WEB-BASED CONTROL OF EMBEDDED SYSTEMS

Dipl.-Ing. Christopher Chlap

Faculty of Information Sciences and Engineering
University of Canberra
ACT 2601 Australia

ABSTRACT

Embedded Systems such as Network Attached Storage devices and DSL Routers can often be enhanced in order to perform many useful autonomous computing tasks. This paper describes a method of interacting with and controlling such computations via an application running on a remote PC that serves as both a Graphical User Interface and Controller for applications running on networked embedded systems. Support for complex GUI standards such as X-Windows is not required, nor is a complicated Remote Procedure Call mechanism used. Instead, all interaction between the systems uses the HTTP protocol. Full-duplex and asynchronous interaction between the systems is made possible by running a web server not only on the target system, but also within the control application. The concepts discussed in this paper have been implemented by the author and that sample application will be demonstrated.

KEYWORDS

Embedded Systems, Web Clients and Servers, Ajax.

1. INTRODUCTION

Embedded Systems are widely deployed in homes, offices and businesses the world over. Such systems range in size from small DSL Routers to Digital Set-top Boxes to Network Attached Storage devices fitted with large disk drives. The systems are typically *headless* (i.e. they have no display hardware) and are usually controlled and configured via an external web browser. Power consumption is usually modest (typically 10-20 Watts) and so the systems are generally left on permanently, which makes them good candidates for running services such as personal web servers, print servers and FTP servers.

Most users quickly discover that the vendor-supplied web interface to the system is often very restrictive and that it does not expose all the features and capabilities of the device. Because the most widely used operating system on these devices is Embedded Linux (with freely available source code), many alternative software distributions for such devices have emerged over the years. The main aim of these distributions is to add new functionality and capabilities to the device. Some well-known examples include NSLU2, OpenWRT, DD-WRT and OpenNAS. Examples of the added capabilities include support for Web Servers, FTP Servers, Print Servers, Email Servers, Multimedia Servers (UPnP-based), P2P Servers and many others. In addition, many distributions include support for a range of USB devices, thus allowing the capabilities of the embedded system to be extended even further. Supported USB devices often include Audio, Video and Wireless LAN adapters.

Many of the applications included in the above distributions are console applications (i.e. they are command-line driven) and others have only very rudimentary web interfaces that often only work correctly with one particular web browser. Users would like to interact with such applications via a web interface so that configuration and setup tasks are simplified.

This paper describes a method of interacting with embedded systems in a way that allows functionality to be added to a target system without the need for installing new applications on that system. Once a lightweight web server and a small set of files and scripts have been installed on the target, the controlling system will then be able to execute code on the target and thus perform useful work. Similarly, the target system can also execute code on the controlling system because the control application is also running a web server. This feature allows symmetrical (balanced) interaction between two systems to take place.

2. WEB-BASED SYSTEM ADMINISTRATION

Much work has already been done in the field of web-based system administration, and several useful packages such as Webmin, cPanel and Appweb are widely deployed. Such packages all depend upon a fairly powerful web server being available on the target system. In addition, that web server must support scripting languages such as PHP, Perl or even C#, and a relatively large number of web pages, style sheets and scripts must also be installed. These requirements could be excessive for many types of embedded systems, either because the needed scripting languages are not available or because of insufficient RAM and CPU resources.

The above web administration packages have a number of significant shortcomings. Firstly, interaction is only possible for the duration of a web "session". In other words, all interaction ceases as soon as the user browses to another location. This means that error messages and status messages generated on the administered system must be sent to the user by other means, typically via email or via a SysLog daemon. In contrast, the control application described in this paper is running an embedded web server of its own, and so the target system can send error and status messages to the control application at any time. If desired, it can even be arranged that severe errors cause a window to pop up on the administrator's desktop.

A second major shortcoming of the above packages is that they generally do not allow the installation of custom web pages on the server being administered. One is generally restricted to using only what a system administrator has previously installed on the system. While this is definitely a prudent policy for large, Internet-connected web servers, it is generally not an appropriate policy for embedded systems connected to small, private networks with a trusted user base. In those cases, more flexibility is required and the control application described in this paper allows additional web pages and scripts to be transferred to the target system very easily.

3. SYMMETRICAL WEB COMMUNICATIONS

The classical model for web-based communications is the well-known Client-Server Model, first proposed in the early 1980's. Since then, many other models of computer communications have been proposed including the Peer-to-Peer model and various other hybrid variants.

The model described in this paper differs from the Client-Server model because the control application has both a client component (in the form of an embedded web browser) and a server component (in the form of an embedded web server). The control application's client can therefore communicate with a server running on the target system, and code executed on the target system can just as easily communicate with the control application's server at any time.

In addition, the server embedded in the control application has unrestricted access to the full functionality of the control application itself, including a large range of system services provided by the underlying operating system. This simple feature has an important consequence: An embedded system can request that certain operations that it itself cannot handle (because of resource constraints) be executed on the system of the control application. One example of this might be file storage: An embedded system such as a DSL Router (that has no secondary storage) could request file storage services from the control application.

In addition, because of the lightweight protocol used in this scenario, it may even make sense for the embedded system to offload the execution of many CPU and memory intensive operations to the control application's system.

4. THE CONTROL MODEL

The following diagram illustrates the major components of the control application and the target system, and the main paths of interaction between those components.

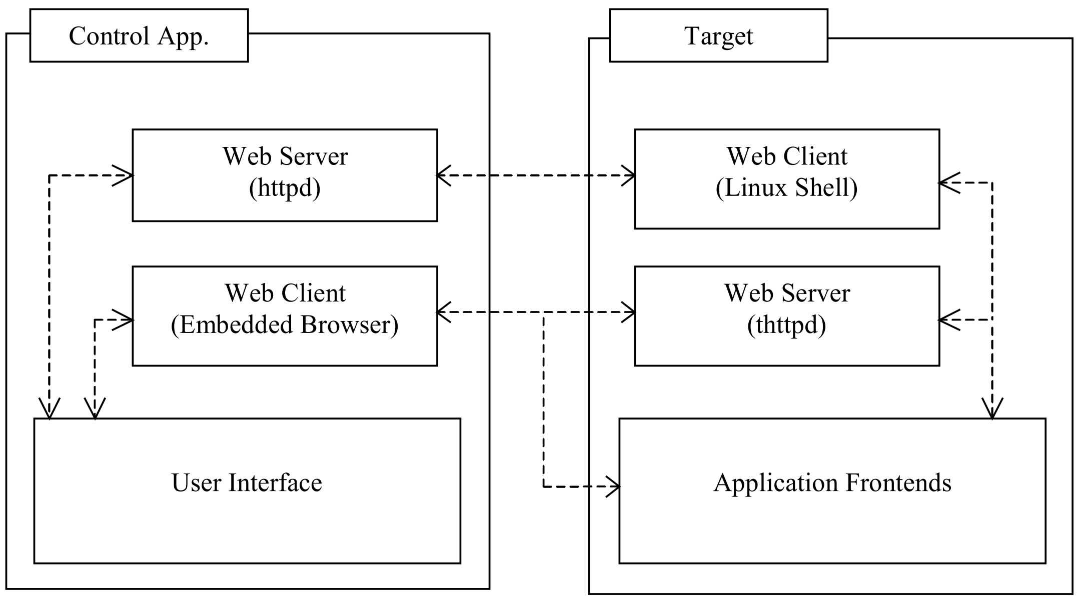

Figure 4.1 The control model

The above diagram shows that each system contains both client and server components. This allows for balanced communications between the systems without the overhead of having to hold open one or more TCP connections.

The control application contains an embedded web browser that serves as the interface between the user and the application. It also contains an embedded web server that is tightly coupled to the application and can therefore access all functionality contained within the application. The embedded web server is multithreaded, as is the entire control application.

The target system contains two major components of interest to us. The first is the web server, which can be any lightweight server with CGI Scripting capability (e.g. thtpd). The second is a client component that is able to communicate with the control application's web server. For Linux systems, the client component is actually just a set of shell commands and scripts that can be executed either by the web server or (in the background) by other applications running on the system.

Figure 4.1 also shows a module consisting of application front-ends. These are pre-supplied interface modules that come with some applications and users usually connect to them via a web browser in order to interact with the application. While our control application can connect to such front-ends via its embedded web browser, they are not an essential component of our control model.

5. THE CONTROL APPLICATION

The Control Application (WEB32) consists of a multithreaded WIN32 application that is structured as a small operating system very similar to Douglas Comer's XINU Operating System described in [1]. All of the usual operating system components such as memory manager, thread manager, synchronization primitives, device I/O, network I/O and command interpreter have been implemented. On Windows platforms, these components make use of the WIN32 API, but the code can easily be ported to other platforms where, for example, POSIX primitives would be used.

WEB32 supports several window devices, and each of them can contain both an embedded web browser and a character mode interface to a command interpreter (shell). The shell provides the fundamental interface to the application and it is able to execute a rich set of built-in commands as well as TCL scripts.

For network communications, WEB32 can be configured to use either the Winsock API or a custom device driver and a TCP/IP stack described in [2]. When the custom stack is used, WEB32 can run servers that can listen at IP addresses other than those of the Windows TCP/IP stack.

The WEB32 source code is available from the author on request:

http://www.nat32.com/message.htm

A software router application use the same code base as WEB32 is available from:

http://www.nat32.com/download2/nat32v2.zip

5.1 The WEB32 Shell

The WEB32 shell supports the concurrent execution of commands, and also includes full support for command pipelines and redirection. Commands are executed concurrently as separate threads, and each thread inherits the standard I/O devices of its parent. The shell normally blocks until all child threads have terminated, but this behavior can be overridden with a background operator. Command pipelines allow multiple commands to be executed concurrently. Redirection can be to any device, whereby WEB32 treats files, windows, pipes and network connections as devices. Finally, a WEB32 command is defined as either a built-in shell command, a file containing one or more shell commands (recursion permitted), a TCL script or a URL.

The WEB32 application can create display windows for user interaction. A shell normally runs with its standard I/O devices attached to such a window. The client area of each window can be either a terminal style Console, or a Web Browser Control. Note that both the Console and the Web Browser Control are classified as devices, and can thus be the target of an I/O redirection.

5.2 The WEB32 Web Server

The WEB32 application also runs a multithreaded web server (HTTPD) that understands GET requests and POST requests issued by any standard web browser. The HTTPD supports WEB32 command execution via GET requests of the following format:

```
GET /shell?cmd=command+arg1+arg2+... http/1.1
```

The above format matches that of HTTP GET requests generated by HTML Forms, and such requests can therefore be easily generated within an appropriate web page. Note that it is this command execution feature that lends the WEB32 Control Application its power and flexibility.

Some example commands (and their associated URLs) are listed below:

- Print the WEB32 Thread Table:

  http://localhost:8080/shell?cmd=ps

- Print the WEB32 Thread Table as an Ajax (auto-refresh) page:

  http://localhost:8080/shell?cmd=xps

- Run an external program on the host system:

  http://localhost:8080/shell?cmd=exec+name+arg

- Display a remote Samba front end (e.g. SWAT) in a WEB32 browser window:

  http://nas.box:901/

Many more commands have been implemented (approx. 400 in total), and as the WEB32 sources are freely available [3], the command set can be extended by C programmers. Functionality can also be extended by writing HTML code that uses the existing WEB32 commands, and by writing TCL scripts to perform string manipulation and socket-based communications.

5.3 The WEB32 Web Client

The WEB32 web client is implemented as an embedded web browser object (either Internet Explorer or Mozilla) that serves as a graphical user interface to the application. This approach has the advantage that the interface can easily be customized and expanded simply by providing appropriate HTML files, script files and style sheets. Each WEB32 window can contain its own, independent browser object.

An embedded browser is not subject to many of the restrictions that apply to web browsers intended for use in the Internet environment. This is because the browser fetches its pages either directly from the hosting application or from the web server embedded within the application. The application can exercise fine-grained control over all aspects of browser functionality including navigation, cookies, agent strings, headers, file downloads, script execution, application of style sheets and embedding of additional objects. In addition, debugging, popup blocking and selective script execution can be performed.

As an example of WEB32's control over the embedded browser, let us assume that the browser has just loaded a page that contains a "layered advertisement". Such "layer ads" are **not** implemented as browser windows, and so most popup blockers will fail to detect them. WEB32 will not only detect them, but it can also determine which Javascript function within the page is used to close the popup. It can then immediately call that function and close the popup before it has a chance to waste bandwidth downloading its content.

Interestingly, because of its powerful debugging and monitoring capabilities, some WEB32 users are using the program solely as a tool for web page development and Javascript testing. Others are using it for the detection and analysis of malicious web sites.

6. THE EMBEDDED SYSTEM

The hardware requirements of an embedded system that is to be controlled by WEB32 are very modest. The system does not require large amounts of RAM or a fast CPU, nor does it require secondary storage. It must be running a web server that can at least execute CGI scripts and that honors HTTP POST requests. It must also support an operating system shell and a small set of basic shell commands. Additional commands will be installed on the system as needed.

Once a suitable web server has been installed and configured, a small set of shell scripts and HTML files must be placed in the path of the web server. It is these shell scripts and web pages that the controlling system (WEB32) will execute in order to provide much of the control and management functionality.

6.1 The System Shell

The embedded system is controlled and interrogated by means of web pages containing specially crafted URLs that are fetched from the target's web server by the control application's web browser. The embedded system's shell will most likely support a BusyBox command set, but scripts for other command sets could also be written. Note that the shell commands will return data over the TCP connection on which they were invoked, and the WEB32 browser component will then encapsulate that data in pleasantly arranged web pages for the user's enjoyment.

Some sample commands and their associated URLs are listed below:

- Print the process list:

  http://nas.box/shell?cmd=ps

- Print the process list as an AJAX (auto-refresh) page:

  http://nas.box/shell?cmd=xps

- Run a local program:

  http://nas.box/shell?cmd=exec+name+arg

- Run a remote program:

  http://web32.box/shell?cmd=exec+name+arg

- Fetch a file from a remote web site:

  http://nas.box/shell?cmd=fetch+url

Many more commands exist, and extending the command set is relatively easy for a shell programmer. Functionality can also be extended by writing HTML code that makes use of existing scripts and shell commands, and by executing scripts on the WEB32 server to perform tasks that cannot be done on the embedded system because of resource constraints.

6.2 The Web Server

As stated above, the embedded system must be running a web server that can at least execute CGI scripts and that honors HTTP POST requests. Full-featured web servers such as Apache will also work, but are not required. A suitable server that is commonly available on NAS devices is thttpd. In addition, a few basic shell scripts need to be installed in the path of that web server. No web pages (html files) are required initially, as the needed pages can be fetched later by executing a shell script that downloads files from the control application's web server. Alternatively, if the embedded system is running either Samba or an FTP server, the needed web pages, script files and style sheets can also be transferred to the device by such means.

7. WEB32 IMPLEMENTATION DETAILS

The WEB32 application was written entirely in C and consists of around 100,000 lines of code. This seemingly large amount of code is a result of the maturity of the project. As most programmers know only too well, an application can always be extended and enhanced, no matter how mature it is. At run time, WEB32 requires around 1 MB of memory and very little CPU time.

7.1 Web Pages

All of the needed web pages for an interface to an embedded system were handcrafted in HTML, Javascript, and CSS. No integrated web development environments were used. The described concepts were tested with a NAS device that was running OpenNAS Version 1.7. Several shell scripts for execution on the NAS device were written, as well as several HTML pages that served as the user interface. Emphasis was placed on functionality, not appearance, and so a production version would most likely have a quite different look and feel.

7.2 Screenshot

The following WEB32 screenshot (see Fig. 7.1) conveys an impression of one particular style of user interface currently available. It shows a command tree in the left panel and an Explorer window in the right panel. Toolbars can be located in both the client area of the WEB32 window and also within the web page currently being displayed by the browser object.

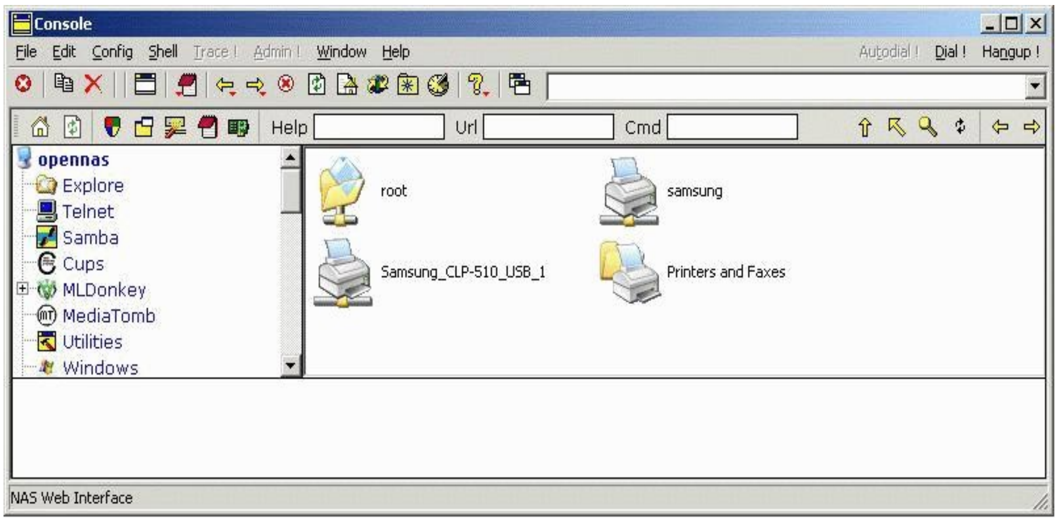

Figure 7.1 WEB32 Screenshot

8. CONCLUSION

The WEB32 application has been under development for several years, and it has served as a sample system for students of Systems Programming and Networking subjects at the University of Canberra. Only recently has the software been applied to the control of embedded systems such as DSL Routers and Network Attached Storage devices.

WEB32's symmetrical communications model has the following advantages:

- No TCP connections need to be held open indefinitely in order to allow asynchronous interaction between a control application and an embedded system.
- Each system can use Ajax methods to retrieve status information from the other system at any time.
- Each system can make use of functionality residing within the other system.
- Each system can transfer enhancements to the other system, thereby extending its functionality and modifying its user interface.
- The embedded system can update the contents of any window belonging to the control application thus allowing such windows to act as displays for the headless device.
- The control application's web server allows custom web pages to be generated that contain selected content fetched from external sites.

The model does have a number of limitations:

- A custom control application is required.
- The control application must be able to embed a web browser object.
- A web server must be integrated within the control application.
- That web server must allow external systems to execute code located within the control application.

In its present form, the WEB32 application is not suitable for use by computer novices. By its very nature, it lends itself more to expert users and computing students who wish to study web principles and conduct experiments in the exciting world of the Internet.

ACKNOWLEDGEMENT

Many thanks go to the staff and students of the University of Canberra for the invaluable feedback and assistance given during the development and testing phases of this project.

REFERENCES

Many of the ideas implemented in the WEB32 application were inspired by the following books written by Douglas Comer et al. of Purdue University.

[1] Douglas E. Comer and Timothy V. Fossum, 1988. *Operating System Design Volume 1.PC-Edition* Prentice-Hall, New Jersey, USA.

[2] Douglas E. Comer and David L. Stevens, 1999. *Internetworking with TCP/IP Vol. 2 3rd Edition* Prentice-Hall, New Jersey, USA.