

EMBEDDED WEB TECHNOLOGY: APPLYING WORLD WIDE WEB STANDARDS TO EMBEDDED SYSTEMS

Joseph G. Ponyik and David W. York
National Aeronautics and Space Administration
Glenn Research Center
Cleveland, Ohio 44135

ABSTRACT

Embedded Systems have traditionally been developed in a highly customized manner. The user interface hardware and software along with the interface to the embedded system are typically unique to the system for which they are built, resulting in extra cost to the system in terms of development time and maintenance effort.

World Wide Web standards have been developed in the passed ten years with the goal of allowing servers and clients to interoperate seamlessly. The client and server systems can consist of differing hardware and software platforms but the World Wide Web standards allow them to interface without knowing about the details of system at the other end of the interface.

Embedded Web Technology is the merging of Embedded Systems with the World Wide Web. Embedded Web Technology decreases the cost of developing and maintaining the user interface by allowing the user to interface to the embedded system through a web browser running on a standard personal computer. Embedded Web Technology can also be used to simplify an Embedded System's internal network.

TERMS AND DEFINITIONS

To establish a common base of understanding, the following definitions will be used:

Client—A logical entity that initiates a request for data or for an action to take place. A client depends upon the presence of an associated server to perform requests. A client may refer to client software, client hardware, or a combination of the two to implement a logical client.

Server—A complementary logical entity to a client. A server listens for client requests and services those requests, whether the request is for data or for an action to be performed. A server may refer to server

software, server hardware, or a combination of the two used to implement a logical server.

Web client—A client that is designed to communicate with servers using the Hypertext Transfer Protocol.

Web server—A server that is designed to communicate with clients using the Hypertext Transfer Protocol.

EMBEDDED WEB TECHNOLOGY INTRODUCTION

Embedded Web Technology was developed for the Fluids and Combustion Facility (FCF) of the International Space Station (ISS). FCF is being developed to perform investigations in combustion science and fluids physics in the microgravity environment of the ISS. One of the goals of the FCF is to be able to perform at least 10 investigations in each discipline per year for 10 to 15 years. In order to meet this goal, it is important for the software to be adaptable to changing requirements. One of the areas of concern is the user interface software that the ISS astronauts will use to operate the FCF. The challenge facing the FCF software engineers is that the laptop computer that the user interface software will operate on is supplied by the ISS, not FCF. In the event that the ISS decides to upgrade the laptop computer, the FCF software team will be required to modify existing user interface code for a new environment while still developing new code for possibly the old laptop computer and also the new laptop computer. The FCF software engineers are also faced with the task of developing a system that would accommodate unknown experiments. This scenario can be extended to any system where the users, hardware and applications are unknown and the system is expected to have along life.

The FCF software engineers realized that the World Wide Web had already solved a similar problem. With the World Wide Web, a person uses a web browser to request a web page from a web server. In this scenario, the web server and the web browser interface is independent of the hardware and software being utilized at the other end of the interface. Despite

Copyright © 2001 by the American Institute of Aeronautics and Astronautics, Inc. No copyright is asserted in the United States under Title 17, U.S. Code. The U.S. Government has a royalty-free license to exercise all rights under the copyright claimed herein for Governmental Purposes. All other rights are reserved by the copyright owner.

this, the web page is successfully transmitted to the web browser and properly displayed.

The World Wide Web is based on the Hypertext Transfer Protocol (HTTP), the protocol used by web servers and web browsers to communicate. The FCF software engineers did an extensive search to find an HTTP compliant web server that would fit the requirements of FCF. FCF, being an embedded, real-time system, would require the web server to be small, operate under VxWorks®, and still allow the system to meet its real-time requirements. The search failed to find such a web server.

The FCF software engineering team decided to write their own web server and this proved successful. The web server, known as Tempest, is HTTP compliant. It implements two of the seven request methods defined in the HTTP specification, GET and HEAD. The GET method is a request by a web browser for a file from the web server. The HEAD method only requests header information. These two methods are the only two required to be implemented to make a web server HTTP compliant and also offer a degree of security to the system by not allowing the web server to accept a file from a web browser.

Tempest also met the other needs of FCF. It is small, requiring less than 50K of memory in its minimal configuration, does not take up a lot of disk space, and has minimal impact on system performance. It is active only during the times that a web browser is requesting a file. It can operate at a low priority with adequate response time for the user so that it does not interfere with the real-time aspect of the system.

The successful implementation of Tempest made it apparent that there are advantages to utilizing other World Wide Web standards in an embedded system. These standards allow embedded system projects to take advantage of work being done by thousands of developers, thus reducing the problems inherent in developing and utilizing custom protocols.

The term “Embedded Web Technology” was the name given to this merger of embedded systems with World Wide Web Technology.

OVERVIEW OF THE WORLD WIDE WEB

The World Wide Web is a collection of protocol standards that are controlled by the World Wide Web Consortium®. The protocol standards promote evolution of the World Wide Web and ensure its interoperability.

The key protocol standard behind the World Wide Web is the Hypertext Transfer Protocol, HTTP, specified in RFC 2616. HTTP is “an application-level protocol for distributed, collaborative, hypermedia information systems.” (RFC 2616).

HTTP is used to transfer information between a web server and a web client, which is typically a web

browser such as Netscape. A typical web client request of a service from a web server will consist of a GET request. The web server responds to the GET request from the web client by transmitting the requested information, typically an electronic file.

The following is an example of the HTTP messages that are exchanged when a web page gets transferred from a web server to a web client. First, the request from the client:

```
GET /index.html HTTP/1.0
Connection: Keep-Alive
User-Agent: Mozilla/4.7 [en] (WinNT; U)
Host: jgp6290.grc.nasa.gov
Accept: image/gif, image/x-xbitmap, image/jpeg,
image/pjpeg, image/png, */*
Accept-Encoding: gzip
Accept-Language: en
Accept-Charset: iso-8859-1,*,utf-8
```

The first line tells the server that this client wants to get index.html and it is following version 1.0 of the HTTP. The second line, Connection, requests that the connection be left open. The third line, User Agent, describes the web client making the request. The fourth line, Host, is the IP address of the web client. The last four lines describe the types of messages the web client can receive.

The response from the web server is the following:

```
HTTP/1.1 200 OK
Host: jgp6290/139.88.219.70
Date: Mon, 020 Aug 2001 17:27:31 GMT
Server: TempestJava 1.2 (NASA/GRC Java Version of
Tempest)
Connection: Close
Content-Length: 293 293
Content-Type: text/html
```

The first line tells the web client that the web server understands HTTP 1.1 and that the request from the client is allowed. The second line, Host, is the IP address of the web server. The third line, Date, time tags the message. The fourth line, Server, identifies the web server software that is responding to the request. The fifth line, Connection, tells the web client that the web server will be closing this connection upon completion of this transaction. The sixth line, Content-Length, tells the web client how many characters to expect in the file to be sent. The last line, Content-Type, tells the web client that the data will be in text. After this line, there will be a blank line and then the file requested, index.html, will be sent to the web browser.

In the event that the web page sent to the web client contains references to image files, applets, etc.,

the web browser will make additional requests to the web server for these resources. Each one of these resources will require a full transaction similar to what was just described.

There are many additional items that can be included in a transaction that are described in RFC 2616. For a real-time, embedded system where resources such as memory are limited, it is not necessary to implement the entire protocol.

COMPARISON OF A TYPICAL WEB SERVER ENVIRONMENT VERSUS AN EMBEDDED, REAL-TIME ENVIRONMENT

A typical web server runs on a computer that is running a nonrealtime operating system. The web server software itself tends to be large and complex and requires a lot of memory and disk space, on the order of 10 MB or more. The web servers are designed to run on personal computers and general purpose workstations.

The embedded, real-time environment is much more restricted. The system generally consists of an embedded processor running a real-time operating system and also running a real-time application. Embedded systems have limited memory and disk space that is not easily extensible.

The users of these two environments have different needs of the systems. The user of the nonrealtime environment is usually interested in obtaining static or slowly changing information. This information is stored in files and retrieved by the web browser for display by the user. The user can read the information on the display at his own pace or print it out and read it without the aid of the browser. A user of a real-time system, on the other hand, is interested in obtaining the most current data from the system at regular intervals. The user may require the capability to issue commands to the embedded system.

PUTTING EMBEDDED SYSTEMS ON THE WEB

There are two problems that need to be solved in order for an embedded system to become accessible on the World Wide Web. One is providing a user interface that provides the real-time interaction needed by the user in order to properly interface to the system. The other is to give the embedded system the capability to serve web pages over the World Wide Web.

One feature of the web browser that helped bridge the gap between the two environments is the addition of Java™ applets. Applets are programs that are capable of being executed by a web browser. They are written in the Java™ language, compiled and then stored on the web server's computer. When a web page is transmitted to a web browser, the web browser scans through the web page, looking for, among other things, applet tags.

When an applet tag is found, the web browser automatically makes another request to the web server for the applet. The applet gets sent to the web browser which in turn loads the Java™ Virtual Machine which starts running the applet.

In the typical web environment, applets provide an interface that is dynamic but usually does not interact with the web server's computer. For security purposes, applets are very restricted in what they can do in a web browser environment. It is possible to bypass these restrictions with security certificates if they are an impediment.

One capability an applet has by default is the ability to communicate back to the computer that served it to the web browser. This communication can be accomplished with basic socket technology, Java's™ Remote Method Invocation (RMI), Common Object Request Broker Architecture (CORBA®) technology, or other protocols. RMI is a communication technology specific to Java™ that allows networked Java™ programs to interface in a platform independent manner. CORBA®, developed by the Object Management Group™, is a technology that allows networked programs to communicate in a common manner that is independent of the underlying hardware, operating system or language. With this capability, it is possible to develop an applet that can interact dynamically with an embedded, real-time system and, thus, let the web browser provide a user interface that meets the needs of the user.

On the embedded system side, the problem is providing the capability to be a web server without overburdening the system with all of the functionality specified in the HTTP specification. The embedded system software still needs to be able to perform real-time command and control. This problem was solved by the development of Tempest.

TEMPEST FEATURES

The Tempest software, which was written by software engineers at the NASA Glenn Research Center in Cleveland, Ohio, is a web server written specifically for embedded, real-time systems. Tempest was originally written for the VxWorks® operating system from Wind Rivers Systems, Inc. and then ported to the Java™ language so that it can run on any operating system that has a Java™ Virtual Machine.

Tempest requires fewer memory resources than web servers written for the typical web server environment. Memory requirements are under 100 KB, depending on how it is configured. The amount of disk space is also under 100 KB with additional space needed for the files that make up the web pages, images and applets. Since Tempest is not intended to operate as a general purpose web server, it is not necessary to

implement the entire HTTP specification. Only the GET and HEAD request methods from the HTTP specification are implemented. Methods that allow a web browser to write to the web server are not implemented. The responses generated by Tempest are also limited to those that an embedded system would need.

Tempest can also be run at a lower priority than other application software running in the embedded system. Requests from web browser are very brief so Tempest can serve web pages and other resources at an acceptable speed without having an impact on system performance.

An optional capability requires the user to have an identification and password in order to gain access to the system. This provides a limited level of security to the system. Other security features such as firewalls and virtual private networks can be added without changing the embedded system. It is much easier to have the security features added on as separate entities rather than built into the system. This allows for easier upgrades to the security system and decouples the security from the embedded system.

The user I.D.'s and passwords are stored in an external file. New users can be added to the system without having to recompile Tempest.

A configuration file that allows Tempest to associate a user with a specific image file is another feature. This allows a user to set up the system so that when a remote user gains access to the embedded system, the web page can be customized to that user on the fly by displaying an image file created for that specific user.

Tempest also has configuration files that allow the user to specify which remote clients have access to the embedded system. Tempest utilizes a configuration file to maintain a list of MIME types, used when responding to a request to assist the web browser in determining the type of data is being received.

All of the configuration files are read in by Tempest when it starts up. Tempest first reads in a file called tempest.sys that contains a list of the configuration files. Updates to any of the configuration files require the system to be restarted before the changes take effect.

Web browsers are denied access to any file with a ".sys" extension. If the configuration files use this extension, it is not possible for the person using the web browser to have them displayed.

Tempest will not process any request that contains "..". The ".." (double dot) is used on most computer systems to refer to the parent directory on the disk. By not allowing this, it is not possible for remote users to snoop around the system.

Tempest has a feature known as Server Side Includes (SSI). SSI is the ability of the web server to

dynamically alter a web page at the time of request. Tempest accomplishes this by reading through any file the has an extension of ".sht", ".shtm" or ".shtml" and searching for <Tempest> tags. These tags are a unique feature of Tempest. When Tempest encounters one of these tags, it processes the contents of the tag and substitutes the tag with the result of the processing.

One of the tags is <Tempest image>. When this tag is utilized in combination with the user I.D. and password, Tempest is able to associate the user to an image file and substitute in new html that will contain a tag to an image file specific to that user. The images.sys and users.sys configuration files need to be coordinated for this to work.

The other tag is <Tempest execute=somecmd param>. This tag causes Tempest to execute the command specified in commands.sys that corresponds to somecmd. The parameters to the command are passed to the command. The resultant output from executing the command is inserted into the web page. Error messages are displayed if somecmd is not found in commands.sys or the corresponding command is not found.

The commands may be either commands that are built into the operating system or commands written by the developing team. This feature can be very useful during the development stage as a debugging aid. The output from various commands made to the embedded system can be displayed in a web page. For example, task information, network statistics, etc., can be retrieved in real-time and be monitored from anywhere by system engineers.

The following is an example of how tasking information can be displayed. Here is the text from a file called taskshow2.shtml:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML
3.2//EN">
<HTML>
<HEAD>
<TITLE>VxWorks Task Information</TITLE>
<META HTTP-EQUIV="Author"
CONTENT="Joseph. G. Ponyik">
<META HTTP-EQUIV="ReplyTo"
CONTENT="Joseph.G.Ponyik@grc.nasa.gov">
</HEAD>
<BODY BGCOLOR="Aqua" TEXT="Black"
LINK="Blue" ALINK="Red" VLINK="Purple" >
<H1 ALIGN=CENTER><B>VxWorks Operating
System Task Information</B></H1>

<P><PRE>
<TEMPEST EXECUTE=task 0 2>
</PRE></P>
<HR WIDTH="50%">
```

<P><I>VxWorks </I>® is a registered trademark of Wind River Systems, Inc.</P>
 </BODY>
 </HTML>

When this file is requested by a web browser, Tempest will scan the file and find the line that reads <TEMPEST EXECUTE=task 0 2>. Tempest will then match “task” against the contents of commands.sys. Since this is a VxWorks® implementation, Tempest will execute the taskShow command with the first two parameters being 0 and 2. Tempest will then replace the line with the results of taskShow. The resultant output in a web browser will be similar to:

VxWorks® Operating System Task Information

NAME	ENTRY	TID	PRI	STATUS	PC	SP	ERRNO	DELAY
tExcTask	_excTask	3e8a70	0	PEND	7b7fe	3e89cc	0	0
tLogTask	_logTask	3e615c	0	PEND	7b7fe	3e60b4	0	0
tShell	_shell	3ade1c	1	PEND	597f4	3adacc	1c0001	0
tTelnetd	_telnetd	3c25c8	2	PEND	597f4	3c24dc	0	0
tWdbTask	58802	3af23c	3	PEND	597f4	3af0d8	0	0
tScsiTask	_scsiMgr	3e31c8	5	PEND	597f4	3e315c	0	0
tNetTask	_netTask	3dda6c	50	PEND	597f4	3dda14	0	0
tFtpdTask	_ftpdTask	3bfb70	55	PEND	597f4	3bfa8c	0	0
tTftpdTask	_tftpdTask	3bc9b8	55	PEND	597f4	3bc260	0	0
mon_tempest	fb06e	392940	60	DELAY	2cafa	3928fc	0	5
http2	f9a5a	39917c	75	DELAY	2cafa	398dd4	1c0001	1
t7	_taskShow	37d0f0	76	READY	2f12e	37c360	0	0
tPortmapd	_portmapd	3c109c	100	PEND	597f4	3c0f54	16	0
tempest	f9a5a	38b3a8	100	PEND	597f4	38b2a4	0	0

VxWorks® is a registered trademark of Wind River Systems, Inc.

The Java™ version of Tempest has features similar to the VxWorks® version. All of the configuration files except for commands.sys will work with both versions. Commands.sys needs to be modified to include commands that will work on the platform that Tempest will be operating on. The <Tempest execute> tag also includes a Wait=true|false|yes|no parameter and a Message=”some message” parameter. The Wait parameter indicates if Tempest should wait (yes or true) for the command to finish execution. The Message will be inserted into the web page if Wait is false or no.

The Java™ version of Tempest also includes a tag to allow Tempest to call another Java™ class external to Tempest and insert the resultant output into the web page. The format is <Tempest Object=o Method=m Args=a> where o is the class to be run, m is the method to call and a are the arguments to be supplied in the method call.

THE RESULTING IMPLEMENTATION

Utilizing Tempest, it is now possible to have an embedded, real-time system appear as a node on the World Wide Web. To the remote user, the embedded system appears as a World Wide Web node. The remote user simply needs a computer with a web browser capable of running a Java™ applet. The remote user only needs to enter the uniform resource locator (URL) of the embedded system into the browser.

The embedded system needs to be up and running. Minimally, it needs to have Tempest running and an application to interface to the system and handle requests from the user interface.

The web browser sends an HTTP message to the embedded system, requesting the web page. If the user needs to enter a user I.D. and password, Tempest

responds with a request for the user to be authenticated. Once the user is authorized, Tempest retrieves the web page from local storage. If the file has an extension of “.sht”, “.shtm”, or “.shtml”, Tempest reads through the file and processes any Tempest tags. When this step is complete, the web page is delivered to the web browser.

The web browser displays the web page and also requests any additional resources from the embedded system. These resources may include images and an applet. When the applet gets sent to the browser, the browser starts up its Java™ Virtual Machine which starts up the applet.

The applet establishes a connection back to the embedded system, utilizing a different TCP port from the one being used by Tempest, which is typically 80. While the connection is being established, the applet also begins the user interface. Since Java™ supports multitasking, the applet should be designed so that the

user interface screens run in a separate task from the interface to the embedded system. This gives a smoother running interface and also makes it easier to recover from communication drops without locking the user out.

By taking advantage of the CORBA[®] technology, more flexibility is added to the system. CORBA[®] provides an interface that is independent of the underlying hardware platform, operating system and implementing language. Future upgrades to the embedded system or the web browser will be easier to implement since the change will be transparent to the other end of the interface.

OTHER ADVANTAGES

In addition to the above features, utilizing Embedded Web Technology also provides an added security feature. Since the web browser does not store the applet permanently, the user interface software is not accessible to unauthorized users once the web browser is turned off. It is possible to configure present day web browsers to eliminate the cache so that the applet gets deleted.

User interface software upgrades are simpler with Embedded Web Technology. Without Embedded Web Technology, the user of the real-time system also needs to have the user interface software stored locally. The result is that the user is restricted to using only the computer that has the user interface software loaded and upgrades to the software can be more difficult to obtain. The updates require the user to find out about the upgrade and then a process needs to be put in place to deliver the upgraded software to the user. The upgraded software may also need to be capable of running on various platforms, making upgrades more difficult for the developer.

With Embedded Web Technology, the user interface software is stored in the embedded system and delivered to the user when it is needed. When the user interface software is upgraded, it only needs to be stored in the embedded system. The user gets the new software the next time they access the system.

Tempest also provides the capability to provide output to the remote user in any format the system designer desires. Although the Tempest tag is typically embedded into a web page that is html, it is possible to set up a file that consists of only a Tempest tag calling a local command that outputs something other than html, such as XML. As an example, Tempest could be used to feed real-time data into a database using XML.

SUMMARY

Embedded Web Technology provides for the development of an embedded, real-time system that appears to the users of the system as a node on the World Wide Web. This capability provides for great savings by eliminating the need to develop and distribute user interface software that is platform specific and somewhat cumbersome to configuration manage.

Tempest software is a small, flexible web server that makes it easy to interface to embedded systems. It also has the potential to be an aid in debugging systems.

Tempest workshops have been held for customers in the private sector. Customer remote data acquisition and control applications include medical, telecommunications, aerospace, factory automation, instrumentation, automotive, building management and education.

Tempest is available from the NASA Glenn Research Center by contacting the Commercial Technology Office at 216-433-3484.

REFERENCES

RFC 2616: Hypertext Transfer Protocol—HTTP/1.1, <http://www.w3.org/Protocols/rfc2616/rfc2616.html>
World Wide Web Consortium[®]—<http://www.w3.org/>
XML—<http://www.w3.org/XML/>
CORBA[®] and Object Management Group[™]—<http://www.omg.org/>
Java[™]—<http://java.sun.com/>