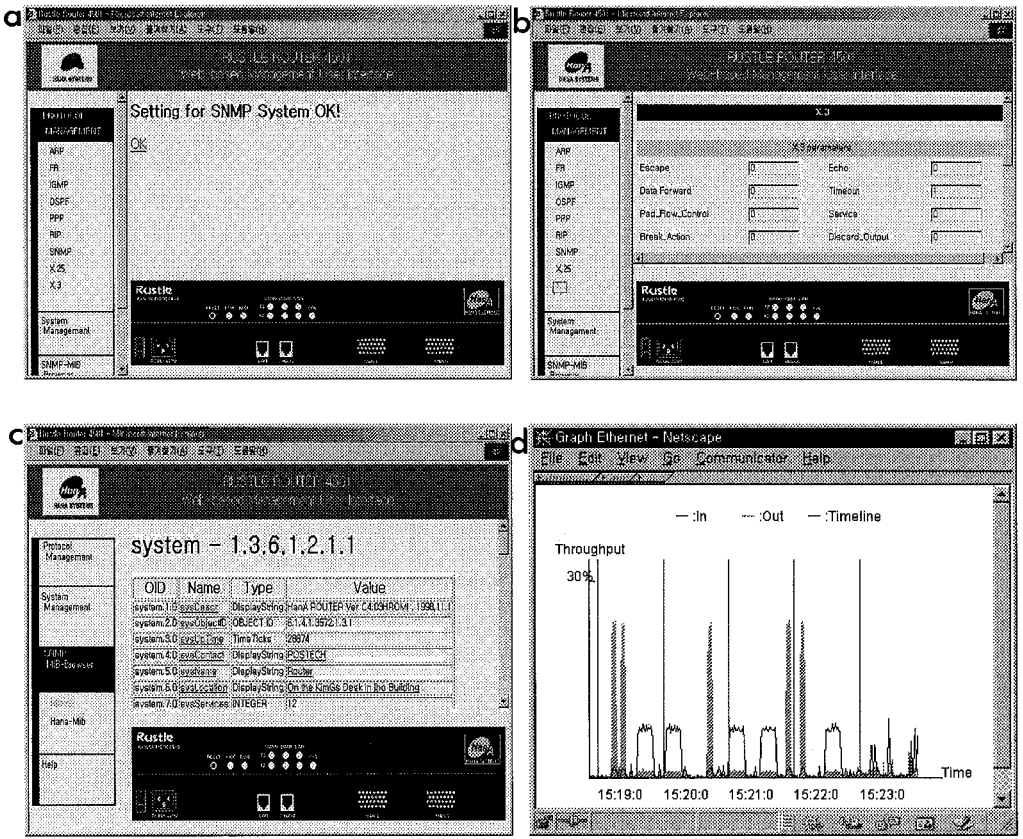


**Fig. 4.** Example of POS-EWS WebMUI.

## ACKNOWLEDGMENTS

The authors would like to thank the Ministry of Education of Korea for its financial support toward the Electrical and Computer Engineering Division at POSTECH through its BK21 program.

## REFERENCES

1. B. McCombie, Embedded Web servers now and in the future, *Real-Time Magazine*, No. 1, pp. 82–83, March 1998.
2. Ian Agranat, Embedded web servers in network devices, *Communication Systems Design*, pp. 30–36, March 1998.
3. H. T. Ju and J. W. Hong, POS-EWS Web compiler for supporting an effective application interface, PIRL-TR-99-003 Technical Report, POSTECH, Korea, December 1999.
4. M. J. Choi, H. T. Ju, H. J. Cha, S. H. Kim, and J. W. Hong, An efficient and lightweight embedded web server for web-based network element management, *Proc. IEEE/IFIP Network Operations and Management Symposium (NOMS 2000)*, Hawaii, pp. 187–200, April 2000.
5. Hong-Taek Ju, Mi-Joung Choi, and James W. Hong, An efficient and lightweight embedded Web server for Web-based network element management, *International Journal of Network Management*, Vol. 10, No. 5, pp. 261–275, September/October 2000.
6. R. Fielding, J. Gettys, J. Mogul, H. Frystyk Nielsen, L. Masinter, P. Leach, andT. Berners-Lee, Hypertext Transfer Protocol—HTTP/1.1, RFC 2616 IETF HTTP WG, June 1999.
7. WBEM, WBEM Initiative, *http://www.dmtf.org/wbem/*.
8. Sun Microsystems Inc., Java Management Extensions (JMX), *http://java.sun.com/products/JavaManagement/*, June 1999.
9. Sun Microsystems Inc., Java Management Programmer's Guide, Developer's Release, June 1996.
10. D. Raggett, A. Le Hors, and I. Jacobs, HTML 4.01 Specification, IETF HTML WG, *http://www.w3.org/TR/html401*, December 24, 1999.
11. CGI, *http://www.w3c.org/cgi*.
12. K. Arnold and J. Gosling, *The Java Programming Language*, Addison-Wesley, 1996.
13. T. Berners-Lee, L. Masinter, and M. McCahill, Uniform Resource Locators (URL), RFC1738, December 1994.
14. The World Wide Web Security FAQ, *http://www.w3c.org/Security/faw/wwwsf8.html*.
15. Ching-Wun Tsai and Ruay-Shiung Chang, SNMP through WWW, *International Journal of Network Management*, Vol. 8, pp. 104–119, 1998.
16. J. Case, M. Fedor, M. Schoffstall, and C. Davin, The Simple Network Management Protocol (SNMP), RFC 1157, May 1990.
17. M. Rose and K. McCloghrie, Structure and Identification of Management Information for TCP/IP based Internets, RFC 1155, May 1990.
18. M. J. Choi, H. T. Ju, and J. W. Hong, MIB2HTML Compiler, Technical Report PIRL-TR-99-002, POSTECH, Korea, December 1999.
19. R. Fielding, Hypertext Transfer Protocol—HTTP/1.0, RFC 1945, IETF HTTP WG, May 1996.
20. Netscape, "An Exploration of Dynamic Documents," *http//home.mcom.com/assist/net_sites/pushpull.html*.
21. Hana Systems Inc., *http://www.hanasys.co.kr*.
22. Barry Bruins, Some experiences with emerging management technologies, *The Simple Times*, Vol. 4, No. 3, pp. 6–8, July, 1996.

23. OMG, The Common Object Request Broker: Architecture and Specification, Revision 2.0, July 1995, OMG TC Document.
24. J. Y. Kong and J. W. Hong, A CORBA-based Management Framework for Distributed Multimedia Services and Applications, Technical Report PIRL-TR-97-1, POSTECH, Korea, March 1997.
25. Sun Microsystems Inc., Java Server Pages, *http://java.sun.com/products/jsp/*, February 2000.

**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 Ph.D. 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 Ph.D. candidate in the Department of Computer Science and Engineering, POSTECH. Her research interests include Web-based network management and policy-based network management.

**James Won-Ki Hong** is an associate professor in the Dept. of Computer Science and Engineering, POSTECH, Pohang, Korea. He received a Ph.D. degree from the University of Waterloo, Canada in 1991 and an M.S. degree from the University of Western Ontario in 1985. Since joining POSTECH in 1995, he has worked on various research projects on network and systems management, with a special interest in Web, Java, and CORBA technologies. His research interests include network and systems management, distributed computing, and multimedia systems. He served as Technical Chair for IEEE CNOM from 1998 to 2000. He was Technical Program co-chair of NOMS 2000 and APNOMS'99. He is a member of IEEE, KICS, KNOM, and KISS.