A Framework for Building Network Efficient HTML Based User Interface

Tanmay K. Mohapatra

HTML is a very versatile platform for creation of user interfaces. However, complex HTML screens could be heavy on the network in terms of data size and require excess processing power on the servers as well. Designing dynamic HTML pages through scripting languages (like JSP and ASP) often results in mingling of user interface and business logic which is difficult to develop and maintain. Clever use of DHTML can result in tremendous benefits in network bandwidth and at the same time help isolate the user interface from business logic.

There are various reasons why HTML is popular as the ultimate flexible user interface solution. It provides for an easy learning curve, it is easy to create, it is platform independent, and a multitude of browsers are already available for displaying them. Complex widgets and screen behavior can be achieved by embedding custom components (ActiveX/Applets) into HTML. Therefore HTML is increasingly becoming the choice for developing user interface even for large business applications including those used for on-line transaction processing (OLTP).

Complex HTML screens can be bulky. Large OLTP applications with big screens suffer the most because of this as the screens are dynamic and can not be cached by the browser. Quite some research has been done to change HTTP at the protocol level to optimize network usage and reduce wait time/latency. However, most of these are only at experimental stage. Though there are tools and technologies available to compress HTTP traffic, compression is a drain on computing resources.

The most efficient way would be to reduce the data traffic at the source itself. In this article, I propose a framework for developing OLTP applications that can have rich HTML based user interface while still minimizing network traffic and reducing processing and data compression overheads.

NEED FOR THIS FRAMEWORK

There are two primary requirements this framework addresses – reduction of network bandwidth requirements of complex HTML based applications and reduction of processing power requirement on the server for delivering user interface. As an additional benefit this framework also provides for the isolation of user interface development from business logic development.

Reduction of Network Bandwidth Requirements

Large OLTP applications, similar to a banking application or an order entry and processing application often have complex user interfaces with lots of fields. By the very nature of these applications, the data contained in the screen are dynamic. It is common for application architects to choose HTML as the base for user interface, generated through server side scripting technologies like JSP or ASP. Since HTML pages generated with JSP and ASP are dynamic, they are not cached at the browser. This results in the complete HTML page to be returned back from the server every time a form is submitted. In complex screens involving many validations and criteria listings, there could be many data validation requests to the server before the transaction is completed. Avoiding transmission of complete HTML in every response can save a considerable amount of bandwidth.

Reduction of CPU Resource Requirements for Delivering UI

Server side scripting technologies like ASP and JSP create HTML response on the fly by writing the bytes constituting the HTML response into the output stream. As described above, the large size of HTML data, their dynamic nature which prevents caching, and the additional validation requests that are often sent before the final submission results in additional processing power being consumed at the server to create the complete response for every request. Though the percentage of CPU power consumed in doing this may be small compared to that consumed by the business logic, it nevertheless is superfluous. To reduce the bandwidth requirements resulting because of this HTTP compression utilities are often deployed, which put additional burden on the processing power. Minimizing HTML creation and data traffic at the source will lessen this burden.

Isolation of UI Design from Business Logic Development

Server side scripting technologies like JSP and ASP allow script code to mingle with HTML user interface. This can often lead to situations where both the UI designer and

the business logic developer need to work on the same file and inadvertently intrude on each other's logic. It is also sometimes difficult to edit such mingled server side scripting pages even with modern GUI based editors. Wouldn't the UI developer love to deliver just a pure HTML and be assured that it is going to integrate seamlessly with the business logic? And wouldn't the business logic developer prefer to just deliver pure data and not bother about the UI?

TARGETS AND SCOPE

In view of the needs for this framework as discussed in the previous section, the following targets are set for this framework to achieve:

1. Reduce network bandwidth requirements through reduction of HTML data traffic.
2. Reduce CPU utilization at the server through reduction of dynamic HTML generation at the server.
3. Provide for isolation of UI design from business logic.

No quantitative figures are set for the targets as they depend on implementation details, and because the purpose of this paper is just to introduce the concept rather that to provide a final product. The basic idea discussed in this article can be manifested in different ways.

A basic version of the framework has been developed along with a simple application to demonstrate its effectiveness. The simple application constitutes of five HTML screens, each with a HTML form. Each form consists of about thirty fields and the overall size of each screen will be around 12KB. Each screen of the application is accompanied with a standard header, footer and a side bar that holds a menu tree. The HTML pages can be viewed in both Microsoft Internet Explorer 6.0 and Mozilla 1.6. Active components (like ActiveX and Java Applets) are not used in the sample application.

THE FRAMEWORK

Concepts behind the Framework

If an analysis is made of the data traffic in a typical HTML/HTTP based application, it can be found that more than 90% of data traffic in every response consists of HTML layout information and field data that hasn't changed.

HTML layout information can be cached by the browser. However, because dynamic

form data is typically embedded within the HTML layout information, no clear separation is available and as a result the complete page needs to be sent every time. We therefore need to isolate layout information from data and send them separately. Parts of form data submitted from the browser also remain static when the user progressively fills up the form and saves the data intermittently. Additional savings can be done by sending only changed data across in HTTP request and response.

The basic version of the framework implemented here will use JavaScript arrays to send data for HTML forms. The layout portion of the forms will consist of static HTML with JavaScript. Using static HTML and JavaScript for the UI layout isolates the business logic from the UI completely. The HTML files can be designed using any standard GUI based HTML editor. The UI designer can provide a JavaScript based interface for the business logic developer to interface with and manipulate UI attributes like colors, visibility and font or to dynamically populate portions of the HTML screen. Thus, a portion of the dynamic nature of the UI which was earlier achieved through JSP will be done at the browser. The UI designer can test the dynamic behavior of the UI independently by invoking the interface functions.

Using a static HTML file for the UI layout ensures that it can be cached by the browser. The dynamic data arrives as a JavaScript array. A few DHTML functions act as the glue between the static HTML and the dynamic data. The static HTML representing the screen is changed only when the data from the server indicates that it belongs to a different screen than what is getting displayed. Otherwise the incremental data is just applied over the existing screen. This ensures maximum caching and minimum overhead on the browser, the network and the server.

Building the Framework

Figure 1 illustrates the various frames of the user interface and their purpose. The application is invoked through a startup URL which provides a frameset with two frames. One of the frames (frame A) is visible and contains the actual HTML user interface. The invisible frame (frame B) contains framework JavaScript. Forms and links are displayed on the visible frame. However, on submit of a form or click of a link, routines provided by the framework are invoked. These routines assemble the data to be submitted, filter out unchanged data and submit the data through the internal hidden frame (frame C) to the server.

Response from the server arrives in frame C. The response indicates the next screen to be displayed and the data to be set in the screen. This data is then examined by the framework JavaScript in frame B. If the screen to be displayed in not the one that is currently showing, it requests for the new HTML to be displayed in the visible frame and then sets the data supplied to the HTML screen. Otherwise it just applies the new data

to the existing screen.

Code listings 1 through 4 provide the details of the implementation of the framework. This along with code listings 5 through 6 form the sample application that can be used to get a feel of this framework. The same application implemented using traditional JSP mechanism is provided in code listings 7 through 8.

The source code for the sample application is available along with this article as a J2EE web application archive. This can be deployed on a JSP engine and explored.

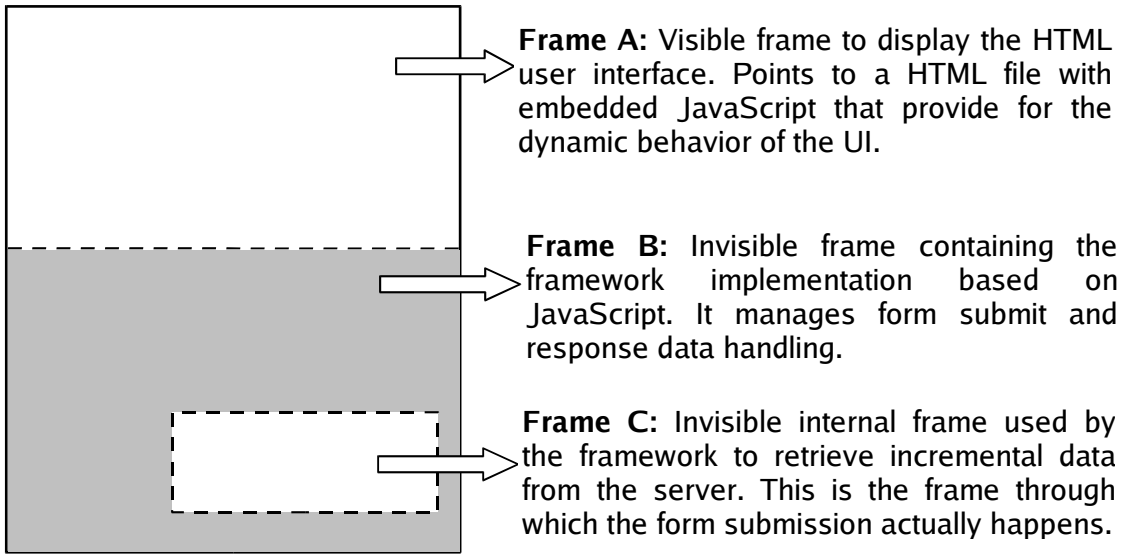

Figure 1: Purpose and contents of the different frames

Working of the Framework

Figure 2 illustrates the workings of the framework as a flow chart. In a typical scenario, the following steps constitute the work cycle:

1. Framework is loaded by typing in a URL on the browser address bar. The initial URL may also indicate the UI to be displayed to start with.
2. Events are handled by the framework event handler JavaScript. Events are mostly generated through user actions (click of a button or a link), but could also be internal (loading of a new page can trigger a data load event).

3. The initial screen (HTML) is displayed by the framework.

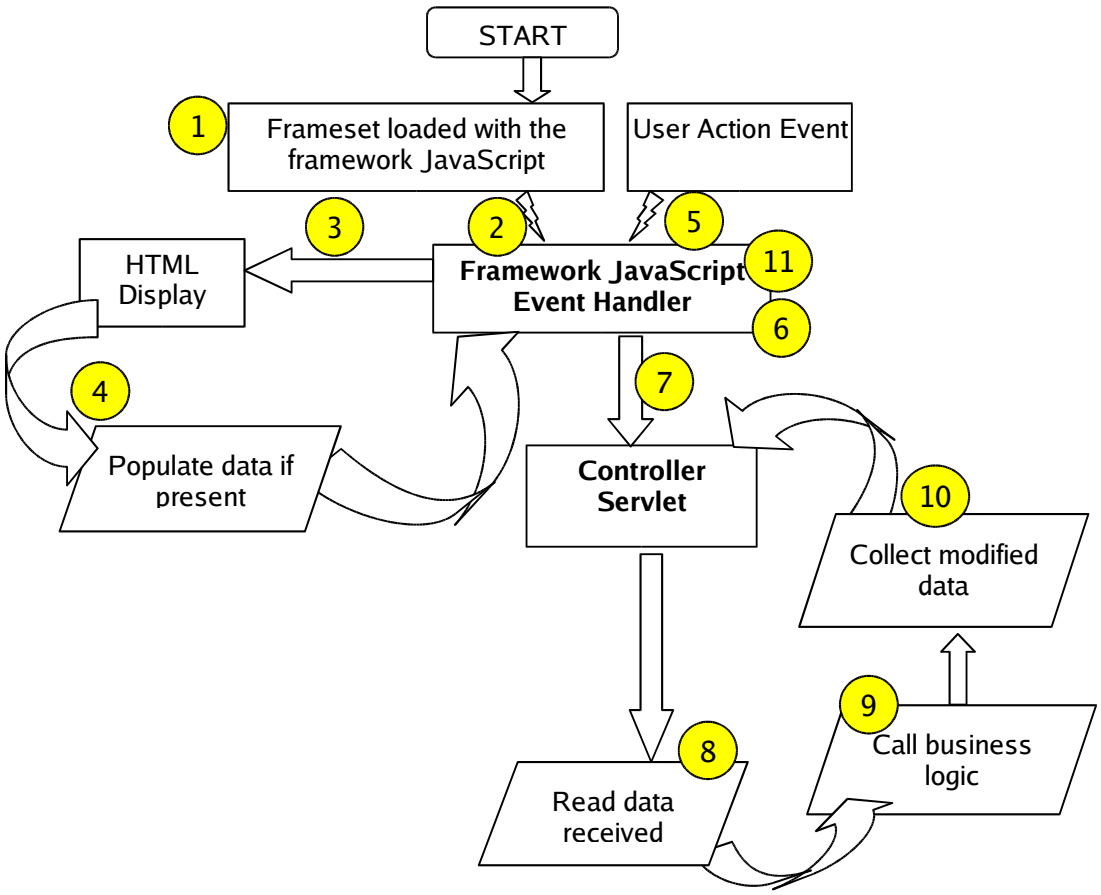

Figure 2: Working of the framework

4. The screen invokes the framework to set onto itself any data that might be available. The framework in turn examines the data frame and populates available data onto the screen.
5. On a form submit action by the user, the framework JavaScript gets called.
6. The framework JavaScript then collects all modified form data from the screen. To check whether the form data has been modified, it is compared with the data that was received initially from the server.
7. The framework JavaScript creates a new HTML form dynamically in frame C

and sets the values accumulated in the previous step into this form. It then submits this form to the controller servlet.

8. The controller servlet reads the incoming name value pairs data and sets them in the session.
9. The controller servlet then calls the business logic to let it work on the input data and come up with the outputs. The business logic uses a function provided by the controller servlet to set output values into the session so that the controller servlet can keep track of values modified by the business logic.
10. The controller servlet then accumulates the modified data, creates a response HTML with modified data in a particular format that can be understood by the framework JavaScript and sends the response back.
11. Once the response HTML is loaded by the browser, it triggers the framework to process the response. If the response requires a new screen to be displayed, the framework points the UI frame (frame A) to the new HTML. From here onwards, steps 4 through 11 are cycled again and again till the application is active.

COMPARISON OF RESULTS

Network traffic generated by the non optimized application and the optimized application (using this framework) were compared by analyzing logs from the web server and a HTTP proxy server.

Table 1 lists the actions performed and the corresponding HTML data traffic generated for the non optimized application. Similarly table 2 lists the data for the optimized application. The data size has been expressed in bytes and does not include the HTTP header.

Action	Request	Request data size (bytes)	Response data size (bytes)
Invoke URL /net/noopt/page1.jsp to bring up the first page	GET /net/noopt/page1.jsp	0	12373
Set target page as page 1, modify first 4 fields of the form and submit.	POST /net/noopt/page1.jsp	876	12377
Set target page as page 1, modify 2 more fields and submit.	POST /net/noopt/page1.jsp	878	12379
Set target page as page 2, modify 2 more fields and submit.	POST /net/noopt/page2.jsp	880	12376
Set target page as page 2, modify first 4 fields of the form and submit.	POST /net/noopt/page2.jsp	876	12380

Set target page as page 2, modify 2 more fields and submit.	POST /net/noopt/page2.jsp	878	12382
Set target page as page 1, modify 2 more fields and submit.	POST /net/noopt/page1.jsp	880	12381
Total (bytes)		**5268**	**86648**

Table 1: HTML traffic data for non optimized application

Action	Request	Request data size (bytes)	Response data size (bytes)
Invoke URL /net/opt/ to bring up the first page	GET /net/opt/	0	316
	GET / net/opt/fwk.jsp?initpage=page1	0	3572
Set target page as page 1, modify first 4 fields of the form and submit.	POST /net/OptServlet	903	963
Set target page as page 1, modify 2 more fields and submit.	POST /net/OptServlet	79	139
Set target page as page 2, modify 2 more fields and submit.	POST /net/OptServlet	79	165
Set target page as page 2, modify first 4 fields of the form and submit.	POST /net/OptServlet	904	963
Set target page as page 2, modify 2 more fields and submit.	POST /net/OptServlet	79	139
Set target page as page 1, modify 2 more fields and submit.	POST /net/OptServlet	79	1014
Total (bytes)		**2123**	**7271**

Table 2: HTML traffic data for optimized application (using this framework)

Comparing the data thus obtained and listed in tables 1 and 2 we find that there is a considerable reduction in data traffic in both the request as well as the response data. Table 3 below lists the cumulative data size and the percentage reduction in data traffic.

	Non optimized	Optimized	Reduction
Request data size (bytes)	5268	2123	60%
Response data size (bytes)	86648	7271	92%

Table 3: Comparison of size of data traffic

The spikes in request and response data as observed for the optimized applications (steps 2, 5 and 8) occur during a change of screen. All data corresponding to a screen get sent during a screen change as the framework does not remember data for past screens. These spikes can be eliminated and data traffic can be reduced further by enhancing the framework to cache data for previous screens.

CONCLUSIONS

HTML is an attractive platform for building user interface. Optimizing large HTML based applications results in considerable savings in network bandwidth and processing overhead at the server. The proposed framework reduces data traffic through innovative use of HTML and JavaScript. The sample application implemented has validated the feasibility and demonstrated the effectiveness of this solution.

CODE LISTINGS

Code Listing 1 (Startup page for optimized application)

```
<%@page contentType="text/html"%>
<%--------------------------------------------------------------------------------
The startup page. Contains a frameset with two frames:
1. ui: Contains the UI that is displayed (visible, but initially blank)
2. fwk: Contains the framework components. (invisible)

This can potentially also do
1. caching of data
2. validation of data before submitting
3. hold multiple frames that can be hidden or displayed for aggressive caching
--------------------------------------------------------------------------------%>
<HTML>
  <HEAD><TITLE>Optimized Framework</TITLE></HEAD>
<%
String sPage2Show = request.getParameter("initpage");
if(null == sPage2Show) sPage2Show="page1";
%>
  <frameset rows="*,0px">
    <frame src='about:blank' name='ui' id='ui' border='no'
         frameborder='no' scrolling='auto'>
    <frame src='fwk.jsp?initpage=<%=sPage2Show%>' name='fwk' id='fwk' border='no'
         frameborder='no' scrolling='no'>
  </frameset>
</HTML>
```

Code Listing 2 (Framework JavaScript for optimized application)

```
<%--
-------------------------------------------------------------------------------
The framework page. Contains JavaScript functions to manage:
1. page display
2. form submit
3. page population with response data

This can potentially also do
1. caching of data
2. validation of data before submitting
-------------------------------------------------------------------------------
--%>

<%@page contentType="text/html"%>
<html>
<head>
    <title>Framework</title>
    <script language='JavaScript'>
<%--
        // Variable used to switch the debug message level of the framework.
--%>
        var iDebugLevel = 1;

<%--
        // String trim function.
--%>
        String.prototype.trim = function() {
            return this.replace( /^\s*/, '').replace( /\s*$/, '' );
        }

<%--
        // Stores the name of the current page now showing.
        // This is also passed back to the server on any form submit.
--%>
        var sCurrPage = '';

<%--
        // Stores the old values for the page showing. Old values are obtained
        // from the response received from the server.
--%>
        var sOldValues = new Array();


<%--
        // Function to submit a form. Goes through the form and collects
        // modified data. Creates a html form with modified data and submits
        // the same to the controller servlet.
--%>
        function chkAndSubmit() {
            var totlen = parent.ui.document.frm.length;
            var sNewValuesName = new Array();
            var sNewValuesValue = new Array();
            for(var iIndex=0; iIndex<totlen; iIndex++) {
                var sName = parent.ui.document.frm[iIndex].name;
                var sValue = parent.ui.document.frm[iIndex].value;
```

```
                if(sOldValues[sName] != sValue) {
                    if(2 == iDebugLevel) alert(sName+'='+sValue);
                    if(2 == iDebugLevel) alert('sOldValues[sName]='+sOldValues
[sName]);
                    sNewValuesName[iIndex] = sName;
                    sNewValuesValue[iIndex] = sValue;
                }
            }
            sNewValuesName[iIndex] = 'CurrPage';
            sNewValuesValue[iIndex] = sCurrPage;

            var totlen = sNewValuesName.length;
            var formsubstr = '<html><body><form name="sub" id="sub"
action="/net/OptServlet" method="post">';
            for(var iIndex=0; iIndex<totlen; iIndex++) {
                if(sNewValuesName[iIndex] && sNewValuesValue[iIndex]) {
                    if(2 == iDebugLevel) alert("appending index " + iIndex);
                    formsubstr += '<input type="hidden" name="';
                    formsubstr += sNewValuesName[iIndex];
                    formsubstr += '" value="';
                    formsubstr += sNewValuesValue[iIndex];
                    formsubstr += '">';
                }
            }
            formsubstr += '</form></body></html>';
            if(1 == iDebugLevel) alert(formsubstr);
            spad.document.write(formsubstr);
            spad.document.sub.submit();
        }
<%--
        // Displays a page in the display frame. Checks whether the page is
        // already showing before showing the new page
--%>
        function showPage(pg) {
            if(sCurrPage != pg) {
                parent.ui.document.location=('/net/opt/'+pg+'.html');
                sCurrPage = pg;
                sOldValues = new Array();
            }
        }

<%--
        // Used to display the startup page when the framework is loaded for
        // the first time on a browser window
--%>
        function doInit() {
            showPage('<%=request.getParameter("initpage")%>');
        }

<%--
        // Checks for incoming data from the server in the scratch pad frame,
        // parses the data and populates the display frame with the data
--%>
        function setPageValues() {
            var dataDiv = spad.document.getElementById('data');
            if(dataDiv) {
                if(1 == iDebugLevel) alert(dataDiv.innerHTML);
                var nvPairs = dataDiv.innerHTML.split("&amp;");
                var iMax = nvPairs.length;
                for(iIndex=0; iIndex<iMax; iIndex++) {
                    var nvPair = nvPairs[iIndex].split("=");
                    if(nvPair[0] && nvPair[1]) {
```

```
                        var fldname = nvPair[0].trim();
                        var fldval = nvPair[1].trim();
                        sOldValues[fldname] = fldval;

                        parent.ui.setValue(fldname, fldval);
                    }
                }
            }
        }

<%--
        // Checks for incoming page name and data from the server in the
        // scratch pad frame.
        // If scratch pad frame contains a new page to be displayed, calls the
        // showPage function.
        // If no new page is to be displayed, calls the setPageValues function
        // to display the incoming data.
--%>
        function processResult() {
            var sNextPage = sCurrPage;
            var showpageDiv = spad.document.getElementById('showpage');
            if(showpageDiv) {
                sNextPage = showpageDiv.innerHTML;
            }
            if(sNextPage != sCurrPage) {
                showPage(sNextPage);
                return;
            }
            setPageValues();
        }
    </script>
</head>
<body onload='doInit()'>

framework
<%--
The scratch pad frame, which is used to send and receive data to and from
the server. To start with it contains an empty page.
--%>
<iframe name='spad' src='spad.html'></iframe>
</body>
</html>
```

Code Listing 3 (Controller Servlet for optimized application)

```
/**
 * OptServlet.
 * The servlet used by the sample framework as a controller.
 * This servlet manages the session data and determines changed values after
 * every page execution. It calls the business logic at appropriate points of
 * the page execution (currently only once).
 */
import java.io.*;
import java.util.*;

import javax.servlet.http.*;
import javax.servlet.*;
```

```
public class OptServlet extends HttpServlet {
    private static String MOD_VALS = "_MOD_VALS";

    /**
     * If the hash map to store modified values does not exist, creates one.
     * Returns any existing hashmap if present. The hashmap is stored at
     * request scope to avoid synchronization requirements in case of multiple
     * requests arriving for the same session.
     */
    private HashMap getOrCreateModValsMap(HttpServletRequest req) {
        HashMap modVals = (HashMap) req.getAttribute(MOD_VALS);
        if(null == modVals) {
            modVals = new HashMap();
            req.setAttribute(MOD_VALS, modVals);
        }
        return modVals;
    }

    /**
     * Any setting of session attribute done through this method is detected for
     * changed value and set in the hashmap used to provide outputs.
     */
    private void setSessionAttrib(HttpServletRequest req, String sName, Object oVal)
{
        HttpSession session = req.getSession(true);
        HashMap modVals = getOrCreateModValsMap(req);

        Object ori = session.getAttribute(sName);
        if( (null == ori) || (!ori.equals(oVal)) ) {
            session.setAttribute(sName, oVal);
            modVals.put(sName, oVal);
        }
    }

    /**
     * The sample business logic. Currently this only modifies two fields in
     * the session. This is the place where the application beans can be called.
     */
    private HashMap doBL(HttpServletRequest req) {
        setSessionAttrib(req, "page1.tf3", "modified21");
        setSessionAttrib(req, "page2.tf3", "modified22");
        HashMap modVals = getOrCreateModValsMap(req);
        return modVals;
    }

    /**
     * The service method of the controller servlet. Does the following steps:
     * 1. Gets all the request parameters and sets them to the session. The
     *    request parameter names are currently hard coded, but in reality it
     *    would probably call a page specific bean to collect the data.
     * 2. Collects the name of the page submitted and the next page requested.
     *    This is purely an implementation for the sample program and is very
     *    likely to be replaced by a more sophisticated navigation mechanism.
     * 3. Calls the business logic.
     * 4. Creates and sends the response back to the browser. Response is in the
     *    form of a coded message. If a new UI is getting displayed, sends data
     *    for all fields of the form as the browser will not be having any
     *    previous data. Otherwise, it sends only modified data as the browser
     *    is already displaying the UI with previous data.
     */
    public void service(HttpServletRequest req, HttpServletResponse res) throws
ServletException, IOException {
```

```
        int jIndex;
        HttpSession session = req.getSession(true);
        String sCurrPage, sNextPage;

        for(jIndex=1; jIndex < 41; jIndex++) {
            String s;
            if(null != (s = req.getParameter("page1.tf"+jIndex))) setSessionAttrib
(req, "page1.tf"+jIndex, s);
            if(null != (s = req.getParameter("page2.tf"+jIndex))) setSessionAttrib
(req, "page2.tf"+jIndex, s);
        }
        sCurrPage = req.getParameter("CurrPage"); // get current page
        sNextPage = "page"+req.getParameter("NextPage"); // get page requested

        // call business logic that can modify data
        HashMap moddata = doBL(req);

        // send response
        PrintWriter pw = res.getWriter();
        pw.println("<html><body onload='parent.processResult();'>");
        if(!sNextPage.equals(sCurrPage)) { // send page to display if required
            pw.println("<div id='showpage' name='showpage'>"+sNextPage+"</div>");
            pw.println("<div id='data' name='data'>");
            // if new page send all data present
            for(jIndex=1; jIndex < 41; jIndex++) {
                String sName = sNextPage+".tf"+jIndex;
                String sVal = (String)session.getAttribute(sName);
                if(null != sVal) pw.print(sName+"="+session.getAttribute(sName)+"&");
            }
            pw.println("");
            pw.println("</div>");
            pw.println("</body></html>");
        }
       else {
            // else send modified data
            pw.println("<div id='data' name='data'>");
            Object [] allKeys = moddata.keySet().toArray();

            int iMax = allKeys.length;
            for(jIndex=0; jIndex < iMax; jIndex++) {
                String sName = (String)allKeys[jIndex];
                if(sName.startsWith(sNextPage)) {
                    String sVal = (String)moddata.get(sName);
                    if(null != sVal) pw.print(sName+"="+moddata.get(sName)+"&");
                }
            }
            pw.println("");
            pw.println("</div>");
            pw.println("</body></html>");
        }
    }
}
```

Code Listing 4 (Deployment descriptor for optimized application)

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<!DOCTYPE web-app  PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
```

```
"http://java.sun.com/dtd/web-app_2_3.dtd">


<web-app>
 <display-name>HTTP HTML Optimization Demo</display-name>
 <description>HTTP HTML Optimization Demo</description>

 <servlet>
    <servlet-name>OptServlet</servlet-name>
    <servlet-class>OptServlet</servlet-class>
 </servlet>

 <servlet-mapping>
     <servlet-name>OptServlet</servlet-name>
     <url-pattern>/OptServlet</url-pattern>
 </servlet-mapping>
</web-app>
```

Code Listing 5 and 6 (Optimized application UI pages 1 and 2)

Code listing 6 is similar to code listing 5. Please refer to source code provided as a download. Partial listing for UI page 1 is provided below.

```
<!--
================================================================================
The first page of the optimized sample application.
This page has 40 labels and associated text fields to simulate a large form.
Though it is possible to optimize this HTML for size in many different ways, it
has not been attempted because:
1. Typical GUI editors generate similar HTML
2. Our goal is to demonstrate the effectiveness of the framework inspite of
   non optimized HTML.

The page contains a dropdown which can be used to select which page to go next
to. This information is used by the controller to request display of the new
page. This is purely an implementation for the sample program and is very likely
to be replaced by a more sophisticated navigation mechanism.
================================================================================
-->
<html>
<head>
    <title>Optimized - JSP Page 1</title>
    <script language='JavaScript'>
        // called by the form of this page to submit data
        function chkAndSubmit(whichPage) {
            if(whichPage) document.frm['NextPage'].value = whichPage;
            parent.fwk.chkAndSubmit();
        }

        // called by the framework when it wants to populate data that has
        // arrived from the server
        function setValue(name, val) {
            document.frm[name].value = val;
        }
    </script>
</head>

<!--
```

```
onload calls the framework to populate any data that has arrived with the page
information. This happens only the first time when the page loads. Till the same
page is getting displayed, the framework will cann the setValue method to set
new data arriving from the server.
-->
<body onload='parent.fwk.setPageValues()'>
<table CELLPADDING="0" CELLSPACING="0" WIDTH="100%" BORDER="0">
    <tr valign=top align=center bgcolor='#cccccc'>
        <td colspan=2>
            <font face='Arial' size='5'>A Framework for Building Network Efficient
HTML Based User Interface <br>for OLTP Applications</font><br>
            <font face='Arial' size='3'>(Optimized - Page 1)</font>
        </td>
    </tr>
    <tr valign=top>
        <td valign=top align=center bgcolor='#cccccc'>
            <font face='Arial'>Sample Menu 1 <small>(valid)</small></font><br>
            <ul>
                <a href='javascript:chkAndSubmit("1");'><font
face='Arial'><small>Page1</small></font></a><br>
                <a href='javascript:chkAndSubmit("2");'><font
face='Arial'><small>Page 2</small></font></a><br>
            </ul>
            <font face='Arial'>Sample Menu 1 <small>(invalid)</small></font><br>
            <ul>
                <a href=''><font face='Arial'><small>Item 1</small></font></a><br>
                <a href=''><font face='Arial'><small>Item 2</small></font></a><br>
                <a href=''><font face='Arial'><small>Item 3</small></font></a><br>
                <a href=''><font face='Arial'><small>Item 4</small></font></a><br>
                <a href=''><font face='Arial'><small>Item 5</small></font></a><br>
                <a href=''><font face='Arial'><small>Item 6</small></font></a><br>
            </ul>
        </td>
        <td>
            <form id='frm' name='frm' method='post' action=''>
                <table cellpadding='10' cellspacing='10'>
                    <tr>
                        <td colspan='2' align='center'>
                            Go to page
                            <select id='NextPage' name='NextPage'>
                                <option value='2'>2</option>
                                <option value='1'>1</option>
                            </select>
                            <input type='button' name='GoToPage' id='GoToPage'
value='Go' onclick='chkAndSubmit();'>
                        </td>
                    </tr>

                    <tr>
                        <td>
                            <font face='Arial' size='2'>Page1 Text Field 1:</font>
<input type=TEXT name='page1.tf1' id='page1.tf1' value='page1.tf1'>
                        </td>

                        <td>
                            <font face='Arial' size='2'>Page1 Text Field 2:</font>
<input type=TEXT name='page1.tf2' id='page1.tf2' value='page1.tf2'>
                        </td>
                    </tr>
--------------------------------------------------------------------------------
Similarly for all other text fields (40 text fields in total) in the page.
Please refer to source code provided as a download for the complete listing.
```

```
--------------------------------------------------------------------------------
                </table>
            </form>
        </td>
    </tr>
</table>

<hr noshade>
<center>
<font face='Arial' size='1'>This is the optimized application</font>
</center>
</body>
</html>
```

Code Listing 7 and 8 (Non optimized JSP application UI pages 1 and 2)

Code listing 8 is similar to code listing 7. Please refer to source code provided as a download. Listing for UI page 1 is provided below.

```
<!--
================================================================================
The first page of the non optimized sample application.
This page has 40 labels and associated text fields to simulate a large form.
Though it is possible to optimize this JSP for size in many different ways, it
has not been attempted because:
1. Typical GUI editors generate similar HTML
2. Our goal is to demonstrate the effectiveness of the framework inspite of
   non optimized HTML.

The page contains a dropdown which can be used to select which page to go next
to. This information is used by the controller to request display of the new
page. This is purely an implementation for the sample program and is very likely
to be replaced by a more sophisticated navigation mechanism.
================================================================================
--%>
<%@page contentType="text/html"%>
<html>
<head>
    <title>Non Optimized - JSP Page 1</title>
    <script language='JavaScript'>
        function chkAndSubmit(whichPage) {
            if(whichPage) document.frm['NextPage'].value = whichPage;

            var sNextPage = 'page2.jsp';
            if(whichPage && (1 == whichPage)) sNextPage = 'page1.jsp';
            else if(1 == document.frm.NextPage.value) sNextPage = 'page1.jsp';

            document.frm.action = sNextPage;
            document.frm.submit();
        }
    </script>
</head>
<body>

<%
    // Collects incoming data and populates the session
    // Typically this or the portion after this will contain the business logic
    // embedded in a bean.
    for(int jIndex=1; jIndex < 41; jIndex++)
```

```
    {
        String s;
        if(null != (s = request.getParameter("page1.tf"+jIndex)))
session.setAttribute("page1.tf"+jIndex, s);
        if(null != (s = request.getParameter("page2.tf"+jIndex)))
session.setAttribute("page2.tf"+jIndex, s);
    }
%>

<table CELLPADDING="0" CELLSPACING="0" WIDTH="100%" BORDER="0">
    <tr valign=top align=center bgcolor='#cccccc'>
        <td colspan=2>
            <font face='Arial' size='5'>A Framework for Building Network Efficient
HTML Based User Interface <br>for OLTP Applications</font><br>
            <font face='Arial' size='3'>(Non Optimized - Page 1)</font>
        </td>
    </tr>
    <tr valign=top>
        <td valign=top align=center bgcolor='#cccccc'>
            <font face='Arial'>Sample Menu 1 <small>(valid)</small></font><br>
            <ul>
                <a href='javascript:chkAndSubmit(1);'><font face='Arial'><small>Page
1</small></font></a><br>
                <a href='javascript:chkAndSubmit(2);'><font face='Arial'><small>Page
2</small></font></a><br>
            </ul>
            <font face='Arial'>Sample Menu 1 <small>(invalid)</small></font><br>
            <ul>
                <a href=''><font face='Arial'><small>Item 1</small></font></a><br>
                <a href=''><font face='Arial'><small>Item 2</small></font></a><br>
                <a href=''><font face='Arial'><small>Item 3</small></font></a><br>
                <a href=''><font face='Arial'><small>Item 4</small></font></a><br>
                <a href=''><font face='Arial'><small>Item 5</small></font></a><br>
                <a href=''><font face='Arial'><small>Item 6</small></font></a><br>
            </ul>
        </td>
        <td>
            <form id='frm' name='frm' method='post' action=''>
                <table cellpadding='10' cellspacing='10'>
                    <tr>
                        <td colspan='2' align='center'>
                            Go to page
                            <select id='NextPage' name='NextPage'>
                                <option value='2'>2</option>
                                <option value='1'>1</option>
                            </select>
                            <input type='button' name='GoToPage' id='GoToPage'
value='Go' onclick='chkAndSubmit();'>
                        </td>
                    </tr>
<%
                    // The below loop is used to create 41 labels and their text
fields

                    for(int iIndex=1; iIndex < 41; iIndex++)
                    {
                        String sVal1 = (String)session.getAttribute
("page1.tf"+iIndex);
                        String sVal2 = (String)session.getAttribute("page1.tf"+
(iIndex+1));
                        if(null == sVal1) sVal1 = "page1.tf"+iIndex;
                        if(null == sVal2) sVal2 = "page1.tf"+(iIndex+1);
```

```
%>
                    <tr>
                        <td>
                            <font face='Arial' size='2'>Page1 Text Field <%=iIndex%
>:</font> <input type=TEXT name='page1.tf<%=iIndex%>' id='page1.tf<%=iIndex%>'
value='<%=sVal1%>'>
                        </td>
<%
                        iIndex++;
%>
                        <td>
                            <font face='Arial' size='2'>Page1 Text Field <%=iIndex%
>:</font> <input type=TEXT name='page1.tf<%=iIndex%>' id='page1.tf<%=iIndex%>'
value='<%=sVal2%>'>
                        </td>
                    </tr>
<%
                    }
%>
                </table>
            </form>
        </td>
    </tr>
</table>

<hr noshade>
<center>
<font face='Arial' size='1'>This is a normal application</font>
</center>

</body>
</html>
```

Tanmay K. Mohapatra *is a Technical Architect associated with the Banking Products Division of Infosys Technologies Ltd., Bangalore, India.*