Building rich web applications using ILOG JViews and evolved technologies such as JSF and AJAX

Mathias Mouly
Technical Account Manager
mmouly@ilog.fr
Learn how to build advanced 2D graphics web interface using Java Server Faces, AJAX and ILOG Visualization Technology
Agenda

- Introducing JSF technology
- Building JSF advanced 2D graphics components
- Few words about ILOG JViews
- JSF JViews Chart component example with DHTML rendering
- JSF/AJAX to improve user experience
- Summary
Introduction - JSF

• JSR 127
• A component framework for building user interfaces for web applications
• A set of APIs for representing UI components, managing state, handling events, inputs, and defining page navigation
• A JavaServer™ Pages (JSP) core tag library
• A set of predefined server-side components and an HTML render kit implementation
JSR 127: JavaServer Faces

Features

- A well defined request processing lifecycle
- Managed UI Component state across requests
- Event model that allows the application to write server-side handlers
- Multiple render kits for one component (HTML, DHTML, SVG, Flash, ...)
- Navigation between pages
- Managed beans
- Bindings or Value Expressions (1.2)
- Input value validation and conversion
- Multiple implementations of the specification
JSR 127 : JavaServer Faces

Request Processing Lifecycle
Introduction - JSF

- **Basic Use**
  - Develop the model objects and register them
  - Create the JSP pages
  - Define page navigation

- **Customizing**
  - Customize converters and listeners
  - I18N

- **Extending**
  - Adding new render kits
  - Defining new components
package myPackage;

public class Customer {
    private String name;

    public void setName(String name) {
        this.name = name;
    }

    public String getName() {
        return this.name;
    }
}
<%@ taglib uri="http://java.sun.com/jsf/html" prefix="h" %>
<%@ taglib uri="http://java.sun.com/jsf/core" prefix="f" %>

<html>
<head>
<title>Enter your name</title>
</head>
<body>
  <f:view>
    <h:form id="myForm">
      Enter your name:
      <h:inputText value="#{theCustomer.name}" />
      <h:commandButton action="send" value="Submit" />
    </h:form>
  </f:view>
</body>
</html>
http://localhost:8080/demo1

DEMO
Advanced 2D Graphics Components

- An advanced 2D graphics component goes beyond simple UI components to provide enhanced feedback and interaction to the user
  - e.g. tooltips, zooming, data selection, data edition
- Examples:
  - Charts components
  - Map components
- Why should you care?
  - To display new types of data
  - To improve user experience on all kinds of data
Advanced 2D Graphics Components

SVG + Servlet/JSP Examples
Advanced 2D Graphics Components

- Enable DHTML or SVG components for the client-side display and interaction
- JSF is the standardized server-side framework
  - To generate the client representation
  - To handle and validate user inputs
  - To manage the component state
  - To synchronize with the business data model
  - ...
- IDE support
Advanced 2D Graphics Components

- Create a JSP tag handler and its descriptor
- Create the server-side component classes
- Create the renderer(s)
- Register the components and the renderers
Few words about ILOG JViews

Diagrammer

Maps

Maps for Defense

Gantt

Charts

JTGO
DEMO
JViews Charts

Charts

- Interactive Charting
  - Point, line, bar, bubble, etc.
  - Radar, pie, polar

- Features:
  - 2D and 3D rendering
  - Innovative interactions
  - Powerful MVC architecture
  - Real-time redraw capability
  - Connectors for XML, JDBC, and Swing table provided
JViews Charts: Principles

- MVC architecture
  - Data and graphical representations are separated
  - Data object / chart displayer object

- Data aware charts
  - Data changes are automatically reflected in the charts that display these data
  - Modifications made by interacting with a chart are automatically reflected on the data
• **Data model designed for integration**
  • Open API for adaptation to any backend
  • Data sets
    • Array of points, cyclic array, function, combinations, load-on-demand, …
  • Data sources
    • Memory, input, XML, JDBC, CSV, Swing table, …
JViews Charts : The Designer
JViews DHTML components

Architecture

- JavaScript API
- Image Servlet from Java Graphic component
Purpose

- Make developing Web Application with JViews easier
- Leverage the DHTML components with JSF
- JSF component view manages the graphic object
- DHTML renderers render HTML + JavaScript
- Heavily subclassed image servlet support
  - The servlet must be subclassed by the customer as a last resort
JViews JSF DHTML components

Architecture
public class ChartTag extends UIComponentTag {
    public String getComponentType() {
        return "ChartComponent";
    }
    public String getRendererType() {
        return "ChartRenderer";
    }
    protected void setProperties(UIComponent c) {
        super.setProperties(c);
        if (url != null) {
            c.getAttributes().put("url", url);
            // ...
        }
    }
    public void setUrl(String url) {
        this.url = url;
    }
    public void setDataSourceID(String dataSource) {
        this.dataSource = dataSource;
    }
    public void setType(String type) {
        this.type = type; // PIE, BAR...
    }
}
public class ChartComponent extends UIGraphic {
    public void setDataSourceID(String dataSource) {
        this.dataSource = dataSource;
    }
    public Object saveState(FacesContext ctx) {
        Object values[] = new Object[3];
        values[0] = super.saveState(context);
        values[1] = dataSource;
        // ...;
        return values;
    }
    public void restoreState(FacesContext ctx, Object state) {
        Object[] values = (Object[]) state;
        super.restoreState(context, values[0]);
        setDataSource((String) values[1]);
        // ...;
    }
}
public class ChartRenderer extends Renderer {
    public void decode(FacesContext ctx, UIComponent c) {
        super.decode(ctx, c);
        Map parameterMap =
            ctx.getExternalContext().getRequestParameterMap();
        String url = (String)map.get("url");
        if (url != null)
            ((ChartComponent)c).setUrl(url);
        // ...
    }
    public void encodeEnd(FacesContext ctx, UIComponent c)
        throws IOException {
        super.encodeEnd(ctx, c);
        ChartComponent chart = (ChartComponent)c;
        ResponseWriter writer = ctx.getResponseWriter();
        writer.startElement("img");
        writer.writeAttribute("src", chart.getUrl());
        writer.endElement("img");
    }
}
The image generator Servlet uses a Chart rendering engine that dumps drawings to a Graphics2D.

A Graphics2D is obtained from a BufferedImage and passed to the Chart engine.

The BufferedImage is encoded according to the required image type and sent back to the client.
<%@ taglib uri="http://java.sun.com/jsf/html" prefix="h" %>
<%@ taglib uri="http://java.sun.com/jsf/core" prefix="f" %>
<%@ taglib uri="http://www.ilog.com/jviews/tlds/jviews-chart-faces.tld" prefix="c" %>

<html>
  <body>
    <f:view>
      <c:XMLElement id="myDataSource" filename="data.xml" />
      <c:chartView id="chart" style="width:800;height:600;"
        dataSourceId="myDataSource" />
    </f:view>
  </body>
</html>
http://localhost:8080/myJSF/faces/chart-user1.jsp

DEMO
Adding interaction

```xml
<c:XMLDataSource id="myDataSource" filename="data.xml" />

<c:chartView id="chart" style="width:800;height:600;"
    dataSourceId="myDataSource" interactorId="zoom" />

<c:chartZoomInteractor id="zoom" />
```
http://localhost:8080/myJSF/faces/chart-user9.jsp
Adding CSS rendering flexibility

```xml
<jvcf:chartView id="chart8" project="data/test2.icpr"
    style="width:1000;height:800;"
    binding="#{chartBean.chartView}"/>

<h:commandButton value="Reset"
    actionListener="#{chartBean.reset}"/>
```
http://localhost:8080/myJSF/faces/chart-user11.jsp
AJAX

- XMLHttpRequest + XML + JavaScript
- IFrames are working too!
- Security problems for the two solutions
  - ActiveX
  - Some browsers don’t like IFrames
- Allows to send requests and process response asynchronously
- Cool …. But no standards
- New issues
  - History & bookmarks
- Real solutions are proprietary
JSF / AJAX Strategies

- **External servlet**
  - Use a dedicated servlet out of the JSF lifecycle to provide the component with its update data
  - Ex: Popup menu
  - Ex: RWC data servlet

- **Servlet Filter**
  - Use the JSF lifecycle
  - Add a servlet filter that cuts the unnecessary information

- **Client side parsing (IBM JavaOne)**
  - Use the JSF lifecycle
  - Parse on the client
JSF / AJAX Strategies

• Phase listener (JViews RWC)
  • Register a phase listener that intercepts the AJAX request
  • Render the update data
  • Cut the JSF lifecycle

• Complete architecture (ex ADF)
  • Proprietary components that handles incremental refresh at the base
Summary

- Advanced graphics components need a server-side framework such as JSF
- JSF based components can be easily assembled in JSF-enabled IDEs
- Think to JViews for advanced 2D Graphics usage
For More Information

- **JavaServer™ Faces**

- **Java and SVG**
  - Batik Toolkit: [http://xml.apache.org/batik](http://xml.apache.org/batik)
  - Bring SVG power to Java™ applications: [http://java.sun.com/developer/technicalArticles/GUI/svg](http://java.sun.com/developer/technicalArticles/GUI/svg)
  - ILOG JViews: [http://jviews.ilog.com](http://jviews.ilog.com)

- **SVG**
  - [http://www.w3.org/Graphics/SVG](http://www.w3.org/Graphics/SVG)
Next Steps

• Visit the ILOG booth in the exhibition space for further information

• Evaluate ILOG JViews

• Try the online tutorials

• Look at the demos