SBOX FRAMEWORK

Maroš IVANČO, Master Degree Programme (1)
Dept. of Computer Science and Engeneering, FEI, SUT in Bratislava, Slovakia
E-mail: ivancoma@hotmail.com

ABSTRACT

This document discusses automated building of user interface using the XML language. I will describe reasons that served as motivation to start the work on the project. You will also find application of the Model-View-Controller design pattern and so called W3C model in the process of design and implementation of the SBox framework. I will also summarize features of the product at the end of the document.

1 INTRODUCTION

I met with automated building of the user interface first time few years ago. After several nights spent by endless changes in the GUI, I decided to automate my work. I started to generate a menu using a configuration file. The information about names, icons and tool-tips was stored in the Lisp-like list structure. Even though I implemented the component, I did not expand the strategy because of the parsing complexity.

I met with the framework Struts (STRUTS PROJECT) few years later. Struts is the implementation of the Model-View-Controller design pattern – the pattern widely used in the world of J2EE as a pattern for creation of multi-tier interactive applications. During the time, I needed to create GUI within a school project. I realized that it is possible to apply most parts of the Model-View-Controller design pattern in the process of creation local (not web) GUI. Thus, I decided to apply the pattern with application of so-called W3C model as a model for presentation tier. To my surprise it worked. Despite the simplicity, I was able to add new functionality and it all worked together. The problem of parsing complexity solved XML.

I would like to present the result of my work within the project SBox. I will describe in short order design pattern Model-View-Controller (SINGH, 2002) and so-called W3C model. Next, I will describe asset of the patterns application in the process of solving the problem of creation local GUI. Furthermore, I will describe design and implementation of the SBox framework. Finally, I will show some interesting features and I will share usage experience.

2 MODEL-VIEW-CONTROLLER DESIGN PATTERN

Model-View-Controller (SINGH, 2002) design pattern is widely used approach for creation of multi-tier interactive applications. Model-View-Controller (hereinafter MVC) divides an application into three layers – Model, View, and Controller. Each of the layers has

specific tasks and responsibilities. Model binds data model of the application and the business logic. View presents input and output screens. The Controller layer controls the flow of the screen according to the actual request and state of the application. One of the important features of the MVC is low level of coupling between the layers. Because of this feature, an application designed according to the pattern is more flexible, less sensitive to changes with more simple and cheaper maintenance. Low level of coupling also allows developers to work at different layers of the application simultaneously.

The MVC pattern is designed according to client-server architecture and expects an implementation on the top of some request-response protocol. The Controller layer cannot be used directly within the SBox framework, because there is no natural flow of screens in the local GUIs.

Almost all components in the Java programming environment follow the concept of different layers separation using implementation of some "model" interfaces. Thus, almost all components can be used directly within the SBox framework. Despite the fact, that the way of a presentation depends on the data presented, and that there must exist a way for presentation layer to communicate the model, I will not deal with the design of the Model layer. The topic is beyond the scope of this document.

2.1 W3C MODEL

The W3C model (W3C PROJECT) is a reflection of the experience with web technologies and indirectly results from W3C specifications. The model further divides presentation layer, gaining interesting features that are easy to map to the environment of visual component composition libraries in Java.

The essence of the model lies in the consistent separation of the data structure from the presentation structure and the way the data will be displayed (Figure 1). Data

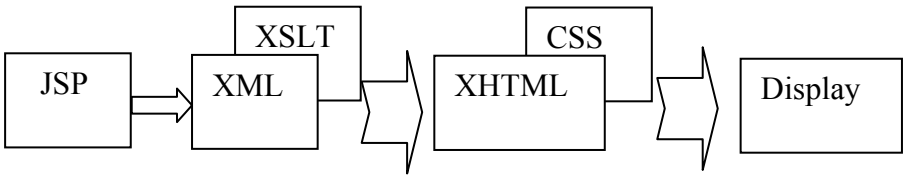

Fig. 1: *W3C model in the context of presentation layer*

structure in the form of XML (what to display) is transmitted to the client, where the data structure is transformed to the presentation structure (where to display) using XSLT. The presentation structure in XHTML is further displayed using CSS style-sheets (how to display). When the dynamic changes are needed, the JavaScript is used. The XML in the W3C model is equivalent to the model components; XHTML is equivalent to the content of the presentation-structure element in a SBox configuration file. Visual beans with resource files serve with functions similar to those in CSS style-sheets. Interpreted JavaScript is replaced by native Java code. The additional flexibility of the presentation layer allows thorough isolation of the changes in the presentation layer. The parts of the layer are more comprendious, reusable, and with cheaper maintenance.

3 SBOX FRAMEWORK

I will describe SBox framework as I have designed, implemented, and used it. I divided chapter into three parts. In the first part, I will describe an encapsulation of a visual component, realization of the division the layers Model and View, and mapping of user inputs into actions. Next, I will describe communication between the layers, and the communication between visual components. In the part Modularity, I will describe the composition of component from its subcomponents. Finally, I will describe standard extensions of the framework and especially custom tag binding mechanism.

3.1 ENCAPSULATION

The base of the framework lies in the two classes `Context` and `RootContext`

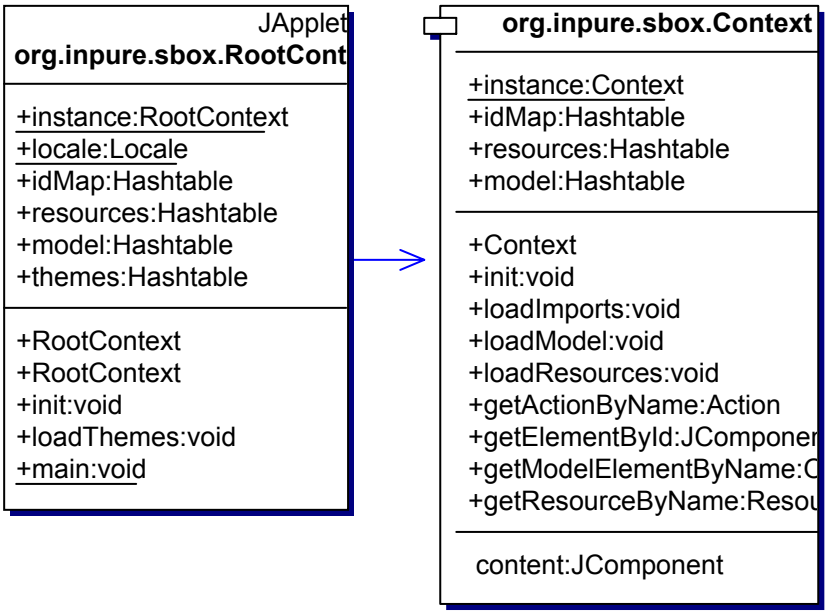

Fig. 2: *Basic classes of the SBox framework*

depicted on the figure 2. The classes represent a context responsible for creation of the visual component. The context also binds model and information about the visual component environment. The context offers to its component information about look-and-feel theme, local settings, subcomponents, components accessible using ID, resources, and about components, which are part of the application model. The context also contains methods for its initialization from configuration files.

Even though the both classes are very similar, there is no way to create the inheritance relation among them. The Java programming language forbids multiple inheritance. Thus, the class `RootContext` serves as a context for top-level visual components (JFrame, or JApplet) and binds application scope properties (theme, locale) and on the other hand, `Context` serves as a context for subcomponents.

During initialization of the context, the information is loaded from the configuration file into several hash tables and the visual component is created. An example of the configuration file for a `RootContext`, with empty presentation structure, is depicted on the figure 3. Following the figure, the model of the application will be filled, according to element `model`,

with two components with names: “machine” and “tableModel”. The element `theme` within the `plaf-mappings` element specifies that the theme with name “idea” will be used. The `input` elements within `input-mappings` element specify mappings between keystrokes

```
<?xml version="1.0" encoding="UTF-8"?>
<rootContext xmlns="http://www.inpure.org/SBox">
  <imports>
    <import name="memoryPane" href="./memory/memoryPane.xml"/>
  </imports>
  <!-- Definition of non visible components that are part of the  application model. -->
  <model>
    <item name="machine" type="org.inpure.emips64.machine.Machine"/>
    <item name="tableModel" type="javax.swing.table.DefaultTableModel"/>
  </model>
  <!-- Specifies theme to use.-->
  <plaf-mappings>
    <theme name="idea" type="org.inpure.sbox.plaf.IdeaTheme"/>
  </plaf-mappings>
  <!-- Defines presentation structure of the application.-->
  <presentation-structure/>
  <!-- Defines the application input mappings between a keystroke and a name of certain action. -->
  <input-mappings>
    <input keystroke="alt X" action="settings"/>
  </input-mappings>
  <!-- Defines the application action mappings between a name of certain action and the actual action
       object of specified type to be used.-->
  <action-mappings>
    <action name="fileMenu" type="org.inpure.sbox.actions.Action"/>
    <action name="runMenu" type="org.inpure.sbox.actions.Action"/>
  <action-mappings>
 <!-- Mapping between a name of the resource and the resource it self -->
  <resource-mappings>
    <resource name="action" href="org.inpure.emips64.resources.Action"/>
  </resource-mappings>
</rootContext>
```

Fig. 3: *An example of the configuration file*

and the action objects accessible by their names. The `action` elements within `action-mappings` element specify mappings between action names and their implementation objects. Finally, the `resource` elements `within resource-mappings` element specify loadable resources, that the application or component will use. Resources typically contain localized information about icons, tool-tips and descriptions of the visual components.

The content of the element `presentation-structure` specifies the presentation structure of the component. The `element presentation-structure` depicted on the figure 3 is for the sake of clarity left empty. An example of the non-empty element is depicted on the figure 4. A context would create, according to such presentation structure element, visual component with one border panel (JPanel with border layout manager). Inside the panel would be a toolbar in its north part. The toolbar contains one button, an empty combobox, and another button. The action with the name “openProject” will fire after the first button invocation and action with name “run” after the second button invocation.

Thus, it is possible to set, using the elements of the configuration files, miscellaneous features of the created component or application. The model layer and presentation layer are

consistently separated and communicate with each other using precisely defined interface of the context.

3.2 MODULARITY

Despite the careful design, even simple application contains too many elements within the presentation-structure element. In order to decrease the level of complexity within the element,

```
<?xml version="1.0" encoding="UTF-8"?>
<context xmlns:="http://www.inpure.org/sbox/sboxComp">
<!-- Definition of non visible components that are part of the  component model. -->
  <presentation-structure>
      <borderPane>
        <north>
          <toolBar>
            <toolBarButton action="openProject"/>
            <item type="javax.swing.JComboBox"/>
            <toolBarButton action="run"/>
          </toolBar>
        </north>
      </borderPane>
  </presentation-structure>
<!-- Defines the component's action mappings between a name of certain action and
the actual action object of specified type to be used.-->
 <action-mappings>
  <action name="first" type="org.inpure.sbox.actions.Action"/>
 </action-mappings>
 <resource-mappings>
  <resource name="action" href="org.inpure.emips64.resources.memory.Action"/>
 </resource-mappings>
</context>
```

Fig. 4: *An example of the simple component configuration file*

I decided to add option to create modules to the SBox framework. An example of the configuration file for simple component is depicted on the figure 4. Yes, the elements of the configuration file are the same as in the chapter 2.1-Encapsulation. You can insert component created according to such configuration file to the other component. First, you must declare element within imports tag using the `import` element (figure 3). Next, you can insert declared component to the certain place using its name as a reference. After the insertion, the inserted component gains access to the models of its ancestors in the component hierarchy. The components at the same level cannot share directly their models. Thus, the modularity, besides the simplification of the configuration files, allows creation of the certain hierarchy in the model of the whole application.

3.3 STANDARD EXTENSIONS

Framework SBox provides standard localization and internationalization features inherited from the Java programming environment.

I have also designed and implemented mechanism of presentation structure custom tag

binding. The user can thus redefine implementation objects of the existing elements or even define new ones. Thus, the meaning of the presentation elements can be modified or extended to suite as many different projects as possible.

4 FEATURES

I created user interfaces applying the SBox framework within two projects. Consistent separation of the model and presentation layer, and particularly flexibility resulting from the use of configuration files allowed me to achieve rapid implementation and robustness. When comparing with statically compiled application, the application runs, after successful "loading" from its configuration file, with none or with minimal slowdown. The set of the XML Schema grams is the integral part of the framework. In combination with suitable text-editing tool this set decreases significantly probability of a bug introduction during the configuration file composition. The framework also allows creation of the subcomponents using the configuration files. You can subsequently use the subcomponents at different parts of your application or even in the different projects. Thus, modularization, besides the hierarchical ordering of the application model, increases reuse of the components. XML representation of the configuration files makes changes of the application look-and-feel simple even for users with minimal (or none) knowledge of the Java programming environment. Thus, the user can edit the application in the way that suits his/her requirements the most. Because XML is the language designed with respect to machine processing, the framework allows automated generation or automated editing of the application using some expert system. Of course, the SBox Framework is published under the terms of GNU GPL.

5 EPILOGUE

The SBox framework already served well in the process of creation GUI in the projects eMips64 and evolutionX. Thus, I can rank it only very well. When comparing with similar projects, the SBox framework especially differs in GNU GPL availability, optimal balance between XML and Java code, and in no need for any scripting language within the framework. In my opinion, the framework is suitable especially for fast GUI prototyping. In the cases where the slightly slower "loading" of the application is not the problem, I would recommend the SBox framework (because of its other features) also for deployment.

ACKNOWLEDGEMENT

I would like to thank to my friend Ondrej Holubek for his useful comments and recommendations.

REFERENCES

[1] ALUR at al. 2001. Sun Java Center J2EE Patterns

[2] SINGH at al. 2002. Designing Enterprise Applications with the J2EETM Platform, Second Edition

[3] STRUTS PROJECT, `http://jakarta.apache.org/struts`

[4] W3C PROJECT, `http://www.w3c.org/`