About an Extension of the Model-View-Controller Design Pattern for Increasing the Flexibility of Web based Applications

Design and First Experiences

Markus Markard and Marc Jansen
Computer Science Institute, University of Applied Sciences Ruhr West, Tannenstr. 43, Bottrop, Germany

Keywords: Web based Applications, Model-View-Controller, Design Pattern.

Abstract: The development of web based applications gained enormous interests in recent years. Most of formerly desktop based applications nowadays provide at least a web based version or are completely re-implemented as web based applications. Nevertheless, from the development point of view, there are still a lot of strategies for the development of web based applications borrowed from the development strategies for desktop applications. Therefore, this paper concentrates on the description of an approach that allows to re-use a from the development of desktop applications well-known Design Pattern with a distinct enhancement for web based applications.

1 INTRODUCTION

In recent years, many applications that were formerly developed as stand-alone applications, moved towards a web based implementation. Still, most of the development approaches like the Design Patterns presented by Gamma, Helm and Johnson (1994) or test strategies used in the area of the development of web applications are borrowed from the development of stand-alone desktop applications. Here, this paper concentrates on the description of an approach based on a well-known Design Pattern, the Model-View-Controller (MVC) pattern (Reenskaug, 1979), for the implementation of desktop applications and shows how this pattern can be extended in order to provide more flexibility with respect to the development of web based applications.

While there are already some modifications to the MVC Design Pattern available, we believe that the described approach still provides some more flexibility.

Therefore, the remainder of this paper is organized as follows: first, an overview about the current state of the art in Design Patterns for flexible user interface development for web based applications is presented. Afterwards, we describe the enhancement provided by our approach, followed by some implementation aspects. After this, first experiences from two industry projects are described, in which our approach was already successfully used. Last but not least, a discussion about the presented approach, along with an outlook to future work, is presented.

2 STATE OF ART

The major Design Pattern nowadays used for user interface development is the MVC pattern. Although there are already some refinements existing for web applications based on the MVC, e.g., the Model-View-Presenter pattern (Potel, 1996), still a lot of web based applications are developed along the MVC pattern.

One of the problems that usually occur while developing web based applications with this pattern is based on the usual partitioning of web applications (Sridaran, Padmavathi, Iyakutti, 2009). In web based applications, the view is usually dislocated from the business logic since the view is visualized on client side (in the web page of the user) and the business logic, which usually is referred to as part of the model, is deployed server side. Therefore, communication between the model and the view becomes difficult. Common solutions to this problem are to keep a representation of both, the view and the model either on server or on client

Markard M. and Jansen M..
About an Extension of the Model-View-Controller Design Pattern for Increasing the Flexibility of Web based Applications - Design and First Experiences.
DOI: 10.5220/0004370303040307
In *Proceedings of the 9th International Conference on Web Information Systems and Technologies* (WEBIST-2013), pages 304-307
ISBN: 978-989-8565-54-9

side. Here, a pattern that provides more flexibility for the description of the view, can be beneficial in order to minimize the requirements for the view and to allow for a more flexible description of the graphical representation of the data provided (and calculated) in the model.

The next section describes the enhancement of the MVC pattern that achieves the mentioned flexibility.

3 ENHANCEMENT OF THE MVC DESIGN PATTERN

The classical MVC pattern allows a distinction between the model and the view of the application. Here, the model represents either the data of the application and the business logic that works on the data of the application in order to calculate the application results. Additionally, the controller is needed in order to foster the communication between the model and view.

While one of the major ideas for the development of the MVC pattern was the idea of providing different (visual) representations for the same set of data, this is still an issue where some improvements can be achieved with respect to web based applications.

The major idea of the enhancement provided by our approach is that we allow to further parameterize the view. Therefore, we provide two possibilities for controlling different parts of the view:

- Parameterization through an XML document
- Parameterization over get parameters passed to the web application

In order to allow this kind of parameterization of the view, we extended the standard MVC pattern by a configurable view, as shown in Figure 1.

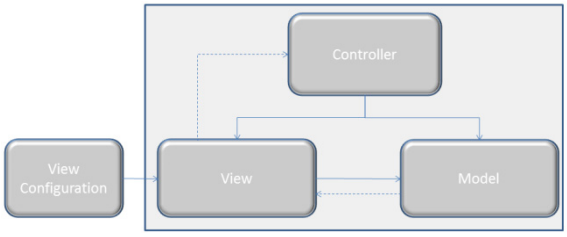

Figure 1: The extended Model-View-Controller pattern with a View Configuration.

At the left part of Figure 1 the usual MVC pattern can be seen. The extension is provided by the external configuration of the View, as explained above either via a XML document or passed as parameters to the web application. The next section describes some implementation details together with an example of a XML configuration file.

4 IMPLEMENTATION

Our approach described in the previous Section was tested in a first prototypical implementation using the Google Web Toolkit. In this Section we present the major parts of the prototype's architecture as shown in Figure 2 and the prototype's potential to parameterize the view.

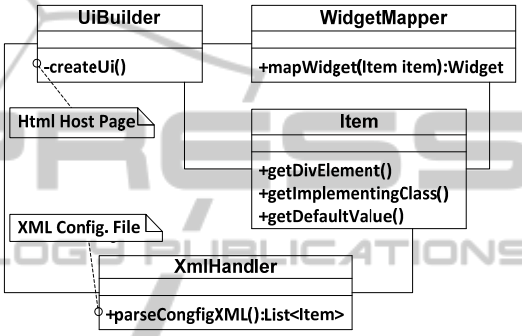

Figure 2: The Prototype's Architecture.

As can be seen in the lower area of Figure 2, the server side located XML handler is responsible for the parsing and deserialization of the XML configuration file. To clarify the XML handler's purpose it is appropriate to previously introduce the structure of the XML configuration file, as shown in the following sample listing. For improved readability of the listing, the attribute's values are only presented as single characters or a single word.

```
<configuration>
<item div="a" class="b" value="c" />
<item div="a" class="ListBox" >
 <entry value="x" />
 <entry value="y" />
 …
</item>
…
</configuration>
```

The root element of the XML configuration file envelops only one type of child element called item. In our listing the first item element is the rule, it has three attributes, namely div, class and value. The div-attribute contains the identifier of a specific div element in the HTML host page. The class attribute holds the full class name of the widget UI component to be positioned in a specific div

element. The last attribute named value represents a default value of the component, e.g., an inscription in case of a button or a default input value in case of a textbox. The second item constitutes a special case by extending the definition of the first item. It also just has one div and one class attribute but can have any number of entry child elements. Since every entry element has a value attribute it is possibly to set multiple default values to list-boxes or other UI components supporting multiple values.

After this brief introduction to the XML configuration file the purpose of the XML handler is summarized quickly. The XML handler parses all item elements in the configuration file and creates an Item object with corresponding initial values for each one. All new created Item objects are directly added to a list. In the middle area of Figure 2 the Item class is shown.

As the name suggests, the widget mapper class, shown in the upper area of Figure 2, delivers a new widget instance for each item in the item list. At runtime, the instance of the widget mapper, instantiates new UI components according to the item's class attribute.

Populating the host page with UI components is the task of the UiBuilder class utilizing the WidgetMapper and the item list constructed by the XMLHandler. The UiBuilder, shown in the upper area of Figure 2, iterates through the list of items and the WidgetMapper returns a UI component corresponding to the certain class attribute of each item in the list. After identifying the according div element in the host page the returned UI component is added to the particular div element. Of course the UI Builder does not only append UI components to the host page but also handles the adding of listeners and DOM identifiers to these components. Adding DOM identifiers to UI components placed on the host page gives us the opportunity to set individual styles for every single component. The listing below shows an example for an individual CSS that is applied to an UI component with the DOM identifier button_id.

```
#button_id {
 height:20px;
 padding-top:0px;
 padding-bottom:3px;
 font-size:10pt;
}
```

Positioning and appearance of the UI components can be defined over the positions of the according div elements in the HTML host page. Their visual representation can be controlled by using Cascading Style Sheets. Every widget used in the UI has its own div element, identified by the id of the item's div attribute. The div elements containing the widgets can be grouped and arranged by using HTML and CSS, so it is possible to rearrange the UI without altering the actual Java source responsible for the widgets and their functions.

Controlling the selection of classes defined in the configuration file is an even wider reaching tool to influence the UI. By combining CSS manipulation and the control over selected classes it is even possible to have completely different UIs, e.g. a desktop UI and a mobile UI without the need to alter the actual sourcecode.

So the UI created in our Prototype can be modified in three different ways without touching the sourcecode of the actual application. The first and simplest way is to modify the default attributes of the item elements in the configuration file. Doing so enables quick changes to label texts or default values of input widgets.

The second way allows the modification of the styles associated to the div elements containing the widgets. These styles should be used to define the position of an UI component, so the possibility to modify these styles allows us to gain control over the whole layout without using a source based layout offered by frameworks like the Google Web Toolkit that we used for our prototype. At this point we also have to mention a problem that has been occurred during implementation of the prototype. While making extensive use of CSS manipulations cross browser CSS issues cannot be avoided.

Last but not least, the individual look of every widget can be influenced by the modification of the styles associated to the DOM identifier of a particular widget.

5 FIRST EXPERIENCES

The developed approach was already used in two independent projects with two independent companies.

The first company is located in the business of hotel room management and provides a framework that allows to easily develop booking engines for hotel rooms based on a rich database of hotels available all over Germany. Whereas the second company is working in the area of management consulting, especially for IT companies.

5.1 Advantages of the Presented Approach for a Hotel Room Management Software

Since the company in the area of the hotel management software provides a framework that has to be very flexible with respect of the graphical representation of the different components, in order for being able to easily integrate the framework into an already existing webpage of a customer, the presented approach was suited for this integration.

The presented approach allowed to easily integrate standard widgets, already used by the rest of the webpage, into the framework, so that the look and feel of the booking engine, in comparison to the rest of the webpage, could be integrated smoothly.

Furthermore, our approach allowed to highly customize the framework without even touching the sourcecode of the framework. Just as one example, we will present one of the customizations that was achieved without touching the sourcecode of the framework itself. Imagine you are the organizer of a local event in your home town. If you want to integrate a hotel room booking functionality within the homepage promoting your event, some of the fields usually available for a hotel room booking engine are obsolete, e.g., the dates of arrival and departure, the zip code of the town where the visitor of your webpage is looking for a hotel, … just because all these parameters are already predefined by the event that you organize.

By using our approach, we were able to customize these type of fields either with the help of the XML configuration file and/or by defining the appropriate values as parameters to the webpages.

Therefore, the presented approach was perfectly suited for this kind of integration of an external framework into an already existing webpages.

5.2 Advantages of the Presented Approach for an IT Management Consulting Company

The second example where we used the presented approach was together with a management consulting company that usually works for large IT companies. Parts of their customers are located in the German public sector. Therefore, our partner also supports their customers in their complex bidding procedures, that are very much regulated by complex German and European laws.

Here, we developed a Web 2.0 based simulation for planning the bidding procedures. One of the requests from our partner was that we should provide two different versions of the simulation, one for internal use (available only via the intranet of the company) and one as sort of an advertisement for their publicly available webpage. The major difference between these two solutions is the degree and amount of parameters that can be set by the user of the simulation.

Again, the presented approach allowed to easily customize the version that was primarily developed for internal use (with a very rich set of different parameters available to the user of the simulation) in order to restrict the functionality of the publicly available simulation by strapping down and pre-configuring certain parameters of the simulation.

6 DISCUSSION AND OUTLOOK

As the examples discussed in the previous section show, the presented approach provides advantages for modern web based applications on different levels. Of course, since there is nothing like a free lunch, these advantages of flexibility come with some drawback, especially with respect to the increasing complexity that the presented pattern adds to the usual Model-View-Controller pattern.

Nevertheless, since the additional complexity is on the one hand not too huge (in comparison to the gained advantages) and can on the other hand easily be tackled by an experienced programmer, we guess that programmers will usually be willing to use the presented approach.

Furthermore, future research will be necessary in order to decrease the added complexity, so that the usage of the presented extension of the Model-View-Controller design pattern will still get easier and therefore will most likely attract more developers.

REFERENCES

Gamma, E., Helm, R., Johnson, R. E. (1994). *Elements of Reusable Object-Oriented Software* (1st ed.). Amsterdam: Addison-Wesley

Reenskaug, T. (1979). *MVC: XEROX PARC 1978-79.* Xerox

Potel, M. (1996). *MVP: Model-View-Presenter: The Taligent Programming Model for C++ and Java.* Taligent

Sridaran, R., Padmavathi, G., Iyakutti, K. (2009). A Survey of Design Pattern Based Web Applications. *Journal of Object Technology*, Vol. 8, No. 2, 2009