Chapter 4

Web Presentation Layer Architecture

In this chapter we provide a discussion of important current approaches to web interface programming based on the Model 2 architecture [59]. From the results we derive how to improve web presentation layer architecture. Enabling technology for this is the NSP concept of typed server side calls to server pages. The concept of higher order server pages is introduced, which enables even more flexible design.

The central architectural questions concerning web based system interfaces are located on the server side. We review current web application frameworks for building dynamic web pages. Web application frameworks consider only the presentation layer in a multi-tiered web application. Our considerations are based on an analysis of the problem addressed by these frameworks. Special attention is paid to proposed composition mechanisms. In that comparison we can analyze the technological contributions as well as the shortcomings of these approaches.

4.1 Model 2 Architecture

In practice the tight coupling of code with layout has become a drawback for server pages technology. Therefore, separation of business logic processing and presentation generation, called processing/presentation separation in the following for short, became a goal.

In the discussion on how to reach processing/presentation separation, Sun has become influential by proposing several server side architectures, therein the "redirecting request" application model - coined Model 2 architecture afterwards [143]. This model has become commonly known as following the Model View Controller paradigm. We will in due course outline that it is a misconception about Model View Controller if the Model 2 architecture is subsumed under this pattern. We therefore give an evaluation of the Model 2 approach

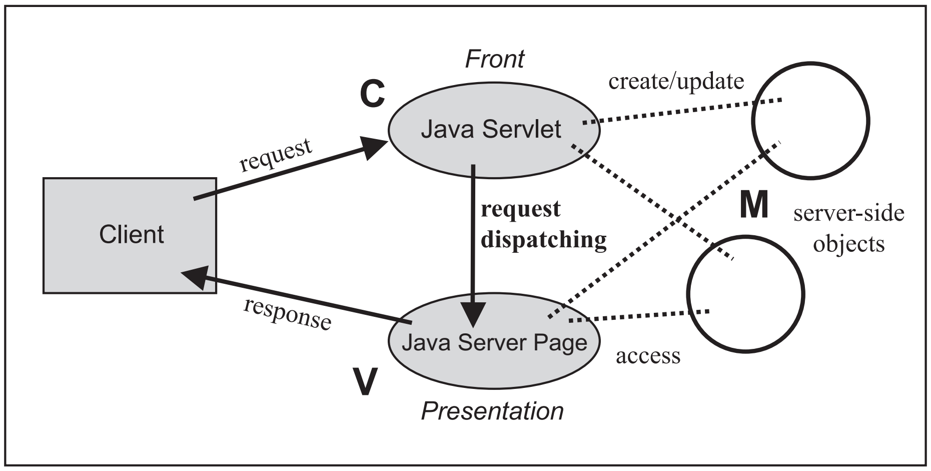

Figure 4.1: Model 2 Architecture. The figure visualizes the "redirecting request" application model coined Model 2 architecture. The model has become commonly known as following the Model View Controller paradigm. The server side objects are considered as model (M), the front components as controllers (C), and the presentation components as views (V).

without relying on the MVC argument.

The Model 2 architecture uses a threefold design in which the request is first directed to a front component, typically a servlet, which triggers the creation of a content object, typically a Java bean (Figure 4.1). The bean is then passed to a presentation component, typically a scripted server page where the data within the bean is embedded in HTML/XHTML. For Model 2 architecture some good practices are established on how to partition the request processing between the three parts. The most important recommendation is related to the use of the server pages: the server pages shall be used only for presentation purposes. Model 2 architectures can achieve a reuse of presentation components. If several front components generate under certain conditions the same output page, this page can be used from both components. Model 2 also allows separate maintenance of totally different response pages that may be generated from the same front component under certain conditions.

The Struts [50] framework is widely accepted as the open source reference implementation of the model 2 architecture. Struts proposes functional decomposition based on a proprietary composition approach in which business processing units do inform the controller object about the next processing step. Parameter passing between processing units is not established by the Java method parameter passing mechanism, but by emulating a parameter passing mechanism through transferring bean objects.

It is important to clarify a serious misunderstanding in architecture proposals for web site development. The web application frameworks following the Model

2 approach do not follow the Model View Controller paradigm. Model View Controller (MVC) [104] was introduced in Smalltalk and is a completely different concept. It only has superficial similarities in that it has three components from which one is related to the user interface, another to the application. However, the problem solved by the MVC paradigm is totally different. MVC is related to event notification problems within a GUI that provides different views on the same data, which have to be synchronized. MVC is renamed within the pattern community as observer pattern [75] and became an accepted general pattern for event model design problems[1]. The misnomer is even more astounding if one considers that the property of GUI's which makes MVC necessary, namely view update, i.e. push technology, is well known to be absent in the pull based approach of HTML/XHTML browsers.

The fact that web application frameworks rely on a misconception of the MVC paradigm does not necessarily imply that these frameworks have a bad design. But the argument for this architecture, namely that it follows a proven good design, is flawed. Only by recognizing that this argument is invalid the way is free for a new evaluation of the architecture and a recognition of advantages as well as drawbacks.

The Model 2 architecture defines a fixed decomposition combined with an intended separation of concerns. The incoming request is performed on the business model, then data are presented to the user in response. The difficulty with the approach lies not in the proposals for separation of concerns, but with the composition mechanism offered. The question is which semantics governs the interplay between the components: after you know how to divide, you have to know how to conquer.

The Model 2 architecture offers a complex communication mechanism based on the passing of beans. Beans are attached to a hashtable by the generating unit and retrieved from the hashtable by the target unit. In that way data is transmitted from the servlet to the scripted page. This mechanism is nothing more than a parameter passing mechanism, but without static type safety. The semantics of the composition paradigms of presentation and business logic is only conceivable by direct reference to the components found in the running system. In contrast, we will later use our NSP approach, where simple method call semantics is sufficient and allows for a sound architecture. Hence in the Model 2 architecture a considerable part of the architecture redefines a parameter passing mechanism which delivers no added value beyond method invocation. The Model 2 architecture therefore is still interwoven with a legacy technology driven design pattern that is far from creating a clear cut abstraction layer.

[1]The Java Beans component model relies on an observer pattern based event model. The GUI event model of the first Java version 1.0 followed the "chain of responsibility" design pattern [80], today it follows the observer pattern.

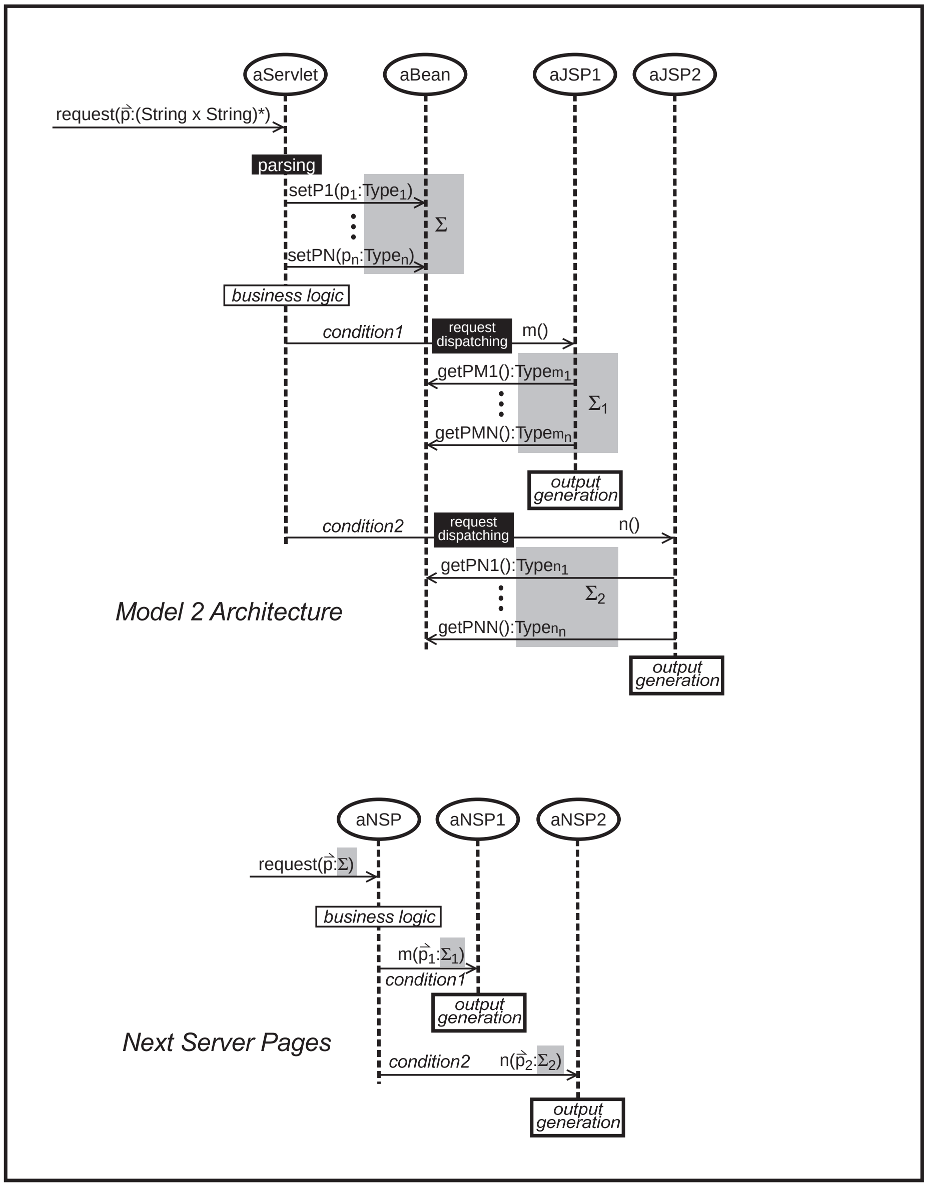

Figure 4.2: Model 2 Architecture versus NSP Functional Decomposition. The figure shows a typical control and data flow in a Model 2 architecture system up to details of request dispatching and the improvement of a counterpart system build on Next Server Pages technology by an interaction diagram.

4.2 NSP Functional Decomposition

NSP is open with respect to architectural decisions. NSP distinguishes between server pages that may be called across the net by a form or a link and server pages that may be called by another NSP server page on the server side in order to be included. The latter server pages has been termed dialogue submethods before. The NSP call mechanism for dialogue submethods has identical semantics as the Java method call with respect to parameter passing. It is the only composition feature that is needed to build sound and well understood web application architectures. NSP does not force the user into a specific design. With NSP no early decision between Model 1, Model 2 or other architectures is necessary.

The NSP approach to design can be seen as generalization of another proposal of the JSP specification, named "including requests" [143].

In order to call a server page from within another server page, the call-element is used[2]. The opening call-tag has a callee-attribute. Upon call the targeted server page generates a document fragment that replaces the respective call-element in the calling document. The output of the dialogue submethod must be a valid content element with respect to the context of the respective call element. Actual parameters are given to the called dialogue submethod by actparam-elements. The opening actparam-tag has a param-attribute, which has the same purpose as the the param-attributes of the NSP control elements. For each formal parameter of a dialogue submethod exactly one actual parameter must be given in every targeting call-element. Call elements may only contain actparam-elements, especially they cannot contain dynamic code[3].

The example in listing 4.1 consists of a main page and two dialogue submethods. The first dialogue submethod receives a String parameter and an int parameter and produces a message with respect to these parameters. The second dialogue submethod is called inside a table element and receives an array of Article objects. An Article object has three String properties x,y, and z. The dialogue submethod generates a table row for each object. A row contains three table data cells, one for each object property.

In the JSP technology parameter passing to a JSP differs fundamentally whether the JSP is called across the net or called on the server side. In the first case, parameters come as raw string data, as it is inherited from the old CGI mechanism. However, if a server page is called locally, it is established coding practice to pass the parameters by a bean object attached to the request parameter. Hence, a page must be designed either to be callable from the net or to be callable from the server and in both cases the developer has to face a parameter passing mechanism different from any reasonable parameter passing

[2]Listing 4.1, line 8-11, line 14-16

[3]The call element provides the NSP equivalent to the JSP request dispatching include mechanism. Analogously NSP supports redirect and server-side redirect, i.e. forward, by appropriate elements in the same way as the call element.

Listing 4.1

```
<nsp name="mainPage">
  <html><head><title>Some Page</title></head>
    <body><java>
      String customer;
      int age;
      Article[] articles;
      // get data for variables customer, age, and articles </java>
      <call callee="prelude">
        <actparam param="customer">customer</actparam>
        <actparam param="age">age</actparam>
      </call>
      <table>
        <tr> <td>X</td> <td>Y</td> <td>Z</td> </tr>
        <call callee="tableContent">
          <actparam param="articles">articles</actparam>
        </call>
      </table>
    </body>
  </html>
</nsp>

<nsp name="prelude">
  <param type="String" name="customer"></param>
  <param type="int" name="age"></param>
  <include>
    Hello Mr. <javaexpr>customer</javaexpr> !
    <!-- other ouptut with respect to customer and age -->
  </include>
</nsp>

<nsp name="tableContent">
  <param type="Article[]" name="articles"></param>
  <include><java>
     for (i=1;i<articles.lentgth;i++) {</java>
       <tr>
         <td><javaexpr>articles[i].getX()</javaexpr></td>
         <td><javaexpr>articles[i].getY()</javaexpr></td>
         <td><javaexpr>articles[i].getZ()</javaexpr></td>
       </tr><java>
     }</java>
  </include>
</nsp>
```

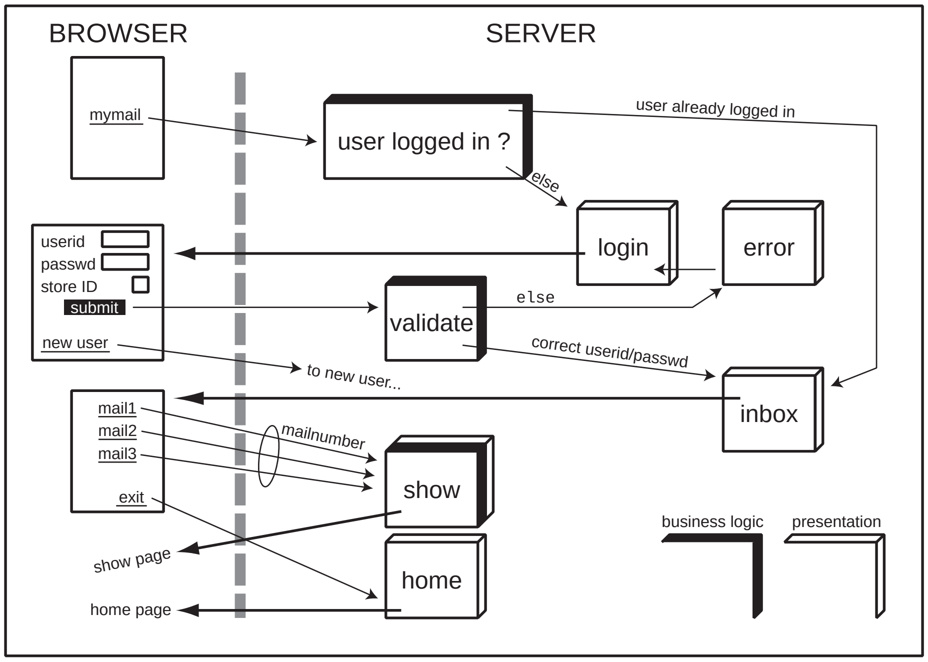

Figure 4.3: Example Interaction Diagram. The figure shows the login dialogue of a web based mail account. The user logs in and views her inbox. If she stores her password, for a certain time no login is necessary.

mechanism[4]. In NSP in contrast parameter passing is identical whether the page is called over the net or within the server. In both cases the parameter passing mechanism is essentially identical to the parameter passing encountered in Java. The parameters of a page in NSP behave identical to local variables in the Java code, in fact they are local variables initialized by the actual parameters. The difference is visualized in Figure 4.2. It follows from the explanation of the NSP implementation in chapter 5 that this transparency in the parameter passing mechanism comes at virtually no additional cost compared to the approaches in web application frameworks.

NSP allows for arbitrary application architectures based on functional decomposition. NSP frees the developer from considering the implementation details of the parameter passing mechanisms. Hence all special runtime entities that are needed in NSP to deliver the method call semantics are hidden from the developer. Processing/presentation separation is in first place a pattern for source code organization. NSP allows to solve the challenges in process-

[4]In the Servlet request dispatching mechanism, it is possible to attach new name/value pairs to the request object URL before invoking another Servlet. The JSP technology provides a JSP standard action, i.e. the param-action, for attaching new arguments for included server pages. However both of these are no parameter passing mechanisms, because only string parameters can be attached in an uncontrolled manner.

ing/presentation separation without referring to system architecture. In contrast, in NSP the functional decomposition mechanism allows for the desired separation of concerns. In Figure 4.3 we give an interaction diagram which shows the login dialogue of a web based mail tool. The user logs in and views her inbox. If she stores her password, for a certain time no login is necessary. In the given example the depth of decomposition is adapted according to the complexity of the respective functionality. The login screen is used for the initial login screen as well as for the login screen after an invalid login attempt. Viewing a mail is realized as a simple server page call. The example demonstrates the openness of NSP for different architectures.

4.3 Higher Order Server Pages

NSP formal server page parameters may receive server pages again. This introduces the notion of higher order server pages. The higher order server pages concept can be exploited to foster system maintainability and system part reusability.

NSP introduces the Page type as single proprietary type to be used for a formal parameter in addition to the types of the amalgamated programming language. Formal parameters of Page type may be used as callee-attributes of opening forms, links, and calls. For a formal Page parameter of a dialogue method either one hidden control, single select menu or at least two radio buttons must be provided in every form targeting the method. The value provided by the control must be the name of an existing server page that belongs to the system or again a formal Page parameter. A web signature with a formal Page parameter may be equally targeted by a hyperlink or, if the web signature belongs to an include server page, by a server-side call, with appropriate usage of hidden parameters respective actparam-elements.

Listing 4.2 gives a simplified example of a typical dialogue cycle:

input page - server-side validation - error page

A registration page includes a registration form by a server-side call. The registration form offers a text input field. The default value of this field is a parameter of the registration form submethod. The registration page chooses an empty string as default for the input field. Importantly the form provides its encompassing include server page as actual parameter. It is used by the targeted server page as error page. On submit the targeted server page first checks a business rule concerning the customer name. If no error occurred, the customer name is processed and the dialogue is continued by forwarding to another server page. Otherwise the form that targeted the server page is redisplayed and serves as a simple error page. This time the last user's input for the customer name is the default value for the input field. The example given in listing 4.2 is an instance of a more general, common design problem that can be given a reusable, flexible solution based on a higher order server page concept. Consider the case that several forms target the same dialogue method, which

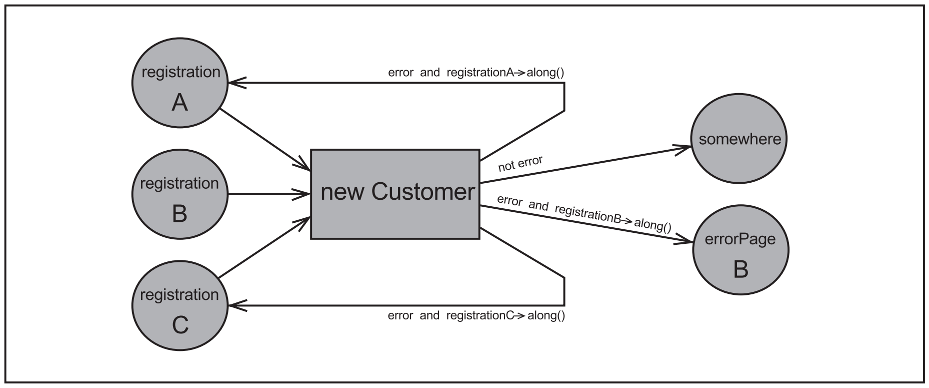
registration A
registration B
registration C
new Customer
error and registrationA->along()
not error
somewhere
error and registrationB->along()
errorPage B
error and registrationC->along()

Figure 4.4: Example Form Chart Diagram. The user dialogue given by a form chart in this figure poses a typical design problem that can be given a reusable, flexible solution based on a higher order server page concept.

processes the data and branches the dialogue flow, whereas the next page in the dialogue depends on the form that triggered the server page. Without higher order server pages the developer must explicitly keep track of the dialogue and must switch to the correct next page accordingly, which is an instance of a design that suffers an "ask what kind" antipattern[5]. Figure 4.4 shows a feature [58] containing three registration pages similar to the one given in listing 4.2. The feature is visualized as a form chart [60][61]. The server action for processing new customer data validates submitted user entry and presents an error page to the user if necessary. In the case that the action has been triggered by the first or third registration page, the respective page is redisplayed[6]. In the case that the action has been triggered by the second registration page, a specific error page is presented to the user.

[5]Our view of "don't ask what kind" is discussed in 8.1.2.

[6]Dialogue Constraint Language [60] is used to express the so called flow conditions in Figure 4.4.

Listing 4.2

```
<nsp name="Registration">
  <html>
    <head><title>Registration</title></head>
    <body>
      <call callee="RegistrationForm">
        <actparam name="defaultCustomer"> "" </actparam>
      </call>
    </body>
  </html>
</nsp>

<nsp name="RegistrationForm">
  <param name="defaultCustomer" type="String"></param>
  <include>
    <form callee="NewCustomer">
      <input type="String" name="customer">defaultCustomer</input>
      <hidden name="errorPage">RegistrationForm</hidden>
    </form>
  </include>
</nsp>

<nsp name="NewCustomer">
  <param name="customer" type="String"></param>
  <param name="errorPage" type="ServerPage"></param>
  <java>import myBusinessModel.CustomerBase;</java>
  <html>
    <head><title>NewCustomer</title></head>
    <body><java>
      if (CustomerBase.validate(customer)) {
        CustomerBase.createCustomer(customer);</java>
        <forward callee="Somewhere"></forward><java>
      } else {</java>
        <call callee="errorPage">
        <actparam name="defaultCustomer"> customer </actparam>
        </call><java>
      }</java>
    </body>
  </html>
</nsp>
```