Java/XML-based Shopping Cart

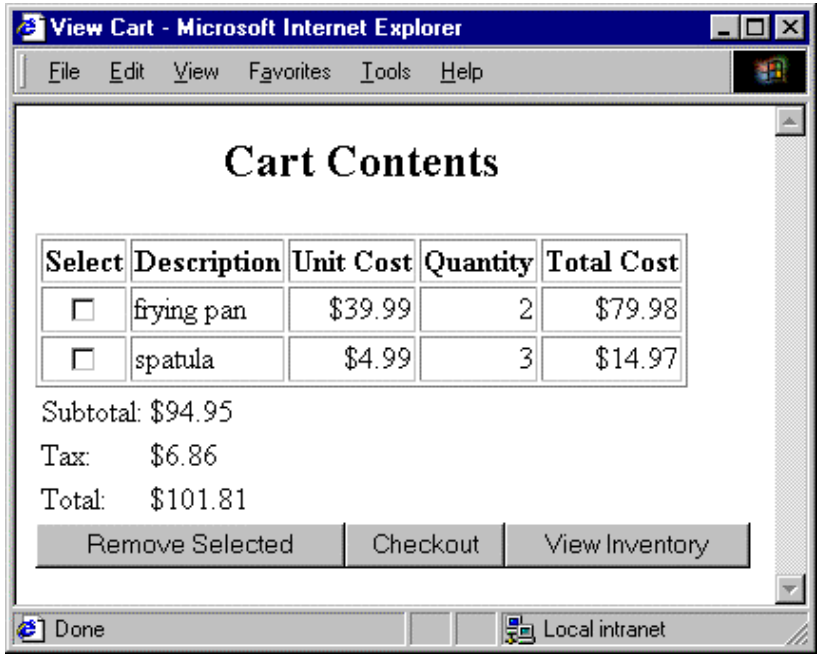
Presented by

Mark Volkmann
mark@ociweb.com
Object Computing, Inc. (OCI)

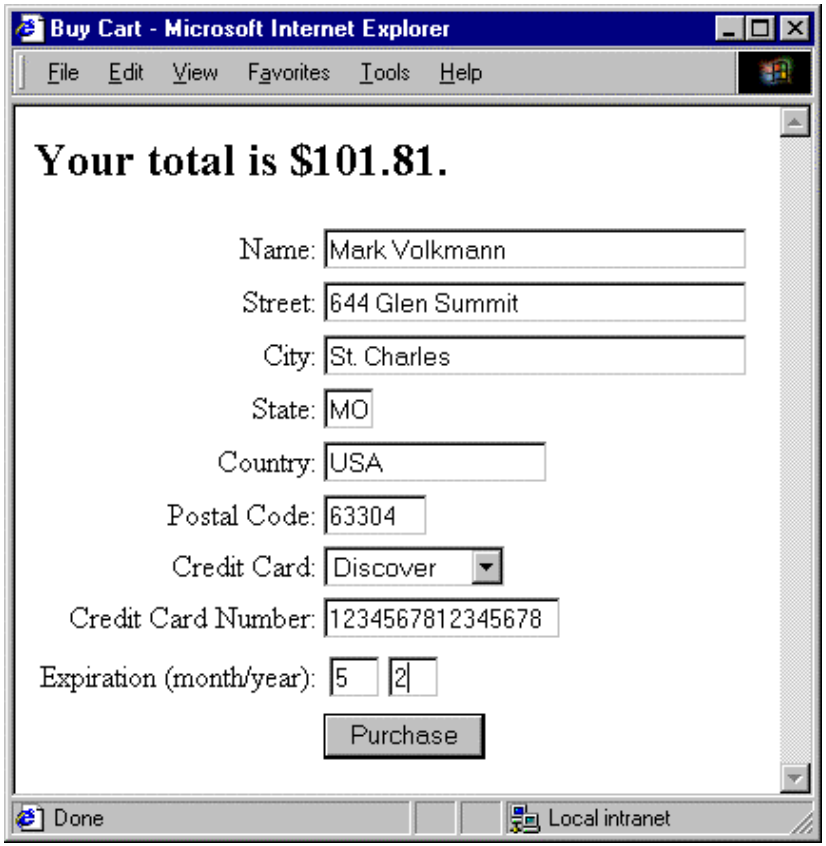
Overview

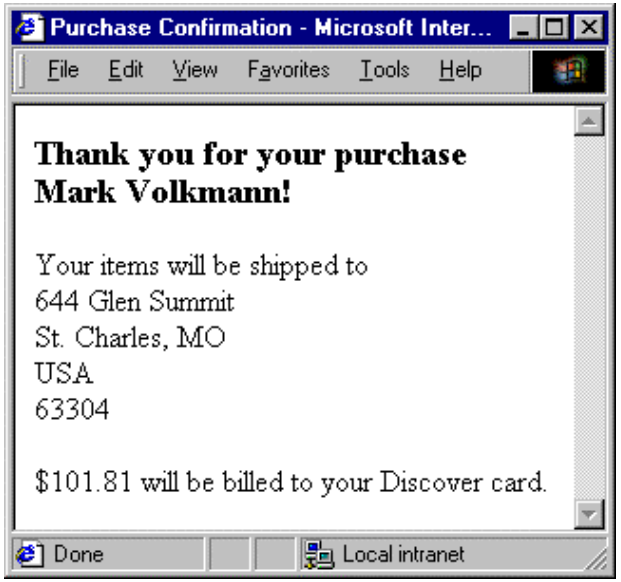
- Goal
  - create a prototype of a web-based shopping cart application that utilizes
    - Java servlets
    - XML and XSLT
    - HTML and JavaScript
- Tools
  - Servlet Engine
    - Apache Tomcat 3.1
  - XML Parser
    - Apache Xerces - 1.1.3
  - XSLT Processor
    - Apache Xalan 1.0.1
  - Java Development Kit
    - Sun JDK 1.2.2

Screen Shots



Screen Shots





High Level Design

- All client requests go through the GatewayServlet
- The GatewayServlet directs each request to a specific UseCaseController
- A UseCaseController interacts with any number of domain objects
- Domain objects can populate themselves from databases or XML documents
  - an improvement would be to move this functionality to other classes that function as data adapters
- Domain objects can create XML representations of themselves
  - an improvement would be to move this functionality to other classes that function as data adapters

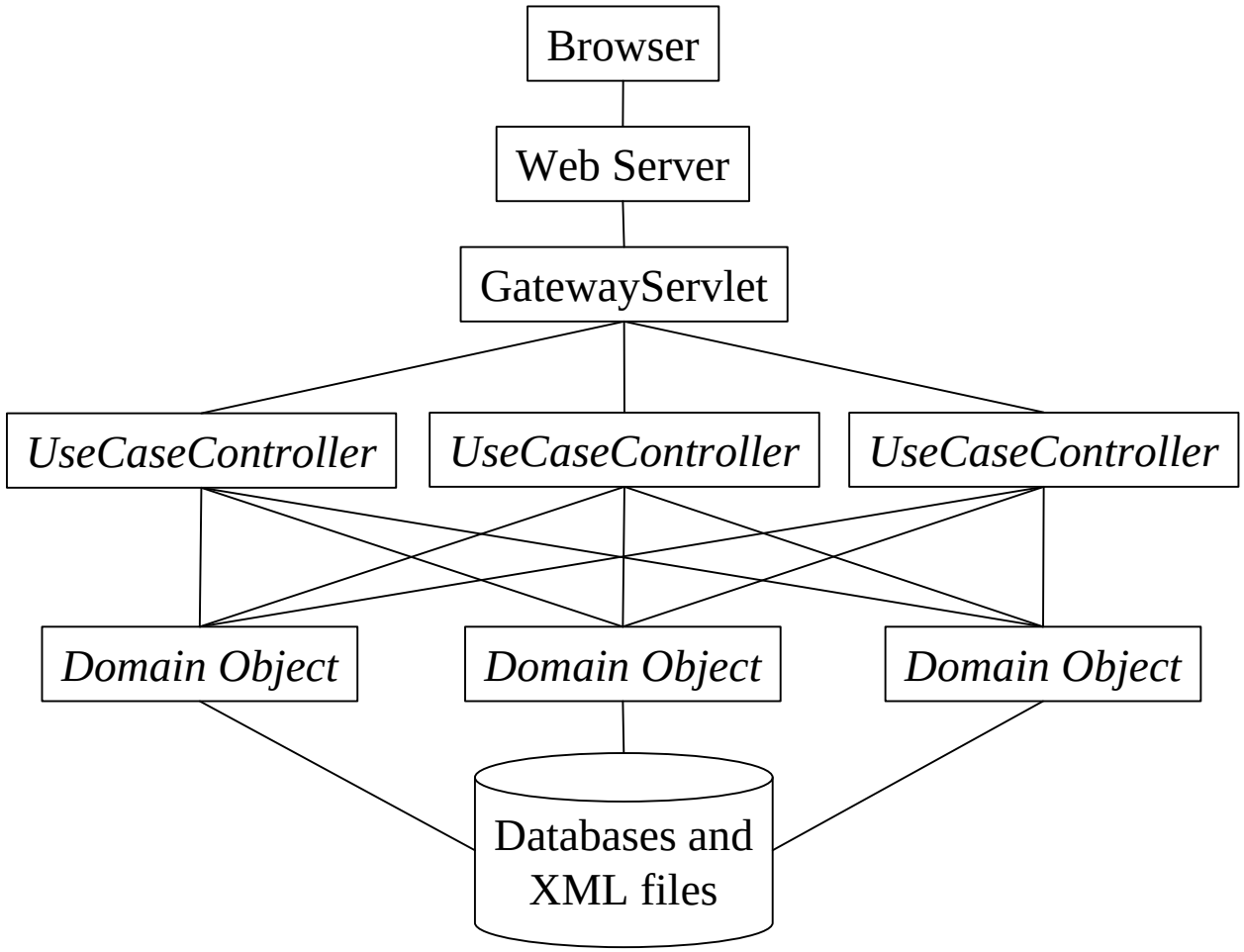
Use Cases

- View Inventory
  - display current inventory
  - allow user to enter item quantities desired
  - related files: ViewInventoryController.java, inventory.xsl
- Add To Cart
  - remove selected items from inventory
  - add selected items to user's cart
  - invoke View Cart use case
  - related files: AddToCartController.java
- View Cart
  - display contents of user's cart
  - calculate total cost of cart including tax
    - done in XSLT stylesheet
  - related files: ViewCartController.java, cart.xsl

Use Cases (Cont'd)

- Remove From Cart
  - remove selected items from user's cart
  - add selected items back to inventory
  - invoke View Cart use case
  - related files: RemoveFromCartController.java
- Buy Cart
  - gather information from user that is needed to process the transaction
    - includes name, address and credit card information
  - related files: BuyCartController.java, buy.xsl
- Process Payment
  - process credit card transaction
    - not actually done in sample code
  - display confirmation message
  - related files: ProcessPaymentController.java, purchase.xsl

Interactions



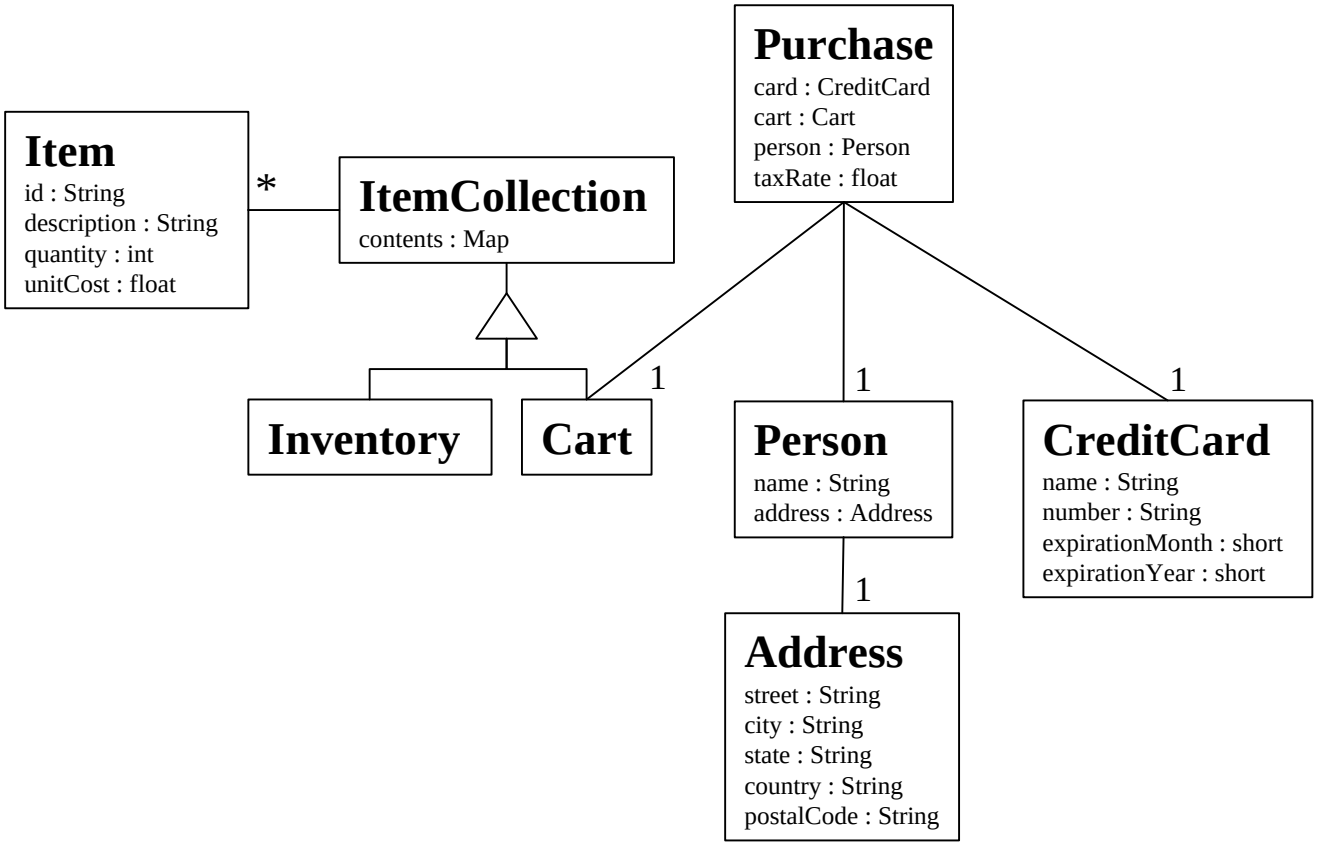
Domain Objects

- Item
  - holds a description, id, quantity and unit cost of an item in the inventory or a user's cart
- ItemCollection
  - holds a collection of Item objects
- Cart
  - extends ItemCollection
  - holds the Items currently in a user's cart
- Inventory
  - extends ItemCollection
  - holds the Items currently in the inventory

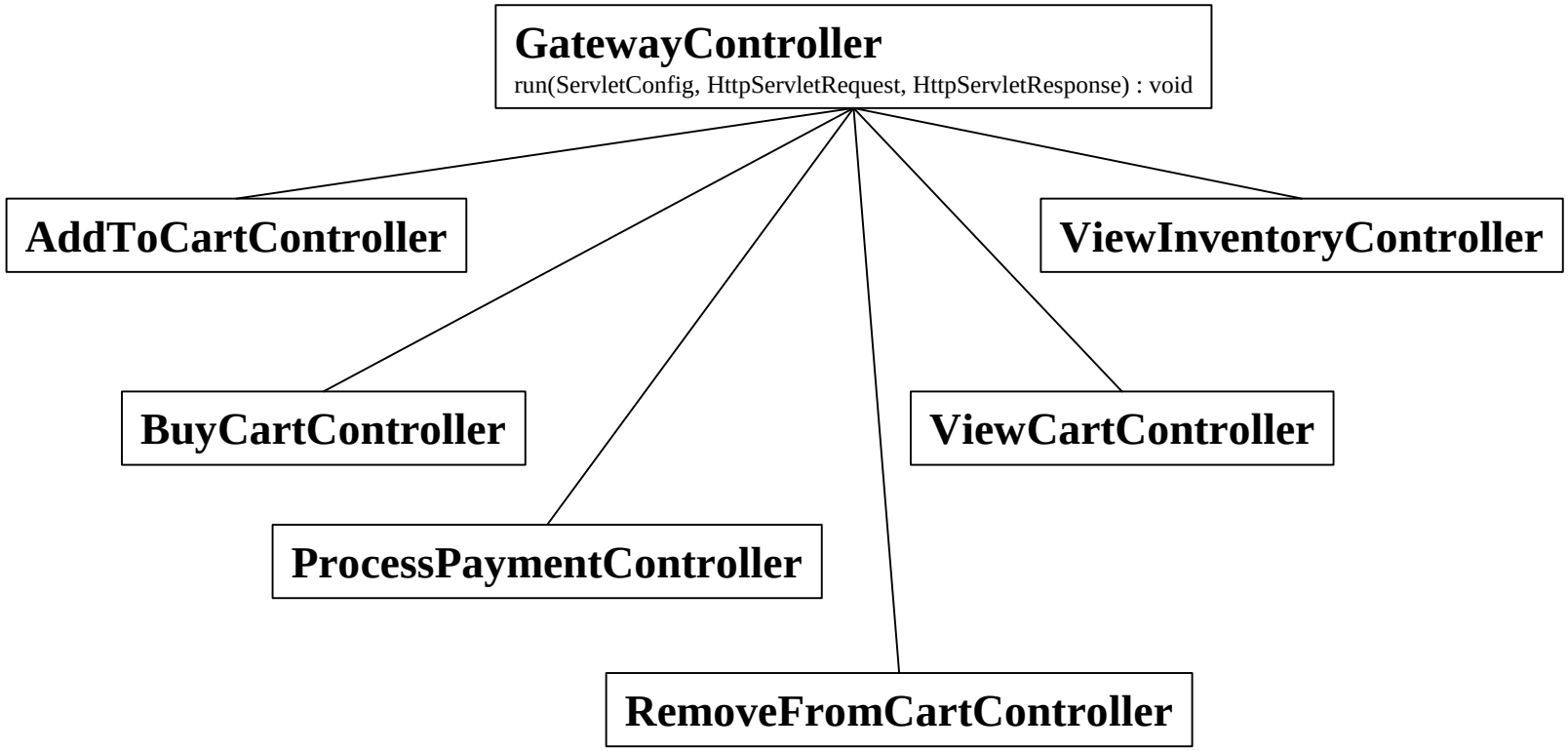
Domain Objects (Cont'd)

- Address
  - holds a street, city, state, country and postal code
  - used for user addresses
- Person
  - holds the name and Address of a person
- CreditCard
  - holds the name, number and expiration date of a credit card
- Purchase
  - holds the Person, CreditCard, Cart and tax rate associated with a purchase

Domain Objects (Cont'd)



Controllers



GatewayServlet Overview

- Creates a Map from use case names to UseCaseController objects
  - only happens on the first request
  - mapping is described in UseCases.xml
    - more on this later
  - a use case name is passed as an initialization parameter to GatewayServlet
  - the use case name is a key in a Map of UseCaseControllers
  - this Map is placed in an HttpSession attribute so that controllers can locate other controllers
  - useful when one controller needs to call another one to finish processing a request
    - AddToCartController calls ViewCartController after adding items to the cart
    - RemoveFromCartController calls ViewCartController after removing items from the cart

GatewayServlet Overview (Cont'd)

- Looks up the UseCaseController that corresponds to the requested use case name
- Invokes the run method of the UseCaseController
- This would be a good place to add
  - authentication
    - if the user has no current session then a login page could be displayed
  - authorization
    - GatewayServlet could restrict the use cases that a given user can execute

HttpSession Attributes

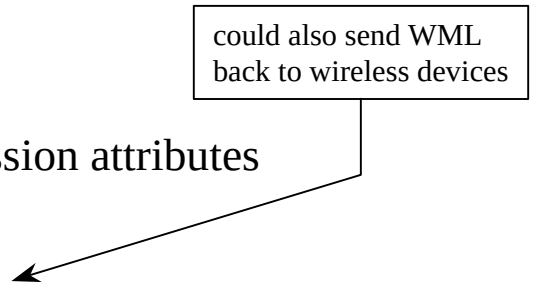
- Used to store
  - the UseCaseController objects
    - in a Map keyed on use case name
  - the Inventory
    - in a Map keyed on item id
    - a subclass of ItemCollection
  - the Cart
    - also in a Map keyed on item id
    - another subclass of ItemCollection

UseCaseController Interface

```
package com.ociweb.framework;
import java.io.IOException;
import javax.servlet.http.*;

public interface UseCaseController {
    void run(ServletConfig config,
             HttpServletRequest request,
             HttpServletResponse response) throws IOException;
}
```

- ServletConfig
  - provides access to the ServletContext which can be used to read files in within the directory structure of the web app.
- HttpServletRequest
  - provides access to form parameters and the HttpSession object which contains a map of session attributes
- HttpServletResponse
  - provides access to the Writer needed to send HTML back to the client



ViewInventoryController Overview

- This is the first UseCaseController that is invoked
- If no Cart exists yet, it creates an empty Cart
- If no Inventory exists yet, it creates an empty Inventory and populates it
  - currently the data comes from inventory.xml
    - see next page
  - it really should come from a database
- Creates a DOM Document object representing the current inventory as XML
- Transforms it to HTML by applying an XSLT stylesheet
  - inventory.xsl

inventory.xml

```
<inventory>
  <item>
    <id>1</id>
    <description>frying pan</description>
    <cost>39.99</cost>
    <quantity>5</quantity>
  </item>
  <item>
    <id>2</id>
    <description>spatula</description>
    <cost>4.99</cost>
    <quantity>7</quantity>
  </item>
</inventory>
```

XMLUtil Overview

- Provides static methods that make it easier to build DOM XML Document objects
  - these return the new element so child elements can be added to it

```
Element appendChildElement(Node parent,
                           String childElementName)

Element appendChildElementWithText(Node parent,
                                   String childElementName,
                                   String text)

Element appendChildElementWithText(Node parent,
                                   String childElementName,
                                   float value)

Element appendChildElementWithText(Node parent,
                                   String childElementName,
                                   int value)
```

XMLUtil (Cont'd)

- Provides a static method that applies an XSLT stylesheet to a DOM Document

```
void applyStylesheet(Document xmlDoc, String xslURL, Writer out)
void applyStylesheet(Document xmlDoc, InputStream xsl, Writer out)
```

- Provides a static method that gets the value of a named child element within a given element

```
String getChildValue(Element element, String childName)
```

- Provides static methods that pretty-print a DOM Document

`void outputXML(Document doc)` - sends to System.out for debugging

```
void outputXML(Document doc, OutputStream out)
```

Mapping of Servlet Names to Use Case Names

- Tomcat web applications use web.xml to specify servlet names and their initialization parameters
- Each use case has a servlet mapping like the following

```
<servlet>
  <servlet-name>AddToCart</servlet-name>
  <servlet-class>
    com.ociweb.framework.GatewayServlet
  </servlet-class>
  <init-param>
     <param-name>useCase</param-name>
     <param-value>Add To Cart</param-value>
   </init-param>
</servlet>
```

Mapping of Servlet Names to Use Case Names (Cont'd)

- There is one servlet mapping for each use case
- All of them use GatewayServlet to service the request
- An initialization parameter passed to GatewayServlet identifies the use case being requested
- Here are the URLs for running locally using the default Tomcat port
  - http://localhost:8080/framework/servlet/AddToCart
  - http://localhost:8080/framework/servlet/BuyCart
  - http://localhost:8080/framework/servlet/RemoveFromCart
  - http://localhost:8080/framework/servlet/PurchaseCart
  - http://localhost:8080/framework/servlet/ViewCart
  - http://localhost:8080/framework/servlet/ViewInventory

Mapping of Use Cases to UseCaseController Objects

- Use case names are mapped to UseCaseController classes by another XML file (UseCases.xml)

```
<useCases>
  <useCase>
    <name>Add To Cart</name>
    <controller>
      com.ociweb.shopping.AddToCartController
    </controller>
  </useCase>
  <!-- Entries for the other use cases are similar. -->
</useCases>
```

- Usage of this was described earlier
  - see page titled “GatewayServlet Overview”

Sample UseCaseController
(ViewCartController)

```
package com.ociweb.shopping;

import com.ociweb.framework.*;
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import org.xml.sax.SAXException;
```

continued on next page

used to produce the right screen shot on page 3

Sample UseCaseController (Cont'd)

```
public class ViewCartController implements UseCaseController {

    public void run(ServletConfig config,
                    HttpServletRequest request,
                    HttpServletResponse response)
    throws IOException {
        Writer out = response.getWriter();

        HttpSession session = request.getSession();
        Cart cart = (Cart) session.getAttribute("cart");

        XMLUtil.outputXML(cart.getXML()); // for debugging
```

Sample UseCaseController (Cont'd)

```
        try {
            ServletContext context = config.getServletContext();
            InputStream is = context.getResourceAsStream("/WEB-INF/cart.xsl");
            XMLUtil.applyStylesheet(cart.getXML(), is, out);
        } catch (SAXException e) {
            out.write(e.toString());
        }
    }
}
```

reads file from the directory of the web app.
(in my case, C:\Tomcat\webapps\framework)

Sample XSLT Stylesheet
(cart.xsl)

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0”
 xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:variable name="taxRate" select="0.07225"/>

  <xsl:template match="/">
    <html>
      <head>
        <meta http-equiv="Pragma" content="no-cache"/>
        <title>View Cart</title>
      </head>
      <body>
        <form
         action="http://localhost:8080/framework/servlet/RemoveFromCart"
         method="post">
```

produces the right screen shot on page 3

prevents old cart quantities for being displayed when this page is revisited after removing items from the cart

Sample XSLT Stylesheet (Cont'd)

```
<xsl:choose>
  <xsl:when test="items/item">
    <table border="1">
      <caption>
        <h2>Cart Contents</h2>
      </caption>
      <tr>
        <th>Select</th>
        <th>Description</th>
        <th>Unit Cost</th>
        <th>Quantity</th>
        <th>Total Cost</th>
      </tr>
      <xsl:apply-templates select="items/item">
        <xsl:sort select="description"/>
      </xsl:apply-templates>
    </table>
```

testing whether cart is empty

table headers

- outputs a table row for each item in the cart sorted on description
- uses template on page 31

Sample XSLT Stylesheet (Cont'd)

```
<xsl:variable name="subtotal"
 select="sum(items/item/totalCost)"/>
<table>
  <tr>
    <td>Subtotal:</td>
    <td>
      <xsl:value-of
       select="format-number($subtotal, '$#.##')"/>
    </td>
  </tr>
  <tr>
    <td>Tax:</td>
    <td>
      <xsl:value-of select=
       "format-number($subtotal * $taxRate, '$#.##')"/>
    </td>
  </tr>
```

calculates the total before tax

calculates the tax

Sample XSLT Stylesheet (Cont'd)

```
      <tr>
        <td>Total:</td>
        <td>
          <xsl:value-of select=
       "format-number($subtotal * (1 + $taxRate), '$#.##')"/>
        </td>
      </tr>
    </table>

    <input value="Remove Selected" type="submit"/>
    <input value="Checkout" type="button"
     onclick="document.location=
      'http://localhost:8080/framework/servlet/BuyCart';"/>
  </xsl:when>

  <xsl:otherwise>
    <p>Your cart is currently empty.</p>
  </xsl:otherwise>
</xsl:choose>
```

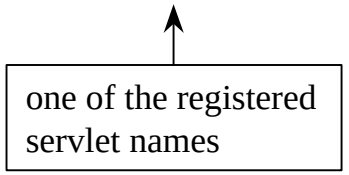
calculates the total including tax

posts form fields to form action

one of the registered servlet names

Sample XSLT Stylesheet (Cont'd)

```
        <input value="View Inventory" type="button"
         onclick="document.location=
          'http://localhost:8080/framework/servlet/ViewInventory';"/>
      </form>
    </body>
  </html>
</xsl:template>
```



Sample XSLT Stylesheet (Cont'd)

```
  <xsl:template match="item">
    <tr>
      <td align="center">
        <input name="Remove {id}" type="checkbox"/>
      </td>
      <td align="left"><xsl:value-of select="description"/></td>
      <td align="right">
        <xsl:value-of select="format-number(unitCost, '$#.##')"/>
      </td>
      <td align="right"><xsl:value-of select="quantity"/></td>
      <td align="right">
        <xsl:value-of select="format-number(totalCost, '$#.##')"/>
      </td>
    </tr>
  </xsl:template>

</xsl:stylesheet>
```

- outputs a table row for the current cart item
- see apply-templates on page 27

Deficiencies in the Prototype

- Controllers are dependent on the Servlet API
  - implementing controllers so they are unaware that they are being invoked from a servlet is difficult
  - some controllers need to
    - access form parameters in the HttpServletRequest
    - send output to the client using information in the HttpServletResponse
    - access and update session-level data in the HttpSession
    - read files in the directory of the web app. using the HttpContext

Deficiencies in the Prototype (Cont'd)

- Inventory persistence
  - the inventory is reloaded from an XML document every time the servlet engine is restarted
  - the inventory is stored in a session-scoped object so multiple users are not supported
  - one solution would to be to modify the Inventory class so that it interacts directly with the database in every get and set method
    - as is done in a container-managed entity EJB

Source Code

- Complete source code is available in a separate zip file
  - ShoppingFramework.zip
  - setup steps are on the next page
- Source code for couple of key classes follows

Setup Steps

- Here are the setup steps to run the provided code
  - installations
    - install Tomcat 3.1
    - install JDK 1.2 or 1.3
  - environment variables
    - insure that JAVA_HOME points to the JDK is installed
    - insure that TOMCAT_HOME points to where TOMCAT is installed
  - copy framework.war to %TOMCAT_HOME%\webapps
  - modify %TOMCAT_HOME%\bin\tomcat.bat so CLASSPATH contains xerces.jar BEFORE %TOMCAT_HOME%\lib\xml.jar
    - xml.jar is Sun's Project X XML parser
  - start Tomcat
  - point browser to http://localhost:8080/framework/servlet/ViewInventory

GatewayServlet

```
package com.ociweb.framework;

import java.io.*;
import java.util.*;
import javax.servlet.*;
import javax.servlet.http.*;
import org.apache.xerces.parsers.DOMParser;
import org.w3c.dom.*;
import org.xml.sax.*;

public class GatewayServlet extends HttpServlet {

    private ServletConfig config;

    public void doGet(HttpServletRequest request,
                      HttpServletResponse response)
    throws IOException, ServletException {
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();

        config = getServletConfig();
```

GatewayServlet (Cont'd)

```
    HttpSession session = request.getSession();
    Map controllers = (Map) session.getAttribute("controllers");
    if (controllers == null) {
        controllers = getControllers();
        session.setAttribute("controllers", controllers);
    }

    String useCase = getInitParameter("useCase");
    UseCaseController controller =
        (UseCaseController) controllers.get(useCase);

    if (controller == null) {
        out.write("Fatal error: No controller found for " + useCase);
    } else {
        controller.run(config, request, response);
    }
}
```

GatewayServlet (Cont'd)

```
public void doPost(HttpServletRequest request,
                   HttpServletResponse response)
throws IOException, ServletException {
    doGet(request, response);
}

private static Map getControllers() {
    Map controllers = new HashMap();

    ServletContext context = config.getServletContext();
    InputStream is = context.getResourceAsStream("/WEB-INF/UseCases.xml");
```

GatewayServlet (Cont'd)

```
        try {
            DOMParser parser = new DOMParser();
            parser.parse(new InputSource(is));
            Document doc = parser.getDocument();

            NodeList useCases = doc.getElementsByTagName("useCase");
            int count = useCases.getLength();
            for (int i = 0; i < count; i++) {
                Element useCase = (Element) useCases.item(i);
                String name = XMLUtil.getChildValue(useCase, "name");
                String controller = XMLUtil.getChildValue(useCase, "controller");
                Class controllerClass = Class.forName(controller);
                controllers.put(name, controllerClass.newInstance());
            }
        } catch (Exception e) {
            System.err.println(e);
        }
        return controllers;
    }
}
```

XMLUtil

```
package com.ociweb.framework;

import java.io.*;
import org.apache.xalan.xslt.*;
import org.apache.xerces.parsers.DOMParser;
import org.apache.xml.serialize.*;
import org.w3c.dom.*;
import org.xml.sax.SAXException;

public class XMLUtil {
    /**
     * Note: parent must already be directly or indirectly appended
     * to a Document before calling this.
     */
    public static Element appendChildElement(Node parent,
                                             String childElementName) {
        Document doc = parent instanceof Document ?
            (Document) parent : parent.getOwnerDocument();
        Element childElement = doc.createElement(childElementName);
        parent.appendChild(childElement);
        return childElement;
    }
```

XMLUtil (Cont'd)

```
public static Element appendChildElementWithText(Node parent,
                                                  String childElementName,
                                                  String text) {
    Document doc = parent.getOwnerDocument();
    Element childElement = doc.createElement(childElementName);
    parent.appendChild(childElement);

    Text textNode = doc.createTextNode(text);
    childElement.appendChild(textNode);

    return childElement;
}
```

XMLUtil (Cont'd)

```
public static Element appendChildElementWithText(Node parent,
                                                  String childElementName,
                                                  float value) {
    return appendChildElementWithText
        (parent, childElementName, String.valueOf(value));
}

public static Element appendChildElementWithText(Node parent,
                                                  String childElementName,
                                                  int value) {
    return appendChildElementWithText
        (parent, childElementName, String.valueOf(value));
}
```

XMLUtil (Cont'd)

```
public static void applyStylesheet(Document xmlDoc,
                                   String xslURL,
                                   Writer out)
throws SAXException {
    XSLTProcessor processor = XSLTProcessorFactory.getProcessor();
    XSLTInputSource xmlSource = new XSLTInputSource(xmlDoc);
    XSLTInputSource xslSource = new XSLTInputSource(xslURL);
    XSLTResultTarget resultTarget = new XSLTResultTarget(out);
    processor.process(xmlSource, xslSource, resultTarget);
}

public static void applyStylesheet(Document xmlDoc,
                                   InputStream xslInputStream,
                                   Writer out)
throws SAXException {
    XSLTProcessor processor = XSLTProcessorFactory.getProcessor();
    XSLTInputSource xmlSource = new XSLTInputSource(xmlDoc);
    XSLTInputSource xslSource = new XSLTInputSource(xslInputStream);
    XSLTResultTarget resultTarget = new XSLTResultTarget(out);
    processor.process(xmlSource, xslSource, resultTarget);
}
```

XMLUtil (Cont'd)

```
public static String getChildValue(Element element, String childName) {
    Node node = element.getFirstChild();
    while (node != null) {
        if (node instanceof Element &&
            node.getNodeName().equals(childName)) {
            return node.getFirstChild().getNodeValue();
        }
        node = node.getNextSibling();
    }
    return null;
}
```

XMLUtil (Cont'd)

```
    public static void outputXML(Document doc) {
        outputXML(doc, System.out);
    }

    public static void outputXML(Document doc, OutputStream out) {
        OutputFormat format = new OutputFormat("xml", "UTF-8", true);
        format.setIndent(2);
        XMLSerializer serializer = new XMLSerializer(out, format);

        try {
            serializer.serialize(doc);
        } catch (IOException e) {
            System.err.println(e);
        }
    }
}
```