A Model-Transformers Architecture for Web Applications

Alexey Valikov, Alexei Akhounov and Andreas Schmidt

Forschungszentrum Informatik
Haid-und-Neu Str. 10-14
76131 Karlsruhe, Germany
{valikov, akhounov, aschmidt}@fzi.de

Abstract. This paper proposes a web application-oriented modification of Model-View-Controller architecture, which allows generic implementations of the controller and the view. Proposed architecture is based on the conjunction of two techniques: remote procedure calls for HTML/HTTP based web applications in the controller part and automatic presentation of the model state information with several XSLT transformations on the view part.

1 Introduction

Accessibility through the web has become an important feature of modern information systems. This surely has a lot of advantages. For instance, global system accessibility and low-cost software support due to the absence of local installations are made possible.

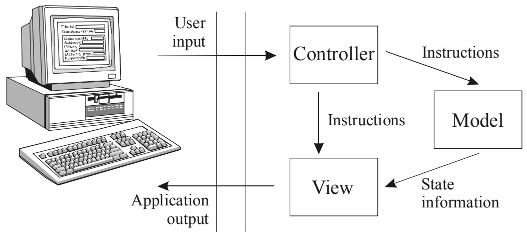


Fig. 1. General architecture of MVC paradigm

Nonetheless, there are certain serious complexities in development of web interfaces. User interface development is a very complex issue even in traditional systems, especially when taking into account that modifications to the UI are quite frequent because of changing user wishes. For more complex applications, this leads to problems of maintenance. In traditional "fat-client" environments, the idea of the model-view-controller architecture [4] has separated different tasks: the state of the application (model), the presentation thereof (view) and the response to user actions (controller) that may result in modifications of the state and the presentation (figure 1).

In web applications, the main difference to traditional UI development is the thin-client approach with a constrained interaction channel between the presentation and user actions on the one side and the application logic and response to user actions on the server side. The controller mainly receives HTTP requests with sets of name-value pairs while the view has to generate "an interpretable description" of the UI in form of HTML forms.

Obviously, procedures of handling parameters and forming an HTML page based on model state information depend on the application. It means that both development of a new application and changes to an existing application will require customization or complete rewrite of the controller and the view.

In this paper we propose an approach, which allows making controller and view objects generic and reusable for different web applications without any adaptation. This approach is based on the conjunction of two techniques: remote procedure calls for web applications and automatic model transformation.

2 Related Work

MVC with its origins in Smalltalk-80 [4] is a well-known approach and was used in numerous technologies. Nowadays, MVC is also employed in web application area, where separation of presentation and application logic is often crucial [1]. This is somehow done in the J2EE platform with Enterprise JavaBeans (EJB). The approach suggested in this paper follows more or less the same design principles. But as long as we are solving a much more specific problem, we may take an advantage of the preconditions we have in web applications and automate certain parts of processing (namely, method calls and presentations).

The approach proposed in this paper is partially based on possibility to invoke methods of model from the client. Despite there are well-developed technologies (RMI, CORBA) and protocols (SOAP) for remote invocation and interaction, they do not solve the problem because of the limitations of HTML/HTTP-based client-server communication. Extension of software or usage of scripting languages is required on the client side; this requirement often simply cannot be met. Model-Transformers architecture does not require usage of advanced features on the client side.

There also are several approaches for MVC-based web applications coming from the open-source community (Apache Turbine, Velocity and Struts). The novelty of Model-Transformers in comparison to these technologies is automatic mapping of the request onto the model's method calls and automatic model presentation.

3 The Controller: RPC for Web Applications

In a web application scenario, user input is mainly a set of parameters submitted via an HTTP request. Controller-model and controller-view communication is usually implemented in conventional systems via simple method calls: in order to change view or model, controller invokes the method of the appropriate object. Consequently, controller fulfils the task of mapping submitted parameters onto method calls (figure 2).

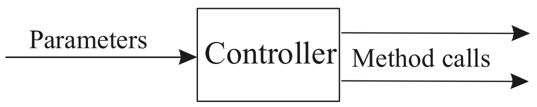


Fig. 2. Controller in web MVC applications

In first approximation, a controller in MVC-based web application actually behaves like a dispatcher for remote method invocations. Of course, it may handle many other useful tasks, but the "dispatcher" functionality is, clearly, the primary task, on which we will be focusing.

By changing control values, the user invokes methods of the model through the controller. The carrier between the user and the controller in this case is a parameter-value set submitted with the request.

This limitation makes it almost impossible to apply current RMI approaches. For instance, SOAP does require SOAP-aware browsers or JavaScript solutions on the client side, which especially poses a problem for very thin devices like handhelds etc. Client-side CORBA or RMI also require more than pure-HTML/HTTP interaction.

Our previous work on this topic [6] proposed a light-weight solution for this problem. The idea is to follow standard naming conventions for parameters, submitted with the request. Standard naming of the parameters allows mapping them correctly onto model method calls. In the simplest case, one parameter invokes one method; for instance parameter value-pair *setDate=12.01.1992* invokes method *setDate("12.01.1992")*. In more complex cases, several parameters may be used to invoke a single method. For example, parameters *setDate(birth).0=1998*, *setDate(birth).1=01* and *setDate(birth).2=12* are mapped altogether onto one call of the method *setDate("1998", "01", "12", "birth")*.

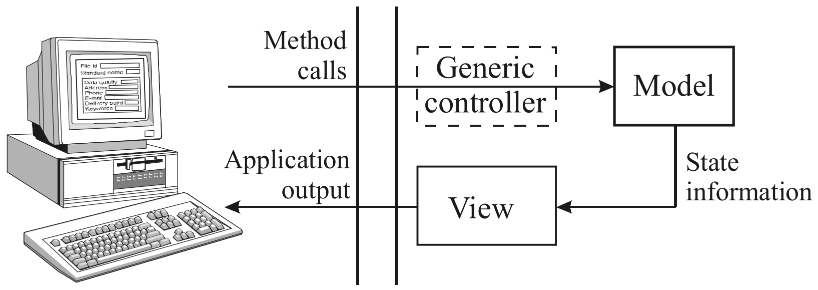


Fig. 3. MVC for web with a generic controller

The controller, which is built according to this specification, provides model-independent functionality of mapping parameters received from the client in an HTTP request onto the model's method calls. In other words, the development of the controller "vanishes" from the web application development. A modified architecture scheme is shown on figure 3.

4 The View: Automatic Model Transformation

The model in the MVC paradigm represents structured information about the state of the real world. This state information is processed by the view and the output is sent to the user. In the context of web applications, the output is an HTML document. In other words, the view is actually a processor, which converts structured model state information into an HTML document (figure 4).

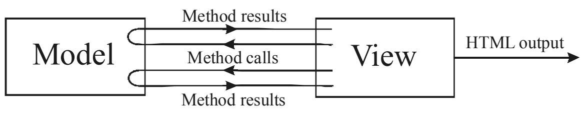


Fig. 4. View in web MVC applications

Similar functionality is implemented in a declarative programming language called XSLT [7], (eXtensible Stylesheet Language for Transformations). Main purpose of XSLT is to transform structure of documents in general and XML documents in particular. If state information of the model could be viewed as a document, then XSLT may be employed to transform it into the required output format, like, for instance, an HTML page. The only question is how to present a model as a transformable document.

The basic idea is that the model exposes its state via readable properties. One of the readable properties is distinguished as a document property. Like in component architectures (e.g. JavaBeans), these properties are represented by methods which (a) are externally accessible, (b) have no input arguments and (c) return booleans, numbers, strings or sets of nodes (types, which can be manipulated in an XSLT transformation). These properties are the input for the transformation process: the document property is the input document for the transformation, and the other properties are mapped onto optional parameters. Via naming conventions, this mapping step can be completely automated.

The remaining problem is how to serialize the type system of the programming language to the data model of XSLT. For primitive types, this is straightforward. For more complex types, we can make use of techniques developed for XML serialization, e.g. SOAP bindings.

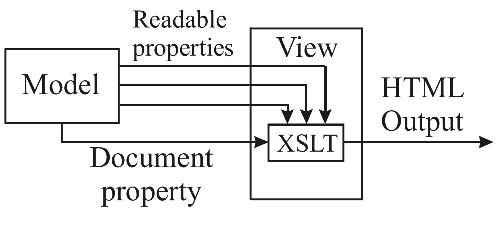


Fig. 5. Automatic model transformation

The task of mapping model's readable properties onto transformation's parameters and input document can be easily automated. Values of readable properties are assigned to parameters with appropriate names; then, transformation is applied to the document returned by document readable property (figure 5).

Obviously, the model-view interaction described above allows the view to be generic. This generic view does not depend on the model: it simply copies assigned readable property values to transformation parameters. For a specific application, only a transformation expressed in XSLT is required for adaptation

The approach may be slightly modified to allow multiple model presentations. This may be achieved with the view consisting of several transformations and the model having several document properties, one document property and one transformation for each presentation. A special *view property* of the model specifies active presentation.

To give an example of automatic view implementation, we will consider the Java programming language. Let the model be implemented as a single class. Then, properties are associated with public methods of the model with names starting with "*get*" and no input arguments. Names of document properties end with "*Document*". View property is associated with the *getView* method.

Mapping of properties onto transformation parameters is straightforward. For instance, *DocumentURL*, associated with *getDocumentURL* model method, defines the value of the *DocumentURL* parameter. View property defines active presentation. For example, "*Edit*" value of the view property means that user will receive the result of *getEditDocument* method previously processed by the *Edit.xsl* transformation.

Figure 6 provides a summary of the process of generating views for a given model.

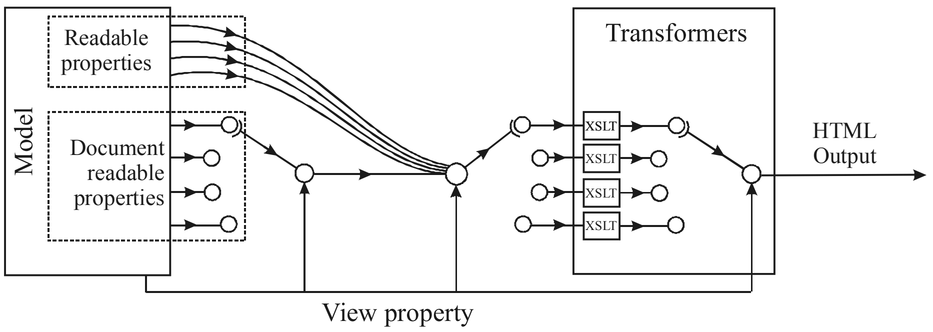


Fig. 6. Automatic model transformation with several transformers

5 Model-Transformers instead of Model-View-Controller

With the automatic model presentation described above, the view implementation phase also disappears from the development of the web application; it is replaced with the design of one or more XSLT transformations (figure 7).

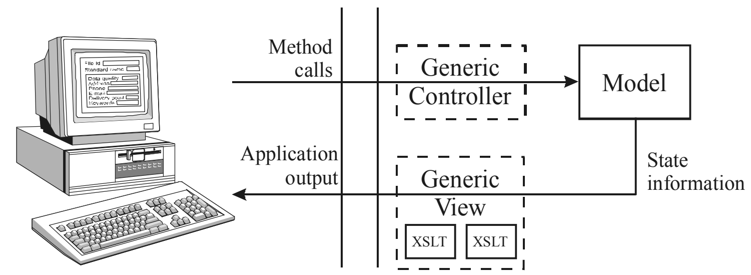


Fig. 7. MVC for web with generic controller and generic view

The solutions presented in the previous sections lead to a simplified design pattern, which consists of a **model** that represents real world and a set of **transformers**, which present the model to the user. We name this pattern *Model-Transformers* architecture; its overall scheme is presented on figure 8.

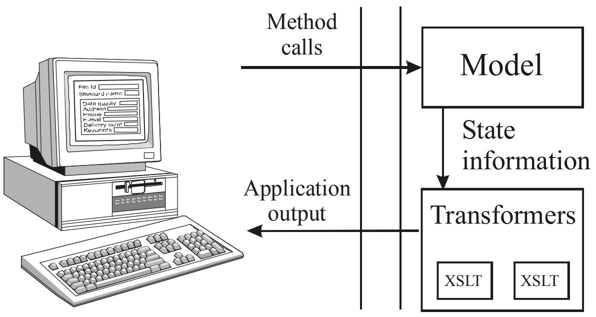


Fig. 8. The Model-Transformers architecture

The Model-Transformers architecture allows a clear separation of two typical application evolution tasks: adding/changing functionality and adding/changing presentations.

The functionality of the application is, basically, what a user may do with the application. In our case, functionality is clearly represented by accessible methods of the model. Hence, to extend functionality of the application, the developer extends the model with one or more methods, which become immediately available to the client. As functional modifications do not change declarations of readable property methods, they do not affect existing views.

New presentations are added by creating new model transformations. If needs of new presentation are not covered by existing model properties, more property methods have to be implemented; otherwise, no model extensions are required. In any case, modification of model presentation does not influence the application logic. Model's readable properties define the interface for accessing model state information. Thus, if application logic is changed but readable properties are not, transformations are not affected.

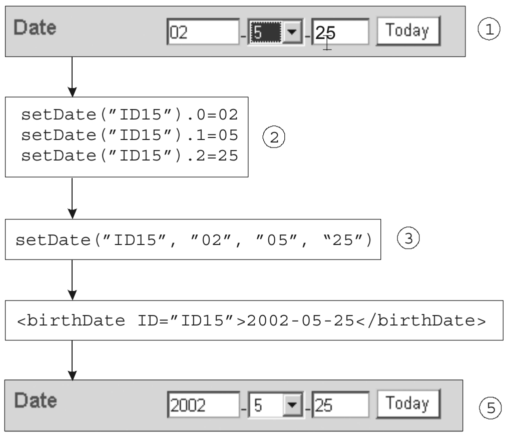
6 Evaluation

The proposed pattern was developed and implemented as an architectural part of XML-based metadata repository designed in FZI for the NOKIS project [3]. NOKIS is a distributed environmental metadata information system for the coasts of the North and the Baltic Sea. Based on previous experiences in CoastBase [2], we have developed a web-based metadata editing and management tool.

One of the key problems was the size of the metadata schema (based on ISO 19115) and the requirement of easy extension and adaptation for institution-specific metadata. We followed a schema-driven user interface generation approach.

Center component in NOKIS project is a web interface for managing metadata in form of XML documents. System requirements such as user-friendly interface for editing complex documents, type-dependent input fields, document validation, possibility to integrate additional applications and context-sensitive help system lead to complex HTML pages overloaded with input controls. Having this multiplied by the size of NOKIS XML Schema and taking into account its possible evolution, results a web application, which is difficult to implement and even more difficult to support.

Design and application of Model-Transformers architecture helped to solve two big issues: request parameters processing and presentation. Automatic handling of parameters and output transformations saved a lot of effort; it even allowed automatic generation of model transformations (including parts of user interface) based on document XML Schema. Support and extensions were drastically eased as well.



```
setDate(”ID15”).0=02
setDate(”ID15”).1=05
setDate(”ID15”).2=25
```

```
setDate(”ID15”, ”02”, ”05”, “25”)
```

```
<birthDate ID=”ID15”>2002-05-25</birthDate>
```

Fig. 9. Example of assigning date value to the element

One simple example, which may illustrate practical usage of Model-Transformers architecture, is assigning element values in document editor. NOKIS model was provided with several methods like *setString(...)*, *setDateTime(...)*, *setFloat(...)* etc., to validate user input and assign values to appropriate elements. Names of input controls displayed in the user interface encode method names. Typical process of assigning an element value is presented on figure 9.

User input (1) is submitted as a set of parameter-value pairs (2), which are automatically mapped onto method calls (3) to validate user input and assign values to the elements of the document (4), which are then presented in the user interface through an XSLT (5). In this example, *setDate(year, month, day)* method implements construction and validation of the date based on three parameters submitted by the client. The method checks and corrects the parameters, if required; for instance, two-digit year parameter was change by this method to four digits.

In addition, we have also found out that with Model-Transformers no custom mechanism is required to integrate external web applications. The generic controller makes no difference between a request from a user and an external application, if they both follow standard parameter naming conventions

Actual implementation of the proposed technique is done in Java programming language. The developed generic controller and view use Java Reflection API [5] to dynamically invoke methods of the model object. Proper caching of reflected method objects makes use of reflection quite efficient: our empirical measurements revealed insignificant invocation deceleration. Alternatively, runtime reflection usage may be avoided by static generation of code method calls.

7 Conclusions and outlook

With the Model-Transformer architecture, we have provided a framework for easy development of complex web applications. There is no need to develop application-specific controller and view objects. The developer simply writes model methods and provides transformations. Moreover, as a method implemented in the model is already defined and (hopefully) documented, tasks of model extension and transformations design may be separated.

However, the Model-Transformer architecture has certain prerequisites. One big issue is the correspondence between parameters submitted by the user and the methods, which have to be invoked in response. A generic controller requires this correspondence to be generic as well. More precisely, it must be possible to encode method name and arguments in parameter name and value. Practice shows this to be true for most of the cases, but this is surely not a panacea.

Another problem is runtime method invocation. Modern programming languages like Java or C# provide runtime reflection capabilities while for older languages there has to be additional effort (for instance, pre-processing or source code).

Model-centricity of the MT approach also increases the model size. For instance, for automated presentation, complexly structured model state information must be transformed into document fragments.

Finally, a solution provided in MT architecture for HTML/HTTP based web applications is simple due to two prerequisites: primitiveness of the user input format and possibility to present the model as a document. These prerequisites are unlikely to be met in subject areas other than web applications. Moreover, development of the Web Services platform (driven by W3C Web Services Activity) may help to overcome complexities connected to the simplicity of HTML/HTTP based client-server communication in web applications. Technologies like WDSL, SOAP, XMLP may allow much more complex interactions over the web. However, general usage of these technologies will require general availability and wide distribution of client software that supports them. Until this is true, solutions like MT for more primitive web environments will continue to be useful.

General conclusion is that Model-Transformers architecture, when applied to the development of web applications gives considerable advantages of easy development, support and integration, while limitations in this context are insignificant. MT-based web applications are also open for integration with other systems from the very start. The model is manipulated through parameter-value sets, which follow defined naming conventions. Hence, it is easier to expose the application as a Web Service.

Although MT in the presented state is a complete solution pattern, several potential architecture extensions may be mentioned. One of the points that require special attention is XML serialization of complex model properties. Another possible improvement is the possibility of decomposing the model into several independent classes.

References

1. Althammer, E. and Pree, W. Design and Implementation of a MVC-Based Architecture for E-Commerce Applications. See http://citeseer.nj.nec.com/443079.html.
2. Kazakos, W., Kramer, R. and Schmidt, A. Coastbase - The Virtual European Coastal and Marine Data Warehouse. Computer Science for Environmental Protection '00. Environmental Information for Planning, Politics and the Public, volume II, pages 646-654.
3. Kazakos, W., Valikov, A., Schmidt, A. and Lehfeldt, R. Automation of Metadata Repository Development with XML Schema. Accepted for the EnviroInfo Vienna 2002 conference.
4. Krasner, G. E. and Pope, S. T. A Cookbook for Using the Model-View-Controller User Interface Paradigm in Smalltalk-80. In Journal of Object-Oriented Programming, Volume 1 Number 3, Aug/Sep 1988.
5. McCluskey, G. Using Java Reflection, 1998. See: http://developer.java.sun.com/developer /technicalArticles/ALT/Reflection/.
6. Valikov, A., Akhounov, A. and Kazakos, W. Remote method invocation for web applications. Submitted to CSIT 2002 conference.
7. World Wide Web Consortium. XSL Transformations (XSLT). W3C Recommendation, Nov. 16, 1999. See http://www.w3.org/TR/xslt.