

## MVC-Webflow: an AJAX Tool for Online Modeling of MVC-2 Web Applications

Marco Brambilla, Alessandro Origgi

Politecnico di Milano, Dip. Elettronica e Informazione, P.za L. Da Vinci 32, 20133 Milano, Italy  
marco.brambilla@polimi.it, alessandro.origgi@mail.polimi.it

### Abstract

*Modern Web applications are characterized by a high level of complexity and deal with huge amounts of data. When the application grows in complexity, manual code development is not suitable, because it lacks in efficiency, reuse, reliability, maintainability, and group work facilities. On the other hand, several Web engineering approaches are too far away from the average developer and designer way of working to be widely adopted. In this demonstration paper we propose a light-weight design notation (with its companion editing tool) that leads to the development of MVC applications. We present an on-line visual editing tool called MVC-Webflow for the specification of simple conceptual models for MVC applications and we provide partial automatic code generation, that can be performed on the flight directly on the deployed application. The advantages of the approach are the closeness to the well known MVC paradigm, the foundation on solid web engineering models.*

### 1. Introduction

Web Engineering has proven a valid approach to the design and implementation of Web applications. A lot of approaches and notations exist to increase efficiency, reuse, reliability, and maintainability. Unfortunately, these approaches are not as widely adopted as one could expect. One of the reasons is that they are often too far away from the average developer and designer way of working. This becomes more and more evident in application development where time to market and continuous update are crucial. Indeed, in these scenarios CMS systems and pre-designed solutions are often preferred. Their main advantage is that most of the updates can be applied directly at runtime, simply by updating the data or the metadata that describe the application content and structure. Unfortunately, this cuts off any design abstractions in the development process, with several disadvantages in

terms of overall design quality and cleanness of the application structure.

Some intermediate solutions have been proposed, that try to provide some design help while developing the application (e.g., see App2You, where data model and business logic are built at runtime while the designer adds contents and pages to the application [<http://app2you.com>]), but they still lack a modeling phase of the application.

With this project we want to offer a simple conceptual model which allows developers to manage the application structure exploiting the MVC architecture. To ease the task, we offer an online tool called MVC-Webflow for drawing the data and application structure and turning the specification to a running application on the fly. Using the tool the developer is able to define the flow of the application, the view composed by the JSP pages and the controller which process the user requests and acts as intermediary between the business logic and the view. The tool is implemented on the AJAX platform, for granting maximum usability and interface quality.

The paper is organized as follows: Section 2 clarifies the purpose of the work; Section 3 describes notation and structure of the modeling language that we adopt in the tool; Section 4 describes the MVC-Webflow editing tool and a small application example; Section 5 discusses the related works and concludes.

### 2. Aims and benefits

The main purpose of the MVC-Webflow tool is to lower the barrier to the adoption of Web Engineering techniques in the development practices. This aim is addressed by the following basic choices:

- We allow to model applications with a MVC approach adapted for the Web (namely, the Model 2 design framework), which is widely adopted by developers and has some valid implementation counterparts (e.g., Jakarta Struts);

- We offer an online design tool that provides both solid model-driven design features typical of Web Engineering and quick application evolution thanks to a Web based interface that does not require a rigid development process;
- We provide basic *on the fly* code generation on Model 2 reference architectures (at the moment, we provide code generation for Struts), with special attention to the specification of the Controller.

The advantages of MVC architectures are well known. They allow the developer to design the Web application defining three separate levels: Model, View and Controller. In complex applications, separation of concerns is fundamental; MVC allows it by separating data and business object (the Model), user interface aspects (the View), and the rules that control the execution logics of the application (the Controller). MVC decouples business logics and data access from the presentation thanks to the Controller intermediate component. For the Web, a special version of MVC called Model 2 (also called MVC-2) has been studied.

In our proposal, we offer automatic code generation on the Jakarta Struts platform, a well known implementation of the Model 2 paradigm. From an implementation point of view, the configuration of the Jakarta Struts controller is specified in the file *struts-config.xml*, that specifies the logics for invoking the *action classes*, the model objects playing as interfaces toward the data sources and the business logic.

We provide a visual model for easily describing the controller behaviour and the used action classes. The visual model can be specified through an online tool implemented on AJAX technologies, that allows to design the diagrams, to store them on a server, to retrieve them and to generate the basic pieces of the Struts components. We automatically generate the contents of the *struts-config.xml* file and the stubs of the action classes from the diagram created using the model. The advantages of the approach are:

1. simple and quick design of the application model;
2. good overall vision of the application, with a notation directly referring to the MVC models;
3. independent design of the controller flow and of the called action;
4. easy reuse of parts of the application defined in the diagram.


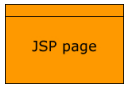

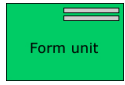




### 3. MVC-Webflow Definition

The MVC-Webflow metamodel is defined by a set of primitives and by the rules for specifying the

connections among those primitives. The diagram resulting from the design phase represents the conceptual schema of the MVC application, that mainly consists of JSP pages (view components), actions (model components), and connections (controller definitions). The primitives of the model are represented in Table 1, together with their main properties, and are briefly described in the following:

- *Action units* specify the way in which the application reacts to the user input: the correct Action unit is called based on the controller decision, and then it processes the user requests, by retrieving data and/or calling the business logic, and provides the information for building the response to the user. The Action unit behaviour must be defined completely by the developer. An Action unit can be linked to a JSP page that shows its results or can be connected to another Action unit for performing further processing (possibly based on the results of the first unit).

**Table 1.** Primitives of the MVC-Webflow models

Symbol	Concept	Properties
	Action Unit	Name (the name of the Form Bean defined in the Formset); Path (to call a specific Action Mapping); Type (path of the Action Class); Scope (persistence of the Form Bean); Validate (to use the validation); Execute method (the logic between the Controller and Business Services).
	Jsp Page	Name; FormBean usage (to generate the corresponding Form attached to the Action unit)
	Jsp Fragment	Name.
	Form	FormBean Name; FormBean Type (the qualified name of the Form Bean Class); Form Properties (array of all the properties of the Form Bean); Plug-in Class name; ValidationXML (validation rules).
	Connection Link	Name.
	Success Link	Name.
	Failure Link	Name.
	Containment Link	Name.

- *JSP pages* represent the View of the model. Each JSP page corresponds to a page in the final Web application and allows to present contents to the user. The JSP page must be defined by the developer

using any sort of editing tool. JSP pages interact with Action units through Links. The JSP page receives the content to be shown in the page. The JSP page can call Action units and send it some parameters.

- *Form units* must be connected to an Action unit and represent the user input from one or more form fields. In this way, Action units can receive the user input coming from the pages of the view.
- *JSP fragments* are portions of reusable code that can be embedded in JSP pages. The main advantage of this primitive is to provide modularization and reusability to the designer. Notice that JSP fragments cannot have input links.
- *Connection Links* allow the developer to define the control flow of the application. A control flow is a directed arc represented with an arrow from the source to the destination. It can connect: (i) a JSP page to an Action unit or to another JSP page; (ii) an Action unit to a JSP page or to another Action unit. A link can carry content objects and parameters. Two variants of the connection links are used for specifying the outcome of a control unit: *Success Links* are shown with a green arrow; *Failure Links* are shown as red arrows. After its execution, the *Action unit* will follow the success path or the failure path depending on a Boolean result. A more refined design approach could consider more complex exit conditions, and add several corresponding outgoing links.
- *Containment Links* are used to include Form units into Action units and JSP fragments into JSP pages. This type of link establishes a relation of containment between the two units. For example a *JSP page* can contain more than one *JSP fragment unit* and this one can be used by more *Action units*.

### 3.1 MVC-Webflow Metamodel

In this section we briefly outline the MVC-Webflow Metamodel. The class diagram in Figure 1 represents the object oriented description of the metamodel. The main model element is *Project*, which represents the whole Web application. The project is composed by the *Data model* and the *Web model*. For lack of space, we concentrate only on the Web model, since the Data model can be specified using a standard representation (E-R, class diagram, ontology model, and so on). A Web model is composed of Web elements, which can be links, JSP pages and fragments, action units and form units. The properties that can be specified for each element are summarized in Table 1. From an MVC standpoint, we can classify

Form units, JSP pages, and JSP fragments as view elements; Action units as model elements; and links as elements that define the controller behaviour.

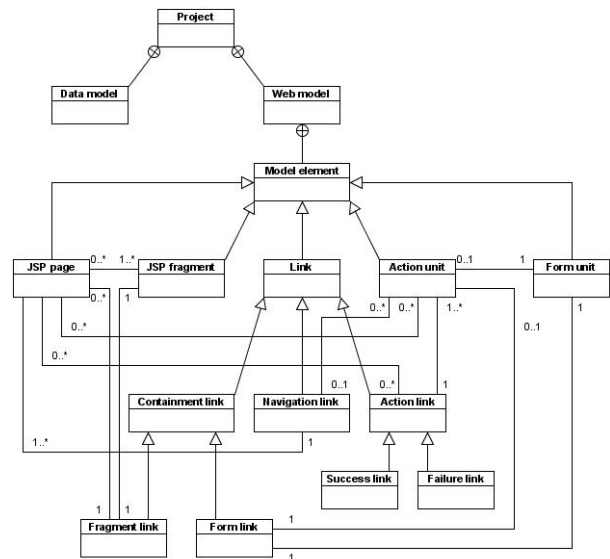
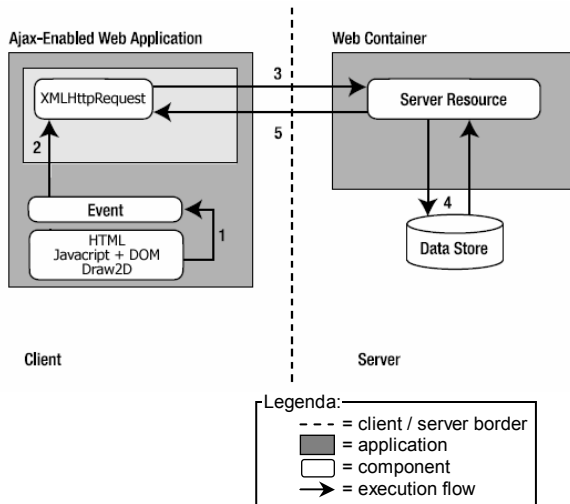


Figure 1. MVC-Webflow Metamodel

## 4. The MVC-WebFlow Online Editor

To ease the specification of MVC-WebFlow models, an online editing tool has been developed. A prototype version of the tool is available online at [7]. The architecture of the editor is based on RIA technologies. A Rich Internet Application (RIA) is an Internet application that attempts to bridge the usability gap between desktop applications and traditional Web sites. At this purpose, it exploits the graphical and execution capabilities of the clients (namely, the browsers), by adding scripts into the pages, thus providing higher quality of interactivity and an user experience. RIAs can be enabled through several technologies, such as Java, Javascript, ActiveX, Adobe Flash, Flex, and others.

The most known platform for RIAs is the AJAX framework, that we adopted for our implementation. In particular, we based the GUI of the tool on the opensource Openjacob Draw2D library and other AJAX technologies: HTML, Javascript, XML objects, W3C DOM, Yahoo! User Interface library (UI), XMLHttpRequest method, Server-side scripting (Java server pages), and MySQL database.



**Figure 2.** Architecture of MVC-Webflow Editor

Figure 2 shows the general architecture of the MVC-Webflow editor. At client side, HTML provides the frame of the editor Web page. The graphic interface is entirely implemented at client side with Javascript.

The modelling canvas is implemented by the Openjacob Draw2D library, which in turns exploits JavaScript and DOM. To support the interaction with the user and to build the editor panels and windows we have used the Yahoo! User Interface library, a set of utilities and controls written in JavaScript for building richly interactive Web applications using techniques such as DOM scripting, DHTML and AJAX. User interactions at the interface level trigger events (1) that in turn activate the interaction with the backend components (2) at the server side. The communication is implemented through AJAX XMLHttpRequest method, which allows sending HTTP requests and getting responses asynchronously with respect to the page refresh in the browser. XMLHttpRequest (3,5) works just as if the browser were making normal requests to the server, but without the need of page refresh, thus allowing to update only small pieces of the page if needed. The resulting application is much more responsive and user friendly, as users spend significantly less time waiting for requests to process. The requests are collected and processed by Java Server Pages at the server-side.

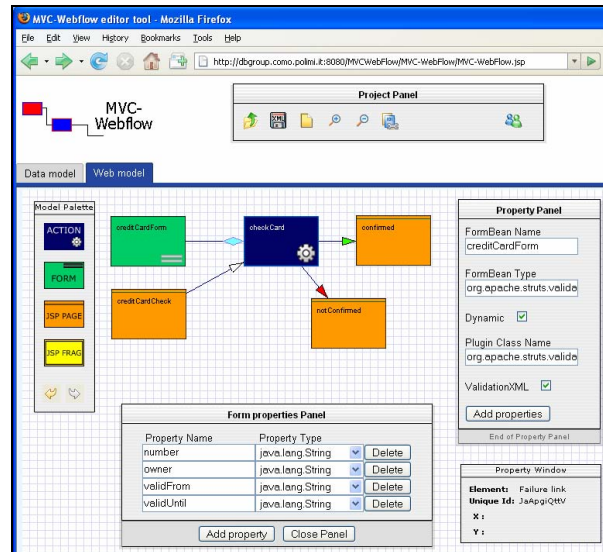
The editor allows the user to login, edit and save his projects on the server, and to reload them in the future. The server-side storage is implemented in a MySQL database. When the user want to load the project the editor makes a parsing of the XML string and generates all the figure objects and connections of the diagram. In the following figure you can see the

overall architecture of the system. The tool also features partial code generation of the elements needed by a Struts MVC application.

#### 4.1 Tool Interface and Sample Application

To show the approach at work, Figure 3 presents a very simple piece of Web application specified with MVC-Webflow and edited within the MVC-Webflow online editor. The example diagram shows a piece of Web application that receives the credit card information from the user and validates the card. It includes a creditCardCheck JSP page, that leads to a checkCard action (that includes the creditCard form). The successful outcome of the checking leads to the confirmed JSP page, while a failure leads to the notConfirmed JSP page.

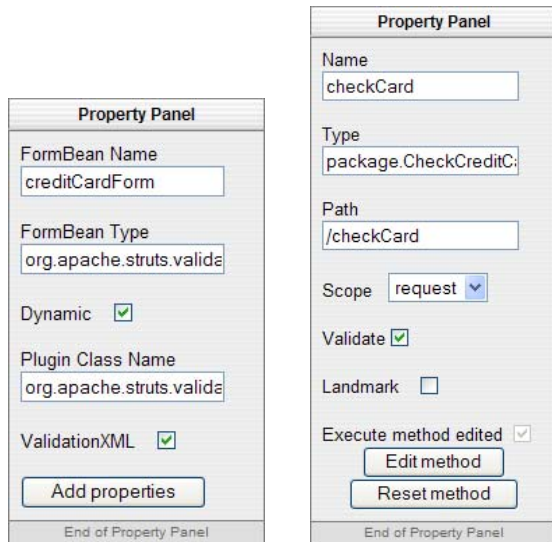
The editor interface allows the designer to edit, save and retrieve projects in the Web browser. The properties of the model elements can be specified within the property panel on the right (and a popup in the middle).



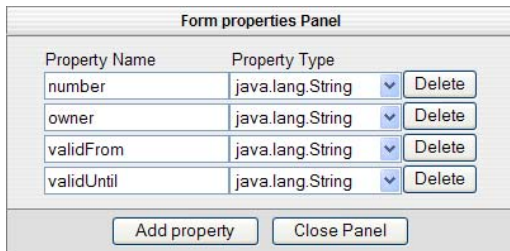
**Figure 3.** Example of MVC-Webflow diagram

The properties of the components are saved in the XML string describing the project and are used for the generation of the controller configuration file of Jakarta Struts. Notice that the business logic behind each application action is not managed by the tool and must be implemented by the developer.

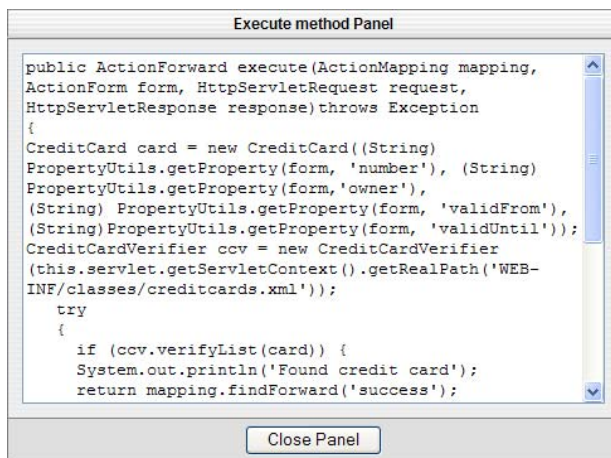
Figure 4 shows two examples of filled-in property panels respectively for the form element and for the action element of the sample application. The properties to be set are the ones listed in Table 1.



**Figure 4.** Property panels for the creditCard Form and for the checkCard Action.



**Figure 5.** Form properties Panel showing the definition of the form fields.



**Figure 6.** Execute method panel showing the Java code for the checkCard Action.

Figure 5 and Figure 6 show additional details about the elements described above. Figure 5 shows the panel for specifying the list of fields of a form: in particular,

the fields of the creditCardForm are shown. Figure 6 shows the Execute method panel, where the developer can write the Java code for the execute method of the action.

## 4.2 Code generation

As already mentioned, the tool includes some basic code generation features. In particular, the tool currently produces the following components of the Struts MVC framework:

- Web.xml, describing the deployed application;
- Struts-config.xml, specifying the behaviour of the controller;
- Validation.xml, describing the validation rules for the user input ;
- JSP pages, implementing a sketch of the JSP pages, including the basic elements that can be inferred from the MVC-Webflows diagrams;
- Action classes, implementing the standard Action structure, filled in with the details provided by the user (e.g., in the Execute method panel).

Figure 7 shows the automatically generated Struts-config.xml configuration file for the credit card application. The generated code infers information from the flow diagram representation and from the objects properties. The result includes the form structure, the action-mappings elements which define the real flow of the application (derived from the checkCard Action), and the inclusion of the Struts validation plugin, as specified in the property panel.

```

<struts-config>
<form-beans>
  <form-bean name="creditCardForm"
type="org.apache.struts.validator.DynaValidatorForm">
    <form-property name="number"
type="java.lang.String"/>
    <form-property name="owner"
type="java.lang.String"/>
    <form-property name="validFrom"
type="java.lang.String"/>
    <form-property name="validUntil"
type="java.lang.String"/>
  </form-bean>
</form-beans>
<action-mappings>
  <action path="/checkCard"
type="package.CheckCreditCardAction"
name="creditCardForm" scope="request">
    <forward name="success" path="/confirmed.jsp" />
    <forward name="failure" path="/notConfirmed.jsp"/>
  </action>
</action-mappings>
<plug-in
className="org.apache.struts.validator.ValidatorPlugIn">
  <set-property property="pathnames" value="/WEB-INF/
validator-rules.xml,/WEB-INF/validation.xml"/>
</plug-in>
</struts-config>

```

**Figure 7.** Generated struts-config.xml file.

## 5. Related Work

Several tools have been developed for engineering Web applications: among them, we can cite VisualWADE [5], WebRatio [1][8], WSDM editor [2], ArgoUWE [6], and Hera Presentation Generator (HPG) [3], and AWAC[4]. All of them are CASE tool that help formalize and automate the production of Web applications, with different focuses and flavours. However, they are often very formal and quite far away from the common practice of Web developers. Our solution adopts a rather opposite position with respect to other approaches, in that it attempts to maximise the simplification of the whole Web application modeling and subsequent implementation. To achieve this objective, some simplifications in the resulting Web application have been introduced: we assume that no AJAX interactions are included in the pages, and that the application structure consists of a basic set of page navigations and server side computations mainly defining of a sequence of input, management, and output of data.

## 5. Conclusions and Future Work

In this work we presented a simple conceptual model for MVC-based Web applications, and a prototype online tool for editing and automatically generating the description of the Controller, which is often the most complex part of the applications. Being online, the editor allows quick (re)design and (re)deploy of the application, with a set of primitives based on MVC, that is very familiar to the Web developer.

Future work will include refinement and enrichment of the model, together with additional implementation effort to make the online tool suitable to public usage.

A first improvement of the model will consider more complex Action outcomes and add several outgoing links corresponding to the outcomes. Additional design facilities will be introduced for the management of complexity of big projects (i.e., facilitating reuse, depuration, team work, and so on).

Further activities can be envisioned in order to make the tool accessible not only to designer but even to end user, thus allowing them to directly design their own application online. However, a long way is still to be covered in this sense, since end user require a set of clearly understandable and predefined components to be used, without the need of writing code or bothering about the architecture components.

## 6. References

- [1] R. Acerbis, A. Bongio, M. Brambilla, S. Butti, "WebRatio 5: An Eclipse-Based CASE Tool for Engineering Web Applications", ICWE 2007, Springer LNCS 4607, pp. 501-505.
- [2] S. Casteleyn, "Designer Specified Self Re-organizing Websites", Phd thesis, Vrije Universiteit Brussel, 2005.
- [3] F. Franciscar, G.J. Houben, P. Barna "HPG: The Hera Presentation Generator". Journal of Web Engineering, Rinton Press, Vol. 5, No. 2, p. 175-200, 2006.
- [4] I. Garrigós, C.Cruz and J.Gómez, "A Prototype Tool for the Automatic Generation of Adaptive Websites", AEWSE07 workshop, ICWE 2007, Como, Italy.
- [5] J. Gómez, A. Bia, A. Parraga, "Tool Support for Model-Driven Development of Web Applications", AEWSE07 workshop, ICWE 2007, Como, Italy.
- [6] A. Knapp, N. Koch, F. Moser, G. Zhang, "ArgoUWE: A CASE Tool for Web Applications". EMSISE03, <http://www.pst.informatik.unimuenchen.de/~kochn>
- [7] MVC-WebFlow online model editor prototype. <http://dbgroup.como.polimi.it:8080/MVCWebFlow/>
- [8] WebRatio Web Site. <http://www.webratio.com>.