SVG-Paint: Java Server Faces, Facelets and Web 2.0 for SVG

Wolfgang Pfnür

Abstract: Java Server Faces (JSF) is a MVC (Model View Control) framework for (dynamic) Web-Pages. Facelets is using x-Html pages and Templates to replace JSP (Java Server Pages) for Java Server Faces. It completes the picture and creates a REAL MCV concept (One could use java code within JSP pages). This document tries to explain how JSF with Facelets works, and give a few short directions on how to get it to work. It will also cover how Web 2.0 can be used, and what this means for SVG (Scalable Vector Graphics). All Example-Code within this document is taken from the SVG-Paint Project.

Introduction

SVG-Paint is a web-based painting program based on SVG and Javascript. It features serveral pre-defined forms as rectangular, circle and line, as well as the possibility to draw free-hand. Drawn objects can be changed, deleted and - in the case of the pre-defined forms - moved and zoomed. Paintings can be saved and loaded. To understand how SVG-Paint works, one first has to understand the techniques used for creating it, which will be presented below.

1 Java Server Faces with Facelets

MVC

The Model View Control concept tries to seperate the code into 3 layers, so each part of it can be changed easily without affecting the others:

- Model. Here resides everything that actually does the work - all the functions, and all the data structures.
- View. This layer is the one the user can see - it displays the results of the program.
- Control. This last layer controls the workflow of the program. It calls the functions in the model, and uses the functions from the view to display the data.

1.1 Model

The Model part is done by "Managed" Beans. The faces-config.xml (Located in the "WEB-INF" directory) defines these Managed Beans.

```
<managed-bean>
<managed-bean-name>Paintings</managed-bean-name>
<managed-bean-class>Paintings</managed-bean-class>
<managed-bean-scope>session</managed-bean-scope>
</managed-bean>
```

The managed-bean-name is the name used to reference the bean from within your .xhtml pages. The managed-bean-class is the name of the class-file to be used. The managed-bean-scope defines, how long the bean should persist. Possible values are: session, request

The Faces/Facelet Engine allows those beans member-variables to be accessed as simple as typing Paintings.names - for this to work you need to have Getters and Setters defined for all the member-variables you want to use (These have to follow the JavaCodeConventions - i.e. setNames and getNames for variable names).

It is worth noting that one doesnt have to just return the variable - you can also execute some logic beforehand (as done with getNames in Paintings.java)

These variables now can be accessed by using the special `#{Paintings.names}` syntax. This only works within special Server-Faces tags! For example, if one wants an image using a path defined by a bean, one needs to use `<h:graphicImage>` instead of the simple `<img>` tag. Further information about Facelets can be found at [Hoo]

1.2 View

When using Facelets, x-Html is used instead of JSP. One can define a template in the folder "templates". There, one can define placeholders like the following:

```
<title><ui:insert name="pageTitle">Page Title</ui:insert></title>
```

Within the actual pages, one can reference a template by using

```
<ui:composition template="/templates/common.xhtml">
```

One can then set the placeholder "pageTitle" within the `<ui:composition>` tags:

```
<ui:define name="pageTitle">SVG-Paint</ui:define>
```

There is no way to use Java-Code, one can only use the Facelet and the Server-Faces and JSTL-core tags. These supply basic functionality like loops (iterating over Arrays/Collections) with the `<ui:repeat> or the <c:forEach>` tags. One can also use conditional code with the `<c:if>` tag. The `<h:commandLink> and <h:commandButton>` tags are used to start a new action.

1.3 Control

In the web.xml (WEB-INF directory) the “Faces Servlet” is defined. All .jsf Requests are mapped to this Servlet. The Servlet looks up the faces-config.xml, and reads out the navigation-rules. Navigation Rules are formatted like the following:

```
<navigation-rule>
<from-view-id>/pages/main.xhtml</from-view-id>
<navigation-case>
<from-outcome>showFiles</from-outcome>
<to-view-id>/pages/load.xhtml</to-view-id>
</navigation-case>
</navigation-rule>
```

All requests coming from *main.xhtml* with the parameter *action* defined as *showFiles* will be redirected to the page *load.xhtml*. This *action-parameter* is defined by using a *commandLink* or *commandButton* as explained above. If needed, one can define other Servlets in the *web.xml* - for example one could define a Servlet to handle XMLhttpRequest, like it was done in SVG-Paint as one can see on Figure 1 on page 4.

2 Web 2.0

2.1 My Definition

Web 2.0 is all about interactivity without having to reload pages in between. Most of the Logic is done through Javascript, and is made viewable with DOM-Tree Manipulation. If additional Data is needed (or should be saved), the XMLhttpRequest object can fulfill this duty.

An exact (and much longer) attempt to define Web 2.0 can be found at [O’R]

2.2 DOM

Dom means Document Object Model. It basically is a tree that mirrors the xml document. This tree can be manipulated - for instance by javascript - to create, change and delete nodes (a node would be one element from start to end tag). This is only done in the clients RAM by the browser - no interaction with the server is needed!

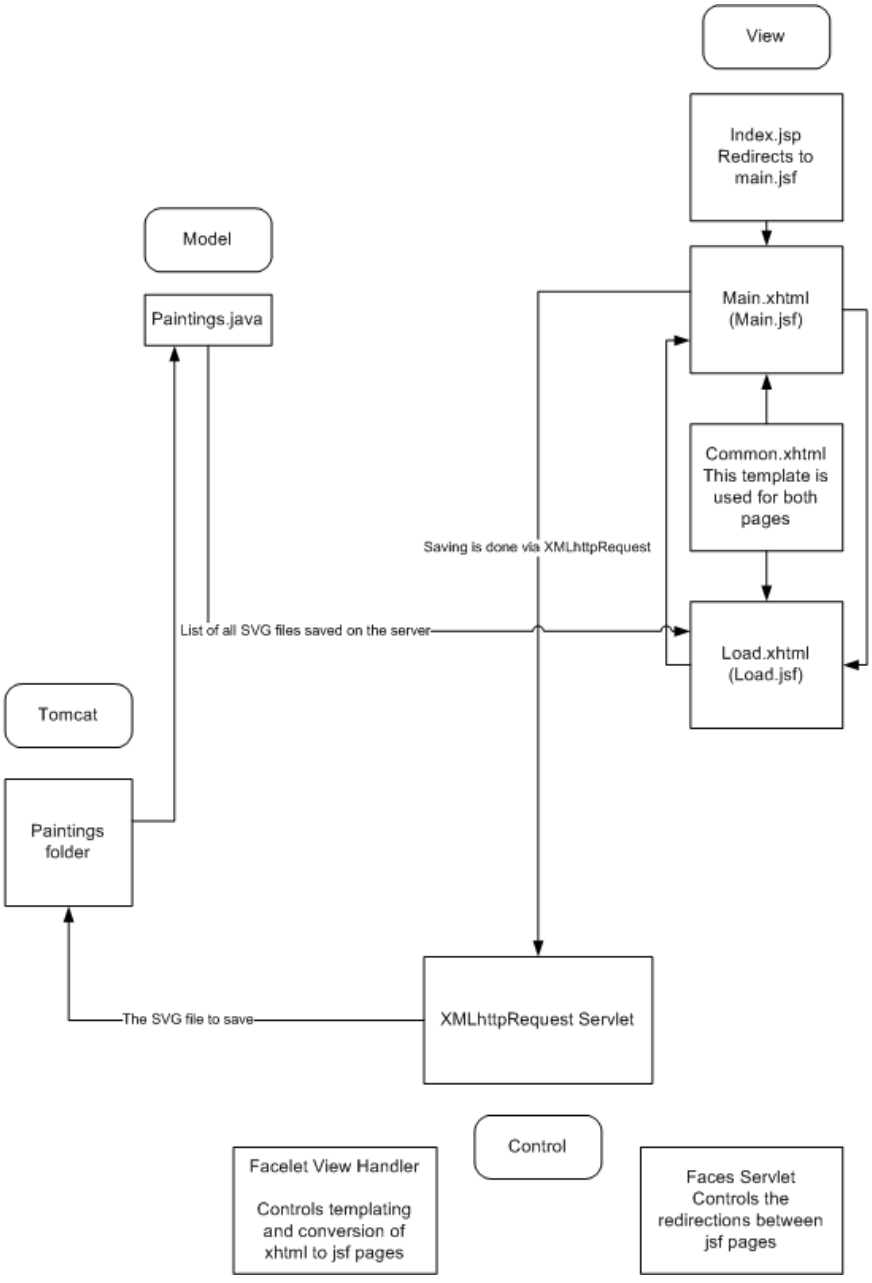

Figure 1: MVC in SVG-Paint

2.3 XMLhttpRequest

For big interactive sites, it would be a hassle to download the complete site before starting to use it, especially if most of that site never shows because it is hidden. To avoid this problem, the XMLhttpRequest object got designed - it can load new content upon request, without reloading the whole page. Only the needed part is loaded! This can be done either synchronous (the script stops until the loading is done) or asynchronous (then one can use the function "onReadyStateChange" to start a function as soon as the loading is done). This can not only be used to load new content, it can also be used to send content to the server.

2.4 SVG

SVG is a XML language that is just starting to gain momentum. It is a Vector-Graphics format, meaning that you can zoom without loss of quality. It is supported by the newest versions of Firefox (1.5.x, 2.x) and Opera (9.x) without the need for a plugin. Internet Explorer and Netscape need a plugin to be able to display SVG content, for example Adobe SVG-Viewer. SVG is a standard maintained by the WWW consortium (W3C). The formal definition can be found at [W3C] As SVG is a XML language, a SVG object can be modified using DOM-Manipulation! This means, one can dynamically change pictures on a site, for example according to theme, without having to keep different versions of the pictures!

3 SVG-Paint

3.1 Introduction

SVG-Paint was created as a showcase for Web2.0 and Java Server Faces. Due to the fact that tomcat cannot understand JSP 2.1 as of now, it was needed to use the Facelets engine on top of JSF. While using Facelets (mainly templating mechanism, action handler - good for projects with many different sites and complex navigation) and Web2.0 (using javascript and dom-manipulation in combination with the XMLhttpRequest object to download/upload instead of reloading and new pages, thus having only very few pages) yields little initial returns, it makes the project more easily expandable (cf. "Possibilities for further development"). The starting point for my developments was an example showcase [Smi], which already had a functioning ant-file. It was a minimal working set, which still showed how JSF works.

3.2 Structure

Upon deployment of the SVG-Paint.war tomcat extracts the SVG-Paint folder, with the subfolders META-INF (standard), pages (which contains all xhtml and js. files), templates (which contains the templates in xhtml format), WEB-INF (contains web.xml, faces-config.xml, has subfolders classes (servlet and bean) and lib (library files) as well as Paintings (where the svg are saved, subfolder preview contains the 200*150 preview pictures). There also is a file called index.jsp.

When a user calls the SVG-Paint root URL, the index.jsp is called. This page simply forwards to /pages/main.jsf. WEB-INF defines all .jsf to be handled by the Faces-servlet. If the bean (Paintings.java) is not loaded yet, it will be loaded by the servlet (as it is defined as "managed bean" in faces-config.xml). Then the facelet view handler (defined in faces-config.xml as well) takes the /pages/main.xhtml (since /pages/main.jsf was called) and evaluates all templating commands. Then the JSF commands are evaluated, and the resulting page is shown on the browser (which would be the main-screen).

Almost all actions (rotate, zoom, ...) are done by javascript and DOM-Manipulation. If the user wants to save his work, the DOM-Tree of the main.svg (which has likely been manipulated) is being sent to the (relative) address "beispiel.xml" with the XMLhttpRequest object. in web.xml the servlet responsible for *.xml is the XMLhttpRequestHandler Servlet [GM]. The servlet reads the information, saves it as .svg (with the name specified by the user) in the folder /Paintings. It also generates a preview picture of size 200*150 in the folder /Paintings/preview. The generating of the preview pictures is done by batik [Teab].

If the user wants to load a previously made painting, he clicks on the load button. This is actually a *commandButton* as defined in the faces-api [micc]. This button calls the *showFiles* action. In the faces-config there is a navigation rule that defines that if the action *showFiles* is called from the page main.xhtml then it should forward to the load.xhtml.

The load.xhtml uses the bean to read out the /Paintings/preview folder and iterates over the preview-pictures using c:forEach from the jstl:core-library [micb]. If the user clicks one of the pictures, then the action loadFile is called, and the file-name is added to the request with a `<f:param>` tag. Faces-config.xml defines that the action loadFile from load.xhtml results in forward to main.xhtml. There, the parameter containing the file-name is read out (with a c:if tag) and the file is used instead of the standard main.svg.

3.3 Creating and modifying a painting

First of all - to avoid problems with scrolling instead of zooming - choose a resolution that still fits into your screen from the resolution-picker. To be able to Draw one needs to select the mode "draw" from the dropdown menu in the upper left. This is due to the fact that it should be possible to create a new circle on top of an old one. If one selects one of the shapes from the other dropdown menu, drawing mode is engaged as well. After creating a new element, it is automatically selected. One can select other objects by clicking on them

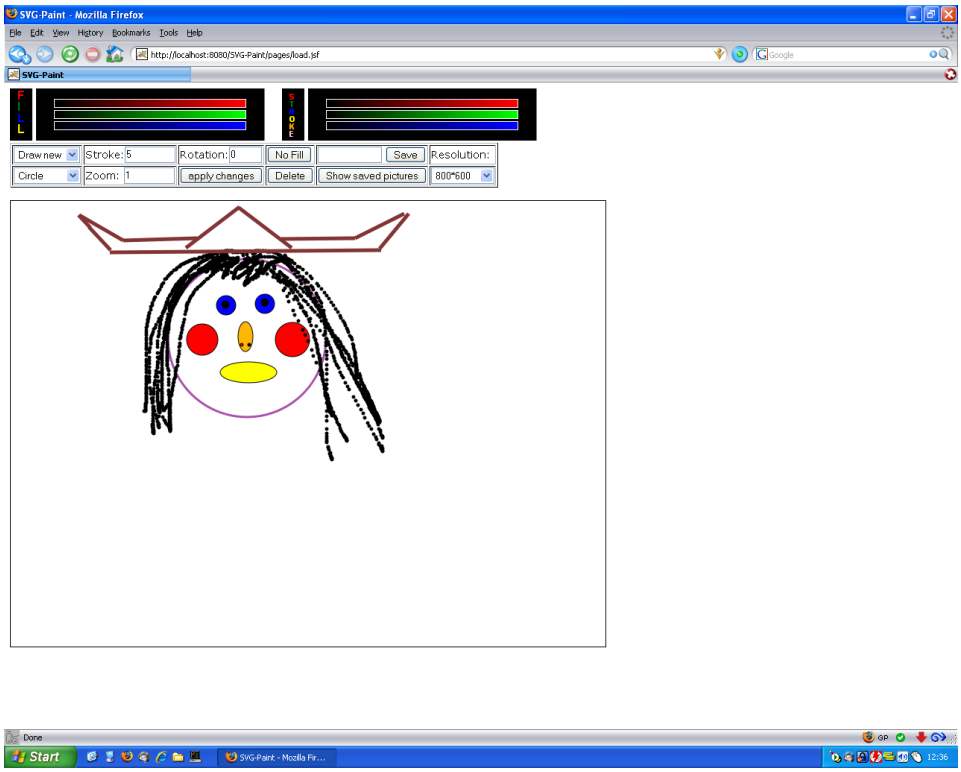

Figure 2: Main Screen of SVG-Paint

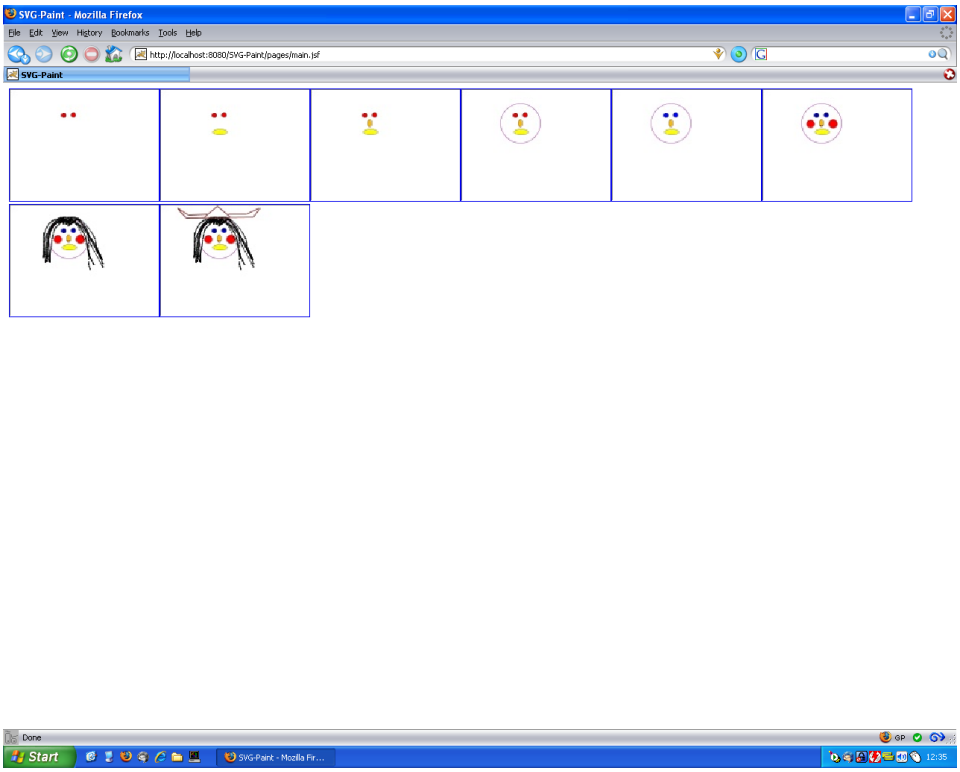

Figure 3: Load Screen of SVG-Paint

in “modify” mode.

A selected Object can be rotated, zoomed - either with the wheel (every notch equals a +5% increase or a -4.76% decrease) or through the input-field. The zoom can be reset to 1 with the “resetZoom” button. It can be dragged to another position, and its stroke-width can be changed with the corresponding input field. In addition one can change the colors of both stroke and filling, or set filling to “none” thus making it see-through. There are 16 million colors possible, a color is defined by setting its RGB values via the color pickers. The current color is shown in the background of the picker. Objects can be deleted as well.

3.4 Saving and Loading

To save, simply type the name your file should have, then hit the “save” button. Only the characters A-Z, a-z, 0-9 and _ are allowed. This sends the current SVG DOM-tree to the XMLhttpRequest servlet, which saves it in the folder “/Paintings” and creates a 200*150 thumbnail in “/Paintings/preview”.

3.5 Possibilities for further development

- The possibility to upload custom SVG (which would have to be parsed, corrected and extended (with the needed variables))
- The possibility to upload ANY picture file (convert raster graphics to vector graphics, for example with Delineate [McK]), then conversion as above
- User System so users can only overwrite / delete their own files
- Version Tracking - could be either diffs or whole file each time - this could go along with an “undo” button.

3.6 Compiling and Installing

To be able to compile, one needs Ant [Teaa], Tomcat [Teac] and Java [mica]. To compile, one only has to edit the build.xml in the ant directory of the source-distribution, and change the deploy.dir property so it leads to the tomcat webapps folder. After this has happened, run ant on this folder. It will automatically build the programm, and create a .war file that is deployed in the deploy.dir.

To install, simply start Tomcat if it is not running already, and the installation is complete.

References

[GM] Ed Burns Greg Murray, Tor Norbye. Using JavaServer Faces Technology with AJAX. https://bpcatalog.dev.java.net/nonav/ajax/jsf-ajax/index.html. [Online; accessed 10-August-2006].

[Hoo] Jacob Hookom. Facelets - JavaServer Faces View Definition Framework. https://facelets.dev.java.net/nonav/docs/dev/docbook.html#gettingstarted-setup-faces. [Online; accessed 10-August-2006].

[McK] Robert McKinnon. Delineate. http://delineate.sourceforge.net/. [Online; accessed 10-August-2006].

[Mei] Dr. Thomas Meinike. SVG Learning By Coding. http://www.datenverdrahten.de/svglbc/. [Online; accessed 10-August-2006].

[mica] Sun microsystems. Java Software Free Download. http://www.java.com/en/. [Online; accessed 10-August-2006].

[micb] Sun microsystems. JavaServer Pages Standard Tag Library 1.1 Tag Reference. http://java.sun.com/products/jsp/jstl/1.1/docs/tlddocs/index.html. [Online; accessed 10-August-2006].

[micc] Sun microsystems. Tag Library Documentation Generator - Generated Documentation. http://java.sun.com/javaee/javaserverfaces/1.2/docs/tlddocs/index.html. [Online; accessed 10-August-2006].

[Mün] Stefan Münz. SELFHTML. http://de.selfhtml.org/index.htm. [Online; accessed 10-August-2006].

[O'R] Tim O'Reilly. What is the Web 2.0? http://www.oreilly.de/artikel/web20.html. [Online; accessed 10-August-2006].

[Ros] Marco Rosenthal. SELFSVG. http://www.selfsvg.info/?toc. [Online; accessed 10-August-2006].

[Smi] Sergey Smirnov. jsf12KickStart. http://www.jsftutorials.net/download/jsf12/jsf12KickStart.zip. [Online; accessed 10-August-2006].

[Teaa] Apache Ant Team. Apache Ant. http://ant.apache.org/. [Online; accessed 10-August-2006].

[Teab] Apache Batik Team. Batik SVG toolkit. http://xmlgraphics.apache.org/batik/. [Online; accessed 10-August-2006].

[Teac] Apache Tomcat Team. Apache Tomcat. http://tomcat.apache.org/. [Online; accessed 10-August-2006].

[W3C] W3C. Scalable Vector Graphics (SVG). http://www.w3.org/Graphics/SVG/. [Online; accessed 10-August-2006].