eMobile: A Sample End-to-End Application Using the Java™ 2 Platform, Enterprise Edition

Integrating an Enterprise Application with Various Client Devices

Part 2: Using Servlet, XML, and XSLT Technologies

Thierry Violleau

MDE Enterprise Java Team

Sun Microsystems, Inc.

Dec 2000

Copyright 2000 Sun Microsystems, Inc., 901 San Antonio Road, Palo Altos, California 94303 USA. All rights reserved.

This product or document is protected by copyright and distributed under licenses restricting its use, copying, distribution, and decompilation. No part of this product or document may be reproduced in any form by any means without prior written authorization of Sun and its licensors, if any. Third-party software, including font technology, is copyrighted and licensed from Sun suppliers.

Parts of the product may be derived from Berkeley BSD systems, licensed from the University of California. UNIX is a registered trademark in the U.S. and other countries, exclusively licensed through X/Open Company, Ltd.

Sun, Sun Microsystems, the Sun logo, EJB, J2EE, JDK, Java, JavaBeans, Netscape, Solaris, and Ultra are trademarks, registered trademarks, or service marks of Sun Microsystems, Inc. in the U.S. and other countries. All SPARC trademarks are used under license and are trademarks or registered trademarks of SPARC International, Inc. in the U.S. and other countries. Products bearing SPARC trademarks are based upon an architecture developed by Sun Microsystems, Inc.

The OPEN LOOK and Sun™ Graphical User Interface was developed by Sun Microsystems, Inc. for its users and licensees. Sun acknowledges the pioneering efforts of Xerox in researching and developing the concept of visual or graphical user interfaces for the computer industry. Sun holds a non-exclusive license from Xerox to the Xerox Graphical User Interface, which license also covers Sun's licensees who implement OPEN LOOK GUIs and otherwise comply with Sun's written license agreements.

Federal Acquisitions: Commercial Software-Government Users Subject to Standard License Terms and Conditions.

DOCUMENTATION IS PROVIDED "AS IS" AND ALL EXPRESS OR IMPLIED CONDITIONS, REPRESENTATIONS AND WARRANTIES, INCLUDING ANY IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT, ARE DISCLAIMED, EXCEPT TO THE EXTENT THAT SUCH DISCLAIMERS ARE HELD TO BE LEGALLY INVALID.

(c) 2000 Sun Microsystems, Inc.

ALL RIGHTS RESERVED

License Grant-Permission to use, copy, modify, and distribute this Software and its documentation for NON-COMMERCIAL or COMMERCIAL purposes and without fee is hereby granted.

This Software is provided "AS IS". All express warranties, including any implied warranty of merchantability, satisfactory quality, fitness for a particular purpose, or non-infringement, are disclaimed, except to the extent that such disclaimers are held to be legally invalid.

You acknowledge that Software is not designed, licensed or intended for use in the design, construction, operation or maintenance of any nuclear facility ("High Risk Activities"). Sun disclaims any express or implied warranty of fitness for such uses.

Please refer to the file http://www.sun.com/policies/trademarks/ for further important trademark information and to http://java.sun.com/nav/business/index.html for further important licensing information for the Java Technology.

Table of Contents

Introduction....................5
  About the eMobile Sample Application....................5
Web Server Tier....................7
  Style Sheet Transformations....................8
    Style Sheet Engine Adapters....................9
      Loading a Style Sheet....................9
      Clearing and Setting Style Sheet Parameters....................10
      Applying a Style Sheet....................11
      Instantiating the XSLT Engine Adapter....................11
      Configuration....................12
    Selecting the Proper Style Sheets....................12
      Mapping MIME Types to Style Sheets....................12
      Configuration....................13
    Applying the Style Sheets....................14
  Generating HTML content....................15
  Generating WML content....................22
  Performance Improvement....................29
    Caching the DOM Trees....................29
    Caching the Style Sheets....................32
The Configuration Demonstrated....................33
Conclusion....................34
References and Resources....................35

Introduction

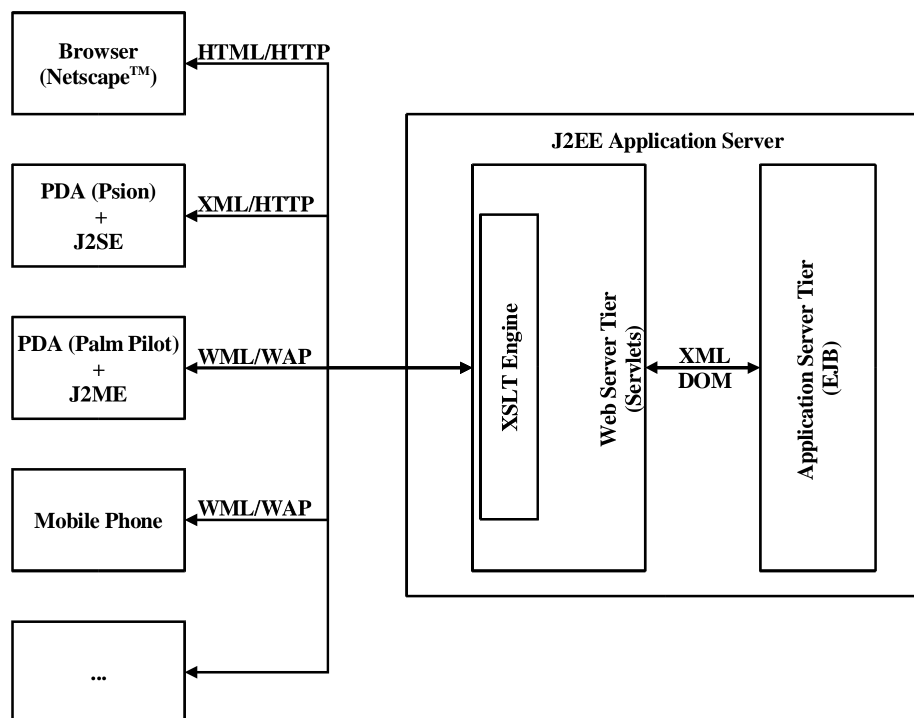
The Java™ 2 Platform Enterprise Edition (J2EE™) architecture provides a standard framework within which enterprises can quickly and efficiently develop and deploy end-to-end e-commerce applications. This paper is the second of a series that explains and illustrates how an application based on J2EE technologies and using a J2EE Application Server can be easily accessed from cell phones or Java™ technology-enabled devices like Psion or Palm Pilot.

The first paper, "Part1: Using Servlet, Applet, and XML Technologies," covered the Web Tier and Client Tier implementations of the eMobile sample application based on servlet, applet, and XML technologies. This paper illustrates how other client devices are integrated using XSLT (eXtensible Style Sheet Language Transformations) technology on the Web Tier.

About the eMobile Sample Application

eMobile is a sample application that demonstrates the use of Enterprise JavaBeans™ (EJB™) technology by modeling an online car-buying service that allows a potential customer to:

- Search for cars based on preferences.
- Select and buy a car.
- Obtain financing information for the car selected.



General architecture of the eMobile application

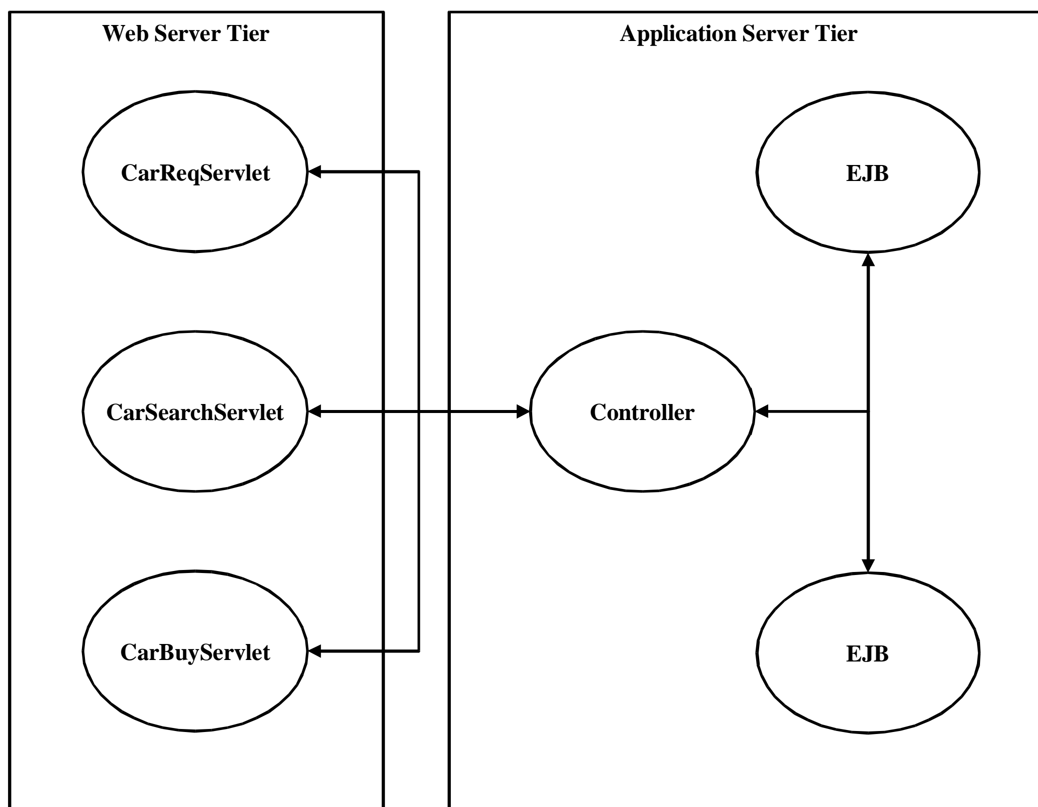
The eMobile sample application is based on J2EE technologies including EJB 1.1 and Servlet 2.2, and can be deployed on any J2EE-compliant server.

The eMobile sample application consists of a combination of Session and Entity beans that provide the business logic to handle transactions. eMobile accesses a database that contains the dealer catalog and inventory. Dealers advertise all of the car models they sell. If the car is not in stock, a broker can order the car directly from the manufacturer. From their search results, users are able to select and order cars. The results from the EJB are passed on to the servlets to convert the content to a format supported by the requesting client device.

Web Server Tier

As described in "Part1: Using Servlet, Applet, and XML Technologies", the web server tier consists of three servlets called `CarReqServlet`, `CarSearchServlet`, and `CarBuyServlet` that interact with the EJB application to process the customer's requests. These servlets are used to obtain the available options used for car selection, to search according to the requirements, and to buy a car matching the requirements, respectively.

The interaction between the servlets and the EJB application is managed by a stateful session bean called `Controller` that returns value objects for each successful request. For each of the three servlets, the value objects returned by the `Controller` are converted to document object model (DOM) trees according to the corresponding document type definitions (DTD), namely `eMobileCarReq.dtd`, `eMobileCarSearch.dtd`, and `eMobileCarBuy.dtd` respectively.

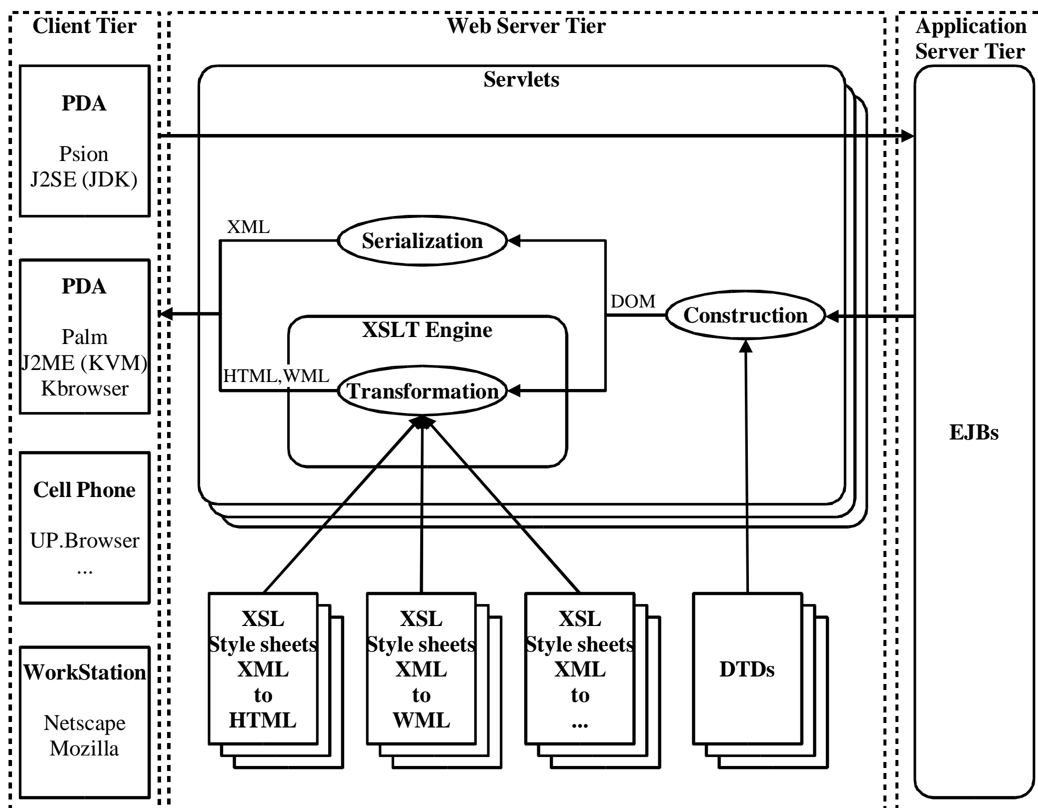


Interaction between eMobile's web and application server tiers

"Part1: Using Servlet, Applet, and XML Technologies" described the next step in generating the content as being only the serialization of the DOM tree. An alternate step was omitted for the sake of simplicity when discussing the generation of XML content; that step is the optional transformation of the XML DOM trees to other formats like HTML and WML (Wireless Markup Language). This transformation is carried out by applying XSL style sheets to the DOM trees.

For each of the three servlets, the overall content generation process from the value objects returned by the EJB application is as follows:

1. Constructing the corresponding DOM tree.
2. Generating the content from the DOM tree as follows:
    a. If an appropriate style sheet is available, transforming the DOM tree to the targeted content type by applying the style sheet.
    b. If no appropriate style sheet is available, directly serializing the DOM tree to generate the XML stream.



Content Generation Process: DOM tree construction and direct serialization or transformation

The construction of the DOM tree and serialization were entirely covered in the chapters "Generating the Document Object Model (DOM) trees" and "Serializing the DOM trees to XML streams" of "Part1: Using Servlet, Applet, and XML Technologies"; therefore this document will only focus on the style sheet transformations.

Style Sheet Transformations

XSL transformations allow defining templates (rules) that will be applied on elements of an XML source document to transform it into another document. The resulting document could be another well-formed XML document (XML, WML...), an HTML document, or any other format provided that the proper output method is available.

An XSLT processor reads both a source XML document and an XSL style sheet. The XSL style sheet is itself a well-formed XML document. In our sample application, the XSL transformations occur whenever such a style sheet, both for the targeted MIME type and the corresponding response's DTD, is available. The appropriate style sheet is loaded and then applied to the response's DOM tree.

Style Sheet Engine Adapters

To deal with different implementations of the XSLT engine, an interface called `XSLTAdapter` was designed, to isolate the application from the XSLT engine implementation, and the SAX or DOM API implementations.

This interface specifies six methods that are implemented by the classes `XalanAdapter` and `SaxonAdapter` to interface with Xalan (from Apache.org) and Saxon (from ICL) respectively. Two methods were already introduced in "Part1: Using Servlet, Applet, and XML Technologies" to:

- Create a new document and set its document type.
- Serialize a document and write it out.

The four remaining methods deal with XSLT and specifically allow :

- Loading a style sheet, given its path.
- Clearing the style sheet parameters.
- Setting the style sheet parameters.
- Applying a style sheet to a document and writing the result on the output stream.

```
public interface XSLTAdapter {

   Document createDocument(String rootTag, String publicId, String systemId);

   void serializeDocument(Document doc,
                          PrintWriter out) throws SAXException, IOException;

   void clearStylesheetParameters() throws SAXException;

   void setStylesheetParameter(String name, String value) throws SAXException;

   void applyStylesheet(Object stylesheet, Document doc,
                          PrintWriter out) throws SAXException, IOException;

   Object loadStylesheet(String path) throws SAXException;
}
```

The style sheet transformation engine adapter interface

Loading a Style Sheet

The `XSLTAdapter loadStylesheet` method allows loading a style sheet from a file. The style sheet object returned is implementation-dependent and is only intended for use by the `XSLTAdapter applyStylesheet` method. The same style sheet object may be used over many transformations.

Following are two code fragments that illustrate how a style sheet is loaded from a file, with Xalan and Saxon respectively.

```
import org.apache.xalan.xslt.*;
...
private XSLTProcessor processor = XSLTProcessorFactory.getProcessor(new
XercesLiaison());
...
public Object loadStylesheet(String path) throws SAXException {
   return processor.processStylesheet(path);
}
```

Loading a style sheet with Xalan.

```
import com.icl.saxon.*;
...
public Object loadStylesheet(String path) throws SAXException {
   PreparedStyleSheet stylesheet = new PreparedStyleSheet();
   ExtendedInputSource inputSource = new ExtendedInputSource(new File(path));
   stylesheet.prepare(inputSource);
   return stylesheet;
}
```

Loading a style sheet with Saxon

Clearing and Setting Style Sheet Parameters

Style Sheets may be passed parameters. Since a style sheet object may be used over many transformations, the parameters from a previous transformation must be cleared prior to being set to their new values.

Following are two code fragments that illustrate how style sheet parameters are set and cleared, with Xalan and Saxon respectively.

```
import org.apache.xalan.xslt.*;
...
private XSLTProcessor processor = ...;
...
public void clearStylesheetParameters() throws SAXException {
   processor.reset();
   return;
}

public void setStylesheetParameter(String name, String value) throws SAXException {
   processor.setStylesheetParam(name, "'" + value + "'");
   return;
}
```

Clearing and setting the style sheet parameters with Xalan

```
import com.icl.saxon.*;
...
private ParameterSet parameters = null;
...
public void clearStylesheetParameters() throws SAXException {
   parameters = new ParameterSet();
   return;
}

public void setStylesheetParameter(String name, String value) throws SAXException {
   parameters.put(name.intern(), new StringValue(value));
   return;
}
```

Clearing and setting the style sheet parameters with Saxon

Applying a Style Sheet

The `XSLTAdapter applyStylesheet` method applies a style sheet previously returned by the `XSLTAdapter loadStylesheet` method to an XML document, then directly serializes the resulting document to an output stream using the output method specified in the style sheet document.

Following are two code fragments that illustrate how style sheets are applied to a DOM tree, with Xalan and Saxon respectively.

```
import org.apache.xalan.xslt.*;
...
private XSLTProcessor processor = ...;
...
public void applyStylesheet(Object stylesheet, Document doc, PrintWriter out) throws
SAXException, IOException {
   ((StylesheetRoot) stylesheet).process(processor, new XSLTInputSource(doc), new
XSLTResultTarget(out));
   return;
}
```

Applying a style sheet with Xalan

```
import com.icl.saxon.*;
...
private ParameterSet parameters = ...;
...
public void applyStylesheet(Object stylesheet, Document doc, PrintWriter out) throws
SAXException, IOException {
   OutputDetails details = new OutputDetails();
   details.setWriter(out);
   StyleSheetInstance stylesheetInstance = ((PreparedStyleSheet)
Stylesheet).makeStyleSheetInstance();
   stylesheetInstance.setOutputDetails(details);
   stylesheetInstance.setParams(parameters);
   stylesheetInstance.renderDocument(DOMDriver.convert(doc), false);
   return;
}
```

Applying a style sheet with Saxon

Instantiating the XSLT Engine Adapter

Depending on the XSLT engine implementation available, one of the XSLT engine adapters must be created. At deployment time, the deployer of the application must specify the proper XSLT adapter (either `XalanAdapter` or `SaxonAdapter`) through an environment parameter.

This parameter is added in the Naming Service and made accessible to the servlet through the `java:comp/env` context. A servlet can use the `Context lookup()` method to get the value of this environment parameter from the Naming Service as follows:

```
String JDNI_XSLT_ADAPTER = "java:comp/env/XSLTAdapter";
...
private XSLTAdapter adapter;
...
try {
   Context context = new InitialContext();
   adapter = (XSLTAdapter) Class.forName(
                       (String) context.lookup(JDNI_XSLT_ADAPTER)).newInstance();
} catch (NamingException ne) {
   ...
} catch (IOException ioe) {
   ...
}
```

Instantiating the XSLT engine adapter specified at deployment time

Configuration

The J2EE web application archive includes a Web Deployment Descriptor. The environment parameter named `XSLTAdapter` is defined in this descriptor and then set to its default value; it can be updated at deployment time with the deployment tool. At deployment time, the proper entry is added in the Naming service.

For example, the Web Deployment Descriptor may include the following lines:

```
<env-entry>
   <description>Class name of the adapter to the XSLT engine</description>
   <env-entry-name>XSLTAdapter</env-entry-name>
   <env-entry-value>com.sun.mde.emobile.servlet.SaxonAdapter</env-entry-value>
   <env-entry-type>java.lang.String</env-entry-type>
</env-entry>
```

Environment parameter for the adapter to the XSLT engine

Selecting the Proper Style Sheets

For each of the three servlets, `CarReqServlet`, `CarSearchServlet`, and `CarBuyServlet`, a Document Type Definition has been specified; these are called `eMobileCarReq.dtd`, `eMobileCarSearch.dtd` and `eMobileCarBuy.dtd` respectively. Similarly, style sheets may be specified for each of those three DTDs regarding a particular MIME type in which the response is to be generated.

Mapping MIME Types to Style Sheets

The proper style sheet is selected according both to the DTD and the targeted MIME type. The names for the style sheet are built up from the name of the DTD and a suffix:

`DTD-name "-" MIME-type-dependent-suffix ".xsl"`

The mapping from MIME types to style sheet suffixes is configurable to allow for the support of new client devices unknown at development time, without requiring any recoding. As for the mapping of clients to MIME types, a `Properties` file is used to configure the mapping of MIME types to style sheet suffixes. A default configuration file containing entries for WML and HTML is provided with the application and a parameter can be set to specify a different configuration file. It's up to the application assembler to include new style sheets and update the mapping file whenever a new MIME type is to be supported. Unlike the mapping of clients to MIME types, the style sheet suffix mapping file is not treated as an external resource referenced by the servlets; it is packaged with the servlets and the style sheets themselves since it has only to be updated when the application assembler includes new style sheets (for example, when the application is reassembled). The assembler can then either set a specific mapping file through an environment parameter or modify the default mapping file itself.

This parameter is added in the Naming Service and made accessible to the servlet through the `java:comp/env` context. A servlet can use the `Context lookup()` method to get the value of this environment parameter from the Naming Service as follows:

```
public static String JDNI_STYLESHEET_SUFFIX_MAP = "java:comp/env/StylesheetSuffixMap";
private Properties stylesheetSuffixMap;
...
try {
   Context context = new InitialContext();
   stylesheetSuffixMap = new Properties();
   stylesheetSuffixMap.load(new FileInputStream(
      getServletContext().getRealPath("/" +
         (String) context.lookup(JDNI_STYLESHEET_SUFFIX_MAP))));
} catch (NamingException ne) {
   ...
} catch (IOException ioe) {
   ...
}
```

Retrieving the name of the local mapping file from the Naming Service and loading it

The path of the style sheet, both for the targeted MIME type and the corresponding response's DTD, is returned by the `getStylesheetPath` method in the `AbstractCarServlet` class that is extended by all the application servlets. The style sheet is then loaded from that path using the `AbstractCarServlet getStylesheet` method. This method relies on an adapter to load the style sheet using the underlying XSLT engine implementation.

```
private String getStylesheetPath(String mimeType) {
   String suffix = stylesheetSuffixMap.getProperty(mimeType, "xml");
   String dtd = "eMobile" + getServletName().substring(0,
      getServletName().length() - "Servlet".length())
   return getServletContext().getRealPath("/" + dtd + "-" + suffix + ".xsl");
}

private Object getStylesheet(String mimeType) throws SAXException {
   Object stylesheet = null;
   String path = getStylesheetPath(mimeType);
   ...
   stylesheet = adapter.loadStylesheet(path);
   ...
   return stylesheet;
}
```

Mapping a MIME type to a style sheet

Configuration

The J2EE web application archive includes a Web Deployment Descriptor. The environment parameter named `StylesheetSuffixMap` is defined in this descriptor and set to point to the default `Properties` file `StylesheetSuffixMap.properties`. It can be updated at deployment time with the deployment tool. At deployment time, the proper entry is added in the Naming service.

For example, the Web Deployment Descriptor may include the following lines:

```
<env-entry>
   <description>Name of the Properties file mapping MIME types to Stylesheet
suffixes</description>
   <env-entry-name>StylesheetSuffixMap</env-entry-name>
   <env-entry-value>StylesheetSuffixMap.properties</env-entry-value>
   <env-entry-type>java.lang.String</env-entry-type>
</env-entry>
```

Environment parameter for the file mapping MIME types to style sheet suffixes

For example, the default `Properties` file named `StylesheetSuffixMap.properties` contains:

```
text/html: html
text/vnd.wap.wml: wml
```

Examples of MIME type to suffix mapping rules

Applying the Style Sheets

XSLT engines allow the passing of parameters to the style sheets. Three kinds of parameters are passed to the style sheets:

- HTTP query string parameters
- HTTP header fields
- URLs to the servlets

The HTTP query string parameters are used to tune the style sheet transformations according to the original query. Those parameters are either those passed to the EJB application or those solely created by, and passed along from, one style sheet invocation to another.

The HTTP header fields, especially `User-Agent` and `Accept` are used to more precisely adapt the content to a particular client device. The `User-Agent` field may be used to recognize a particular client device and may use some specific features or bypass some restrictions of the device. The `Accept` field may be used, for example, to figure out what image format a client device supports.

Passing the URLs to the servlets as parameters to the style sheets allows using the URL rewriting feature of the servlet engine (if available) as an alternate, in case a requesting client device doesn't support cookies.

```
protected Hashtable getParameters(HttpServletRequest request,
                                     HttpServletResponse response) {
   String queryString = request.getQueryString();
   Hashtable parameters = new Hashtable();
   if (queryString != null) {
      Hashtable queryParameters = HttpUtils.parseQueryString(queryString);
      for (Enumeration enum = queryParameters.keys(); enum.hasMoreElements(); ) {
         String key = (String) enum.nextElement();
         String[] values = (String[]) queryParameters.get(key);
         if (values != null && values[0] != null) {
            parameters.put(key, values[0]); // We do not handle multiple values
         }
      }
   }
   for (Enumeration enum = request.getHeaderNames(); enum.hasMoreElements();){
      String name = (String) enum.nextElement();
      String value = request.getHeader(name);
      parameters.put(name, value);
   }
   for (int i = 0; i < URLs.length; i++) {
      parameters.put(URLs[i] + "Alias", response.encodeURL(
         request.getContextPath() + "/" + URLs[i]));
   }
   return parameters;
}
```

Getting the parameters for applying the style sheets; the parameters retuned are the parameters of the request and the encoded URLs to the servlets (cf. URL rewriting)

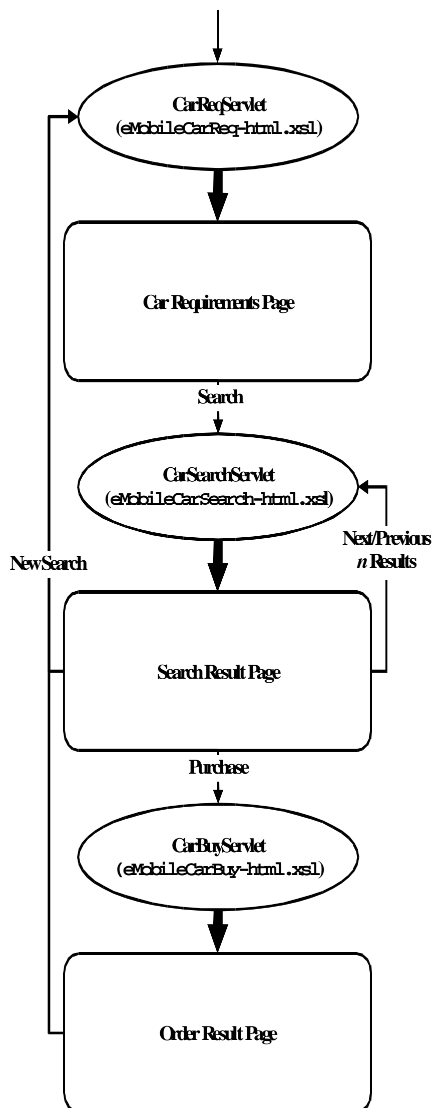
Since a style sheet may be reused from one invocation to another, the parameters are first cleared then set to their new values. The style sheet is then applied to the DOM tree built from the value objects returned by the EJB application. The resulting document is directly serialized to the HTTP response output stream and sent back to the requesting client. The serialization is performed according to the output method specified in the style sheet.

```
protected void applyStylesheet(Document doc, PrintWriter out,
   String mimeType, Hashtable parameters) throws IOException, SAXException {
   Object stylesheet = null;
   // Get the stylesheet required for the specified MIME type
   try {
      stylesheet = getStylesheet(mimeType);
   } catch (SAXException e) {
      if (mimeType.equals(APP_XML_MIME_TYPE)) {
         // No stylesheet for "application/xml". Just serialize the DOM tree and write it
on the output.
         adapter.serializeDocument(doc, out);
         return;
      }
      throw e;
   }
   // Reset the previously set parameters.
   adapter.clearStylesheetParameters();
   // Set the stylesheet parameters (all the query string parameters are passed to the
stylesheet).
   if (parameters != null) {
      for (Enumeration enum = parameters.keys(); enum.hasMoreElements(); ) {
         String key = (String) enum.nextElement();
         adapter.setStylesheetParameter(key, (String) parameters.get(key));
      }
   }
   // Apply the stylesheet to the DOM tree and write the result on the output.
   adapter.applyStylesheet(stylesheet, doc, out);
   return;
}
```

Applying a MIME type dependent style sheet to a DOM tree

Generating HTML content

The generation of HTML content from the DOM trees is performed by the three servlets `CarReqServlet`, `CarSearchServlet`, and `CarBuyServlet` using the style sheets `eMobileCarReq-html.xsl`, `eMobileCarSearch-html.xsl`, and `eMobileCarBuy-html.xsl` respectively. When applied, each style sheet generates an HTML page.



HTML pages workflow

Car Requirements

Select the options of the car you want to buy:

Options			
Make:	FORD	**Model:**	Any
Style:	Any	**Color:**	Any
MinimumPrice:	Any	**Year:**	Any
MaximumPrice:	Any		

Results per Page: 10 **Sorted by:** Price Search

New Search

The page generated by `CarReqServlet` using the `eMobileCarReq-html.xsl` style sheet and displayed in a Netscape browser

Search Results: 215

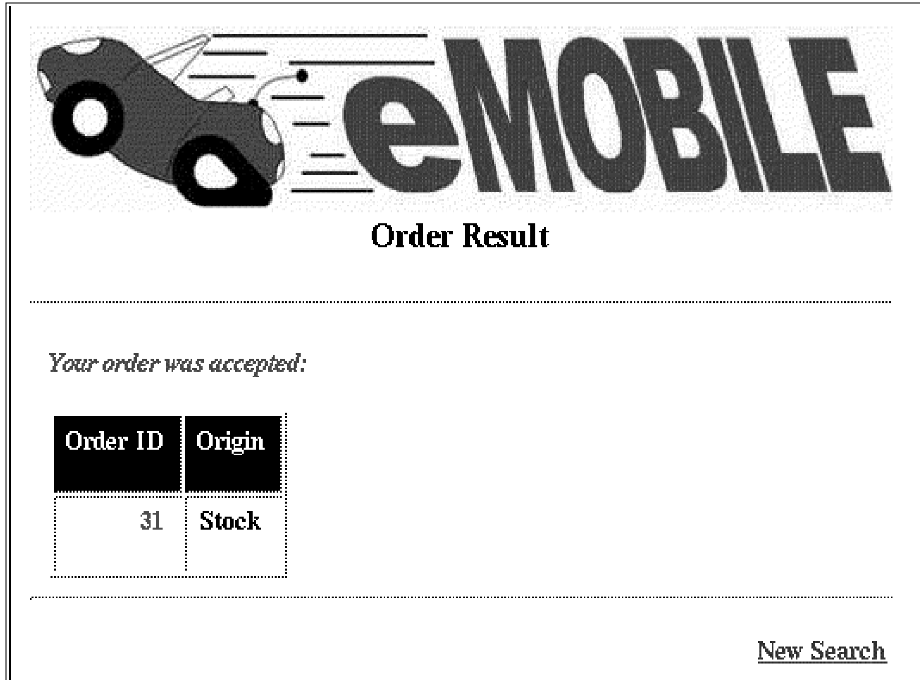
Select the number of the car you want to buy:

#	Color	Make	Model	Price	Quantity	Style	Year	Dealer
1	BLACK	FORD	CONTOUR	15202.87	1	COUPE	1999	MMMcNealy Auto Sales
2	RED	FORD	ESCORT	15609.8	1	SEDAN	1999	MMMcNealy Auto Sales
3	BLACK	FORD	MUSTANG	15898.9	1	SEDAN	1998	Gosling Auto Goods
4	SILVER	FORD	EXPLORER	16306.74	1	COUPE	1998	MMMcNealy Auto Sales
5	BLUE	FORD	TAURUS	16311.91	1	SEDAN	1999	MMMcNealy Auto Sales
6	GREEN	FORD	CONTOUR	16384.62	1	SEDAN	1999	Gosling Auto Goods
7	ICE-BLUE	FORD	TAURUS	16543.24	1	SEDAN	1998	Zander Auto Deals
8	SILVER	FORD	TAURUS	16571.24	1	COUPE	1999	Gosling Auto Goods
9	RED	FORD	ESCORT	16631.78	1	SEDAN	1999	MMMcNealy Auto Sales
10	ICE-BLUE	FORD	TAURUS	16980.76	1	SEDAN	1998	Zander Auto Deals

Results per Page: 10 **Next Results**

New Search

The page generated by `CarSearchServlet` using the `eMobileCarSearch-html.xsl` style sheet and displayed in a Netscape browser



The page generated by `CarBuyServlet` using the `eMobileCarBuy-html.xsl` style sheet and displayed in a Netscape browser

The following code fragment from `eMobileCarBuy-html.xsl` illustrates the general principle of the style sheet transformations (compare this code fragment with its WML counterpart from page 27). The **bold** statements are XSL statements - expressed in the `xsl` namespace (line 3). The remaining statements look like HTML but, since the style sheet is itself a well-formed XML document (line 1), all the HTML tags are properly closed. It's up to the output method (specified by the `xsl:output` statement at line 6) to output the resulting document as a regular HTML page (with all its unclosed tags).

This style sheet document specifies three templates:

- A template that matches the root node of the source document (line 8)
- A template that matches any `OrderResult` node of the source document (line 28)
- A template that matches any `Error` node of the source document (line 56)

```
1. <?xml version="1.0" encoding="UTF-8"?>
2.
3. <xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
4.    <xsl:param name="CarReqAlias" />
5.
6.    <xsl:output method="html" indent="yes"/>
7.
8.    <xsl:template match="/">
9.       <html>
10.         <head>
11.            <title>eMobile</title>
12.         </head>
13.         <body bgcolor="#ffffff">
14.            <table border="0">
15.               ...
16.               <xsl:apply-templates />
17.               <tr>
18.                   <td align="right" colspan="2">
19.                          <hr />
20.                          <b><h3><a href="{$CarReqAlias}">New Search</a></h3></b>
21.                   </td>
22.               </tr>
23.            </table>
24.         </body>
25.      </html>
26.   </xsl:template>
27.
28.   <xsl:template match="OrderResult">
29.      <tr>
30.         ...
31.         <td />
32.         <td>
33.            <table border="1" cellpadding="5">
34.               ...
35.               <tr>
36.                   <td align="right">
37.                          <h3>
38.                                 <font color="red">
39.                                        <xsl:value-of select="OrderID" />
40.                                 </font>
41.                          </h3>
42.                   </td>
43.                   <td align="center">
44.                          <h3>
45.                                 <font color="black">
46.                                        <xsl:value-of select="Origin" />
47.                                 </font>
48.                          </h3>
49.                   </td>
50.               </tr>
51.            </table>
52.         </td>
53.      </tr>
54.   </xsl:template>
55.
56.   <xsl:template match="Error">
57.      <tr>
58.         <td align="center" colspan="2">
59.            <h1>
60.               <xsl:value-of select="." />
61.               <p />
62.            </h1>
63.         </td>
64.      </tr>
65.   </xsl:template>
66.
67.</xsl:stylesheet>
```

eMobileCarBuy-html.xsl style sheet

When the template at line 8 (`<xsl:template match="/">`) is applied to the root node, it instructs the style sheet processor (through the `xsl:apply-templates` statement at line 16) to process the templates in the style sheet that match its child nodes. For the following example of XML source document, only the template matching the `OrderResult` element will be applied.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE CarBuy SYSTEM "eMobileCarBuy.dtd">

<CarBuy>
  <OrderResult>
    <OrderID Type="java.lang.Integer">5069</OrderID>
    <Origin Type="java.lang.String">Stock</Origin>
  </OrderResult>
</CarBuy>
```

Example of an XML source document to be processed by `eMobileCarBuy-html.xsl` style sheet

The `xsl:value-of` statements (at lines 39 and 46 in the style sheet) will be replaced in the output document with the values of the `OrderID` and `Origin` elements: `5069` and `Stock` respectively. Finally, the style sheet parameter `CarReqAlias` declared at line 4 and used at line 20 is instantiated with its actual value `eMobile/CarReq`.

From template matching root element	From template matching `OrderResult` element

From template matching root element:

```
<html>
  <head>
    <title>eMobile</title>
  </head>
  <body bgcolor="#ffffff">
    <table border="0">
      ...
      <tr>
        <td colspan="2" align="right">
          <hr>
          <b>
            <h3>
              <a href="eMobile/CarReq">New
Search</a>
            </h3>
          </b>
        </td>
      </tr>
    </table>
  </body>
```

From template matching `OrderResult` element:

```
<tr>
  <td></td>
  <td>
    <table cellpadding="5" border="1">
      ...
      <tr>
        <td align="right">
          <h3>
            <font color="red">5069</font>
          </h3>
        </td>
        <td align="center">
          <h3>
            <font color="black">Stock</font>
          </h3>
        </td>
      </tr>
    </table>
  </td>
</tr>
```

The resulting HTML document

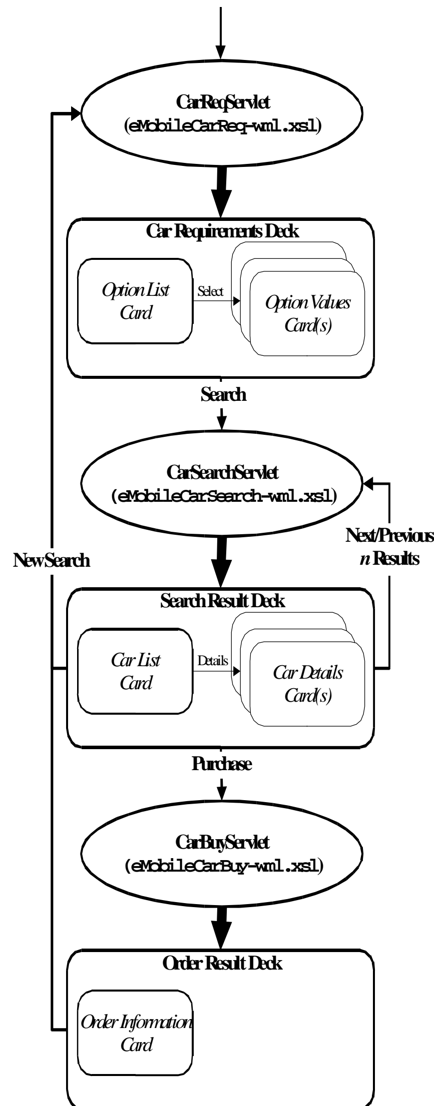
Generating WML content

WML (Wireless Markup Language) is based on XML and is intended to specify the way information is presented on wireless client devices. Similar to HTML, WML specifies user interface components (formatted text, links, text fields, list of options) to support interaction with the user. Those components are laid out within *cards*. A card corresponds to one screen of information to be displayed. To cope with the limited capabilities of the wireless client devices, the content to be displayed may be divided among different cards; and several related cards may be gathered within a *deck* to be sent to the client device as a single HTTP response. A deck contains at least one card. The user can navigate between WML cards much the same way as between HTML pages. Upon loading a deck of cards, the client device display the first defined card.

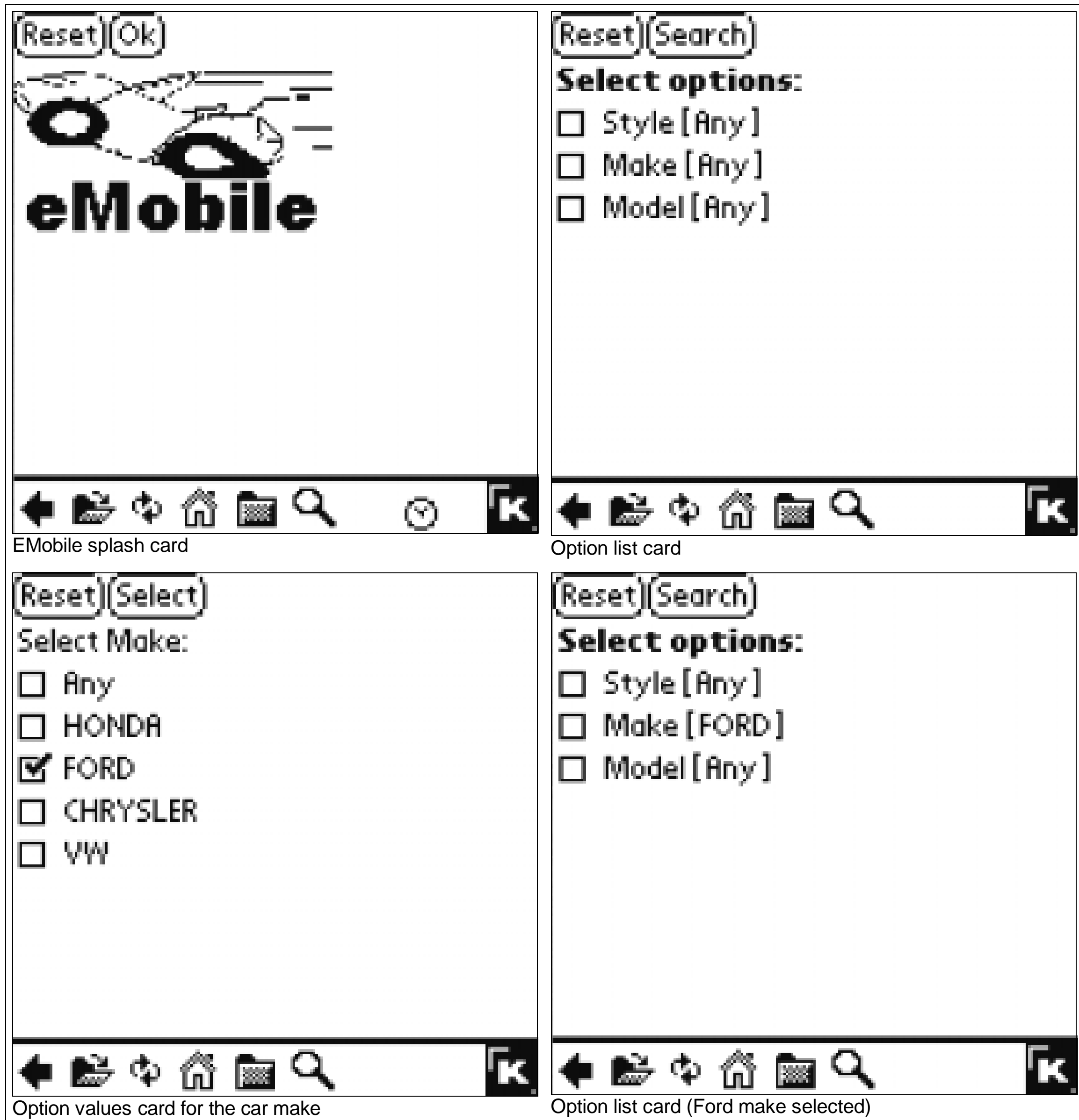
Wireless client devices may have different capabilities and the same cards may be displayed differently depending on the device. Therefore, the WML document to be sent may first have to be specifically

tailored for some devices to render the content appropriately.

The generation of WML content from the the DOM trees is performed by the three servlets `CarReqServlet`, `CarSearchServlet`, and `CarBuyServlet` using the style sheets `eMobileCarReq-wml.xsl`, `eMobileCarSearch-wml.xsl`, and `eMobileCarBuy-wml.xsl` respectively. When applied, each style sheet generates a WML deck.



WML decks/cards workflow



EMobile splash card

Option list card

Option values card for the car make

Option list card (Ford make selected)

The deck of cards generated by `CarReqServlet` using the `eMobileCarReq-wml.xsl` style sheet and displayed in the KBrowser on a Palm device

(Search)(Next)
Results: 0-3/214
☐ ESCORT SEDAN RED 1999
$15609.8
☐ MUSTANG SEDAN BLACK 1998
$15898.9
☐ EXPLORER COUPE SILVER 1998
$16306.74

Car list card (the first three matching cars)

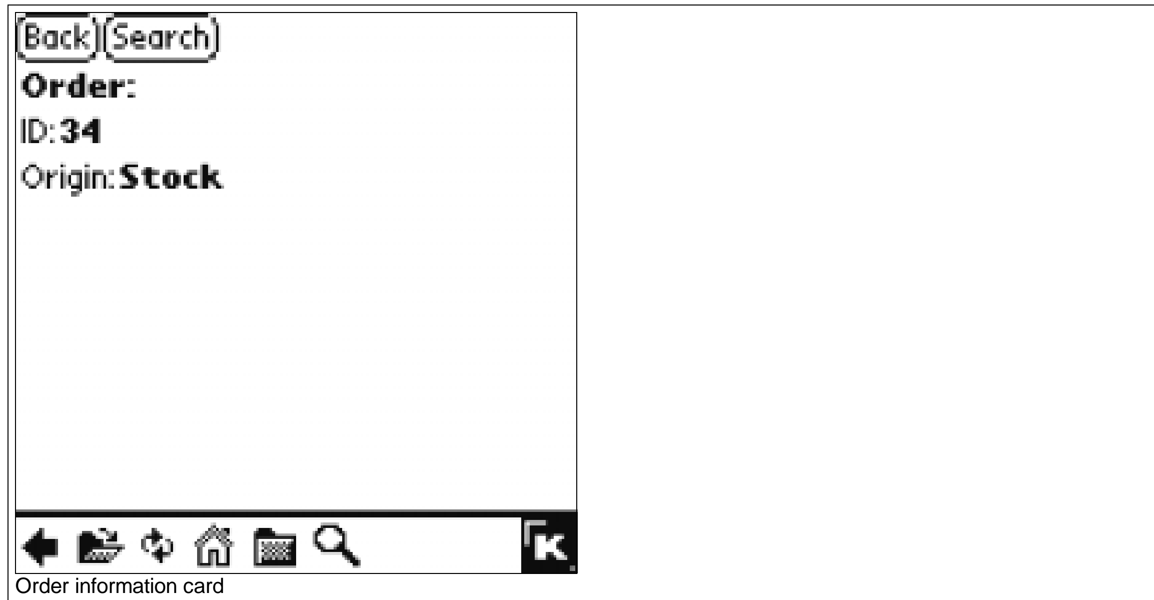
(Search)(Prev)(Next)
Results: 3-6/214
☐ TAURUS SEDAN BLUE 1999
$16311.91
☐ CONTOUR SEDAN GREEN 1999
$16384.62
☐ TAURUS SEDAN ICE-BLUE 1998
$16543.24

Car list card (the next three matching cars)

(Back)(Buy)
Details:
FORD
CONTOUR
SEDAN
GREEN
1999
$16384.62
Gosling Auto Goods
Palo Alto
CA

Car details card

The deck of cards generated by `CarSearchServlet` using the `eMobileCarSearch-wml.xsl` style sheet and displayed in the KBrowser on a Palm device



Order information card

The deck of cards generated by `CarBuyServlet` using the `eMobileCarBuy-wml.xsl` style sheet and displayed in the KBrowser on a Palm device

Similar to the `eMobileCarBuy-html.xsl` style sheet presented above, the following code fragment from `eMobileCarBuy-wml.xsl` specifies three templates:

- A template that matches the root node of the source document (line 9)
- A template that matches any `OrderResult` node of the source document (line 30)
- A template that matches any `Error` node of the source document (line 46)

The **bold** statements are XSL statements - expressed in the `xsl` namespace (line 3). The remaining statements are WML. The output method (specified by the `xsl:output` statement at line 7) instructs the style sheet processor to output the resulting document as an XML document for which the document type will be set for WML 1.1.

This code fragment illustrates as well how the HTTP `User-Agent` field is passed as a parameter to the style sheet (at line 5) and used to conditionally output a WML fragment (at line 18).

```
1. <?xml version="1.0" encoding="UTF-8"?>
2.
3. <xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
4.    <xsl:param name="CarReqAlias" />
5.    <xsl:param name="User-Agent" />
6.
7.    <xsl:output method="xml" doctype-public="-//WAPFORUM//DTD WML 1.1//EN" doctype-
   system="http://www.wapforum.org/DTD/wml_1.1.xml" indent="yes"/>
8.
9.    <xsl:template match="/">
10.      <wml>
11.         <head>
12.            <meta forua="true" http-equiv="Cache-Control" content="max-age=0"/>
13.         </head>
14.         <card>
15.            <do type="accept" label="Back">
16.               <prev />
17.            </do>
18.            <xsl:if test="contains($User-Agent, 'KBrowser')">
19.               <do type="" label="Search">
20.                   <go method="get" href="{$CarReqAlias}" />
21.               </do>
22.            </xsl:if>
23.            <p mode="nowrap">
24.               <xsl:apply-templates />
25.            </p>
26.         </card>
27.      </wml>
28.   </xsl:template>
29.
30.   <xsl:template match="OrderResult">
31.      <b>
32.         <xsl:text>Order:</xsl:text>
33.      </b>
34.      <br />
35.      <xsl:text>ID: </xsl:text>
36.      <b>
37.         <xsl:value-of select="OrderID" />
38.      </b>
39.      <br />
40.      <xsl:text>Origin: </xsl:text>
41.      <b>
42.         <xsl:value-of select="Origin" />
43.      </b>
44.   </xsl:template>
45.
46.   <xsl:template match="Error">
47.      <b>
48.         <xsl:value-of select="." />
49.      </b>
50.   </xsl:template>
51.
52.</xsl:stylesheet>
```

`eMobileCarBuy-wml.xsl` style sheet

The following code fragment illustrates how the HTTP `Accept` header field is used to select images in a format supported by the client devices.

```
<xsl:param name="accept" />
...
<img alt="eMobile">
   <xsl:attribute name="src">
      <xsl:choose>
         <xsl:when test="contains($accept, 'image/vnd.wap.wbmp')">J2EECar.wbmp</xsl:when>
         <xsl:otherwise>J2EECar.bmp</xsl:otherwise>
      </xsl:choose>
   </xsl:attribute>
</img>
```

Using the HTTP Accept header field

The number of cars matching the user requirements may be bigger than what can fit in one deck. Therefore, special care has been taken to:

- Maximize the number of car descriptions that can fit in one deck by removing redundant or irrelevant information
- Split the result of the search among several decks and allow the user to browse by downloading one deck at a time

The following code fragments illustrate those two features:

```
<xsl:variable name="Requirements" select="concat('Make=', $Make, ',', 'Model=', $Model,
',', 'Style=', $Style, ',', 'Color=', $Color, ',', 'Year=', $Year, ',')" />
...
<xsl:for-each select="Description/*[not(self::VIN)]">
   <xsl:if test="not(contains($Requirements, concat(name(), '='))) or
contains($Requirements, concat(name(), '=Any'))">
      <xsl:value-of select="concat(., ' ')" />
   </xsl:if>
</xsl:for-each>
```

Filtering out redundant information: the criteria of the search are not displayed since it's the same information for all the matching cars; the car description elements are here compared to the user's requirements and output only if they are not part of the requirements (i.e. the search criteria). For example, if the user searched for all Ford Mustang Coupe, the make "Ford", the model "Mustang" and the style "Coupe" will not be displayed.

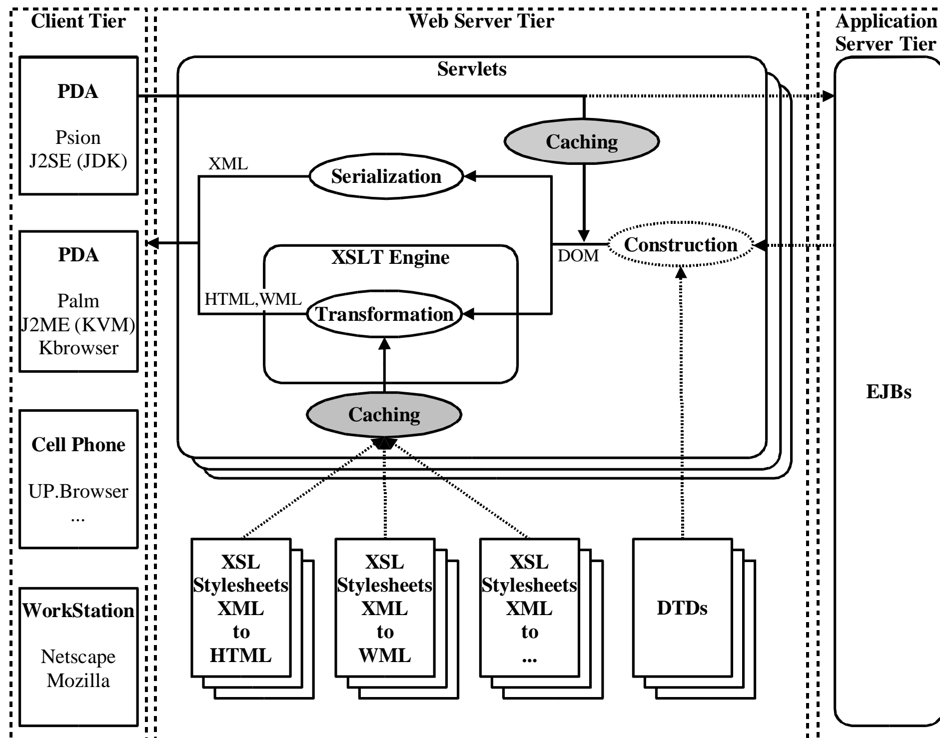
```
<xsl:param name="SortedBy" />
<xsl:param name="ResultsPerPage" />
<xsl:variable name="ResultIndex">...</xsl:variable>
...
<xsl:for-each select="//Car">
   <xsl:sort select="descendant::*[name() = $SortedBy]" />
   <xsl:sort select="Price" />
   <xsl:call-template name="Cards" />
</xsl:for-each>


<xsl:template name="Cards">
   <xsl:if test="position() &gt; $ResultIndex and position() &lt;= ($ResultIndex +
$ResultsPerPage)">
      <card id="{concat('car', position())}" >
         ...
      </card>
   </xsl:if>
</xsl:template>
```

Splitting the result among several decks: only the cars from the current index up to the maximum number of cars that can fit in a deck are included in the generated deck; controls are added to allow the user to browse from one set of cars (deck) to the next/previous.

Performance Improvement

There are two places where the performance of the web tier of the eMobile sample application has been improved:

1. The construction of DOM trees from the value objects returned by the EJB application
2. The loading of style sheets from files



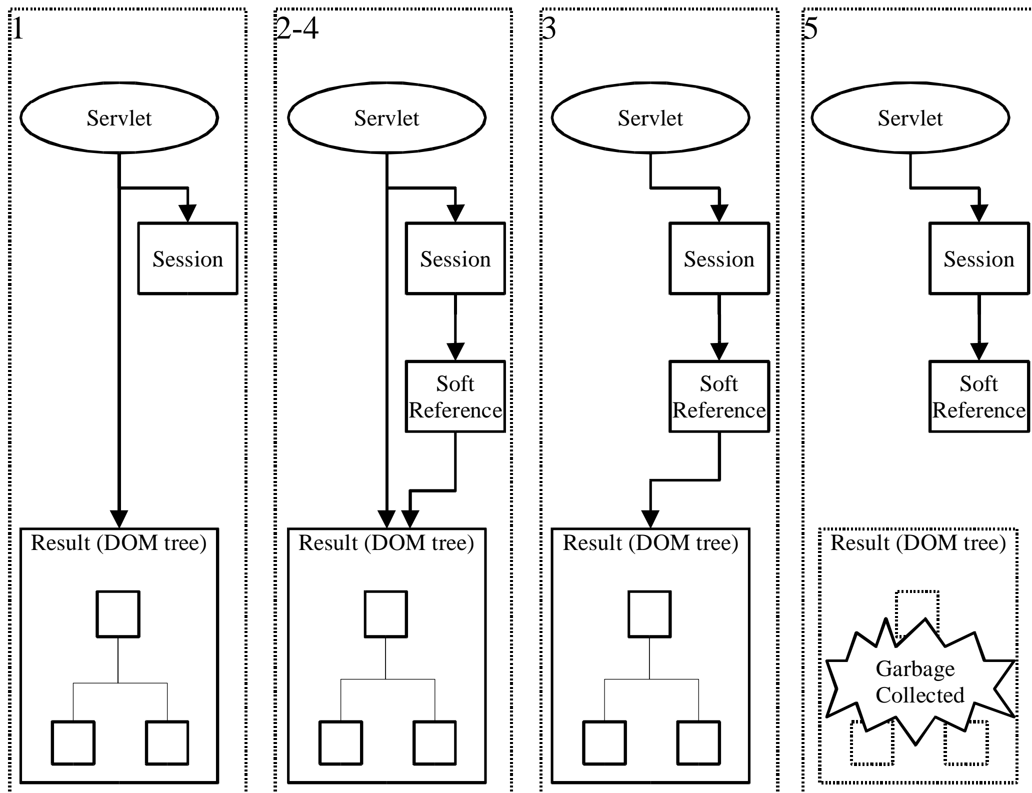
The two places where performance has been improved

When the number of cars matching the user requirements cannot fit in one WML deck or in one HTML page, the result is divided among several decks or pages to allow the user to browse the overall result. The decks or pages are generated one at a time upon the user's request, from the DOM tree. To avoid invoking the EJB application again, the DOM tree is cached in the user's session. When loaded, the style sheets are also cached to avoid reloading them for every content generation.

Caching the DOM Trees

Caching the result of a user request to serve subsequent related requests more quickly consumes memory. It must not be done to the detriment of the other users: the application must not fail because of memory shortage due to the cached results. Soft references introduced with Java 2 allow interacting with the garbage collector to implement caches.

The following diagram illustrates how soft references were used to implement a cache. For more information on the soft references and the definitions of reachability, refer to the Java 2, Standard Edition API documentation.



This diagram describes five states (assuming no other references to the Result (DOM tree) than those described):

1. The servlet processes the user's request and creates the Result (DOM tree) from the value objects returned by the EJB application: the Result (DOM tree) is strongly reachable.
2. The servlet creates a soft reference to the Result (DOM tree) and adds it as an attribute of the user's session: the Result (DOM tree) is still strongly reachable.
3. The servlet ended processing the user's request and may process another user's request: the Result (DOM tree) is softly reachable - only reachable through the soft reference assuming that the servlet does not keep any reference to the Result (DOM tree) while inactive or processing another request.
4. The servlet retrieves the Result (DOM tree) from the user's session and processes it further: the Result (DOM tree) becomes strongly reachable again.
5. The servlet ended processing the user's request and may process another user's request: the Result (DOM tree) is again softly reachable. As memory is used up, the Garbage Collector reclaims the Result (DOM tree) which is softly reachable: the Result (DOM tree) is no longer reachable.

The Java virtual machine guarantees that all soft references to softly-reachable objects have been cleared before throwing an `OutOfMemoryError`.

The cached Result (DOM tree) is reused only if the corresponding query matches the query being currently processed, otherwise it is discarded from the cache.

The caching of the Result (DOM tree) is managed through three methods of the AbstractCarServlet class, namely, cacheQueryResult, getCachedQueryResult, and clearCachedQueryResult.

The AbstractCarServlet cacheQueryResult method stores the query and its result in the client's associated session so that the response to the same query will be immediate. A soft reference is used so that if the memory is running low the GC may reclaim the query and the result objects.

```
private class CacheEntry {
   Object query;
   Object result;

   CacheEntry(Object query, Object result) {
      this.query = query;
      this.result = result;
   }
}

protected void cacheQueryResult(Object query, Object result, HttpSession session) {
   session.setAttribute(QUERY_ATTRIBUTE, new SoftReference(new CacheEntry(query,
result)));
   return;
}
```

Caching the query and its result in the client's associated session

The AbstractCarServlet getCachedQueryResult method gets the result to a previously executed query that has been cached in the session with the cacheQueryResult method. The previously cached result may be returned only if the cached query and the requested query match. Since soft references are used, a matching result may not be returned if the query itself, or the result, has already been reclaimed by the GC due to a shortage of memory.

```
protected Object getCachedQueryResult(Object query, HttpSession session) {
   SoftReference reference = (SoftReference) session.getAttribute(QUERY_ATTRIBUTE);
   if (reference != null) { // A reference to cached entry was retrieved
      CacheEntry entry = reference.get();
      if (entry != null) { // The referred cache entry was not reclaimed by the GC
         Object cachedQuery = entry.query;
         Object cachedResult = entry.result;
         if (cachedQuery.equals(query)) { // The cached query and the requested query
match
            if (TRACE) {
               System.err.println("Query cache hit.");
            }
            return cachedResult;
         } else if (TRACE) {
            System.err.println("Query cache miss (mismatch).");
         }
      } else if (TRACE) {
         System.err.println("Query cache miss (GC).");
      }
   } else if (TRACE) {
      System.err.println("Query cache miss (session).");
   }
   // The queries didn't match or the cached query itself and its result could not be
retreived, let's just clean
   session.removeAttribute(QUERY_ATTRIBUTE);
   return null;
}
```

Getting the result to a previously executed query that has been cached in the session

The `AbstractCarServlet` `clearCachedQueryResult` method removes the query and its associated result, if any (i.e. if it has not been garbage-collected) cached in the specified session.

```
protected void clearCachedQueryResult(HttpSession session) {
   session.removeAttribute(QUERY_ATTRIBUTE);
   return;
}
```

Removing the query and its associated result from the user's session

Caching the Style Sheets

Caching the style sheets relies on the same principle as caching the result DOM trees. The style sheets when loaded are cached in a hash table using soft references. The hash table is shared among all the servlets (i.e. the instances of the sub-class of `AbstractCarServlet`). The style sheets are assumed to be thread-safe. The style sheet cache is managed by the `AbstractCarServlet` `getStylesheet` method (discussed on page 13).

```
private static Hashtable stylesheets = new Hashtable();

private Object getStylesheet(String mimeType) throws SAXException {
   Object stylesheet = null;
   String path = getStylesheetPath(mimeType);
   // Even if the stylesheets Hashtable get and put methods are synchronized, the
   // following block is synchronized to avoid multiple cache miss.
   synchronized (stylesheets) {
   // Try to get the required stylesheet from the set of stylesheets already compiled.
      SoftReference reference = (SoftReference) stylesheets.get(path);
      if (reference != null) { // Got a soft reference to the stylesheet, let's get the
actual stylesheet.
         stylesheet = reference.get();
         if (TRACE && stylesheet == null) {
            System.err.println("Stylesheet cache miss (GC).");
         }
      } else if (TRACE) {
         System.err.println("Stylesheet cache miss (...).");
      }
      if (stylesheet == null) { // The stylesheet has been reclaimed by the GC or was
never created, let's create a brend new one
         // Compile the required stylesheet (the one associated with the response media
type).
         stylesheet = adapter.loadStylesheet(path);
         // Cache the stylesheet for future use, using a soft reference so that the GC
may reclaim it if it's not referenced anymore
         // and if the memory is running low.
         if (stylesheet != null) {
            if (CACHE) {
               stylesheets.put(path, new SoftReference(stylesheet));
            }
         }
      } else if (TRACE) {
         System.err.println("Stylesheet cache hit.");
      }
   }
   return stylesheet;
}
```

Gets the style sheet associated with the specified MIME type either from the cache or from the local file system

The Configuration Demonstrated

The following configuration was used to create the eMobile sample application's client and web tiers:

Location	*Hardware*	*Software*	*Functionality Demonstrated*
Web Tier	Sun™ Ultra™30 (Solaris 8)	J2EE Reference Implementation 1.2.1	Servlets and EJB
		Xalan-j 1.0 /Xerces1.0.3	XML, XSLT
		Saxon 5.0 /JAXP	XML, XSLT
Client Tier	Sun Ultra 30 (Solaris 8)	Mozilla 6.0 beta	HTML
		Yospace SmartPhone Emulator1.0	WML
	Palm IIIxe	J2ME (KVM)	WML
		Kbrowser1.0 (Java and native)	
	Psion Series 5mx	JDK™ 1.1.4	XML, applet
		JAXP	
Connectivity	Clarinet Systems EthIR LAN		PDA Infrared Connectivity

Conclusion

The two white papers "Part1: Using Servlet, Applet, and XML Technologies" and "Part 2: Using Servlet, XML, and XSLT Technologies" demonstrate that Java and XML are complementary technologies (on all tiers of an enterprise application). eMobile is an end-to-end Java application using XML technologies to address the inherent diversity of the client devices on the internet, including those connected from wireless networks. The eMobile sample application particularly illustrates the ubiquity of Java from the server side with J2EE to the client side with both J2SE and J2ME.

The message regarding XML and Java association and partnership is: Portable data and portable behavior. While J2EE allowed the deployment of the eMobile application onto different vendor server implementations, J2ME and J2SE allowed it to be run or accessed from different client devices. With XML as the primary data format, it not only ensured the portability of the data but also provided a powerful and flexible way to transform it to address a rich diversity of client devices. By using XML, eMobile was created as an open application - it could be accessed directly by a variety of end user's devices, or by intermediary services such as search or shopping agents that may easily further process the XML based content.

References and Resources

eMobile End-to-End Application Using the Java™ 2 Platform, Enterprise Edition, Integrating an Enterprise Application with Various Client Devices, Part1: Using Servlet, Applet, and XML Technologies: http://developer.java.sun.com/developer/technicalArticles/javaone00/eMobileApplet.pdf

4thpass Kbrowser: http://www.4thpass.com/kbrowser/download/

Apache XML Parser for Java - Xerces: http://xml.apache.org/xerces-j/index.html

Apache XSLT Style Sheet Engine - Xalan: http://xml.apache.org/xalan/index.html

Clarinet System EthIR LAN: http://www.clarinetsys.com

Psion Series 5mx: http://www.psion.com/

Saxon XSLT Style Sheet Engine: http://users.iclway.co.uk/mhkay/saxon/

Sun's Java API for XML Parsing (JAXP): http://java.sun.com/xml/download.html

The WAP Forum Ltd: http://www.ericsson.se/WAP

W3C XML Architecture: http://www.w3.org/XML/

Yospace SmartPhone Emulator: http://www.yospace.com/