An Overview of State-of-the-Art Architectures for Active Web Sites

Technical Report B-02-07

Dirk Draheim and Gerald Weber
Institute of Computer Science
Free University Berlin
email: {draheim,weber}@inf.fu-berlin.de

March 2002

Abstract

In this paper we provide a discussion of important current approaches to web interface programming based on the Model 2 architecture. From the results we derive how to improve the web presentation layer architecture. Enabling technology for this is NSP, a typed, composable server pages technology.

Contents

1 Introduction

Today the single most important technology for web applications still are HTML interfaces. Web interfaces guarantee maximum reach to the customer and the highest compatibility across platforms. Dynamic generation of pages is the key mechanism for building such interfaces. The central architectural questions concerning these interfaces are therefore located on the server side.

In this paper 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. Based on that we propose our own approach, NSP, which is designed to overcome these problems.

In section 2 we outline the driving forces for web application frameworks as they have been proposed. We then discuss in section 3 the current solutions to these challenges given by the Model 2 frameworks as well as their motivations. In section 4 we give our own analysis of the problem domain addressed by the Model 2 architecture and evaluate these approaches. In section 5 and section 6 we give an overview of NSP, an improved server page technology, and outline how it can be used in application architectures. Finally we give in section 7 an outline of the implementation.

2 Motivations for Web Application Frameworks

Approaches for dynamic generation of HTML are even today still based on the protocol introduced for the CGI interface of the first web servers. Server side HTML technologies can be divided into server scripting and server page approaches. Both approaches differ only with respect to the focus: server scripting approaches consider the generator for a page as classical code unit in the scripting language, while server pages consider the server components as HTML pages with embedded script code. Typically both approaches are available for each scripting language. Server pages have been considered as intuitively advantageous with respect to WYSIWYG HTML editors. These editors are supposed not to change the script tags they encounter. However, in practice the tight coupling of code with layout has become a drawback for server pages. Therefore, separation of business logic processing and presentation generation, called processing/presentation separation in the following for short, became a goal.

We concentrate on Java as a scripting language, so the discussed frameworks are immediately comparable. In the Java community the HTTP-servlet mechanism is the undisputed object oriented wrapper for an HTTP request handler. The JSP approach is an only slightly concealed HTML embedding of the servlet mechanism.

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 [7]. 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 without relying on the MVC argument. In section 4 we will critically reflect on the current practice to use system architectures as a means of implementing processing/presentation separation. First we present the architecture of these approaches.

3 Web Application Frameworks for Ultra-thin Client Systems

Model 2 web application frameworks typically use a servlet as a front component and a scripted server page as a presentation component. An elaborated web application framework based on this proposal is Struts [4]. An independent approach is the webmacro technology [8].

3.1 Principles of the Model 2 Architecture

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 (Fig.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. Webmacro uses server pages with a special interpreted scripting language different from Java while other approaches rely on Java Server Pages. 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. The Struts framework is widely accepted as the 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.

3.2 The "System Calls User" Approach

An interesting research project was the language Mawl [1]. In Mawl the control flow in the server script spanned the whole user session. The server script was suspended whenever a page was presented to the user. The approach could be seen as a "system calls user" approach, since the process of presenting a page to the user and retrieving the input from her had the semantics of a function call from the viewpoint of the script: the data presented to the system are the parameters of the procedure, the data entered by the user are the return values. However, Mawl has not prevailed, since it allowed only for one form per page, hence it abandoned the core paradigm of hypertext. Principally a workaround would be possible by emulating several forms and links as one single superform. This however would lead to a bad design of systems built with this technology, which would suffer from an "ask what kind" antipattern: for every form hard-wired case structures would have to be used to branch the session flow. This would imply high coupling and low cohesion. A project following the Mawl approach is the Bigwig project [2], which did not overcome the design problems mentioned above.

4 Analysis of Web Application Frameworks

4.1 Specific Problems in Request Processing for Web Applications

Web applications are used in a submit/response style. The server system is only contacted if the user submits a form or uses a link. The browser of the user then waits for a response page. In the browser of the user this leads to a page change. The complete old page is replaced by the response page, which is completely new parsed and rendered. We do not consider frames in this context. This interaction style leads to a fundamental difference of HTML interfaces compared to GUI's. With respect to presentation HTML interfaces are conceptually clear and rather simple. The task of the presentation logic is to create the whole new response page. This can be done in a virtually functional style. After the user request has come in, the control component in the presenation layer has to perform simple side effects: the system state may be updated, then information necessary for the page generation has to be gathered from the business logic tiers, then from this information the output page has to be generated. We will later argue that tasks like this are well understood and simple functional decomposition as it is achieved with subprograms is fully sufficient for this task.

4.2 Misconception of the Model View Controller Paradigm

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 [6] 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 [5]. 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 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.

4.3 Component Interaction in the Model 2 Architecture

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 runtime architecture, i.e. 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.

4.4 Reuse of Presentation Components

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.

5 Strongly, Statically Typed, Composable Server Pages

Based on the findings made so far we propose an improvement of web presentation layer architecture. For this we recur on a solution for server page programming, NSP [3], which is open with respect to architectural decisions. NSP offers a procedure call semantics for server pages which views every server page as a strongly typed procedure. Server pages can call each other. The NSP method call mechanism 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 delivers to scripted server pages the advantages of statically typed languages without forcing the user into a specific design decision. With NSP no early decision between Model 1, Model 2 or other architectures is necessary. NSP can be seen as generalization of another approach of the JSP specification, named "including requests" [7]. NSP as a language is realized as a set of XML tags. Therefore NSP documents are well formed XML documents.

5.1 Strongly Type System for Server Pages

The NSP approach is derived from the following basic considerations about client/server interaction in web applications. In web interfaces, server requests are triggered by the user following a link or submitting a form. In the case of submitting a form, the contents of the input fields are sent together with the command to the server. Hence submission of a form can be seen as a method call with a parameter list to the server. In web applications links can be used to submit parameters in the same manner. Hence all user interaction with the server, regardless of via links or forms, can be seen as remote method calls by the user. However, the HTTP request is an untyped remote method call. NSP starts with offering a static typing mechanism for client/server communication. The parameters of a server page can be specified in a type system which allows primitive as well as user-defined types. User defined types can be lists, records, and combinations thereof. NSP offers a static type check which is performed on the complete collection of server pages for a web application. Such a collection is called an NSP collection. The static type check compares forms with the server page that is called by the form and checks whether the signature of both do match. The static type check works even for server pages which demand complex types, e.g. a user defined record. In this case, an input field for each primitive typed element of the record must be provided. One of the most advanced features of the NSP type check is that it works even in server pages which use control structures. This is achieved by the so called NSP coding rules.

The parameters provided by the request are presented to the script in the page in the native types of the language. NSP consequently enforces the view that server pages are strongly typed functions in the scripting language. NSP even offers advanced widgets, which perform client side type checks and allow to specify declarative concepts like required form fields.

5.2 Functional Decomposition of Server Pages

The HTML output generated by a server page can be seen as a special return value concept. In a further project formal semantics for this concept will be given. It should be noted that the page output cannot be further processed within most of the server page mechanism. Once textual output is generated, filters and stylesheets are the more adequate solution for further processing. Hence, good practice coincides here with the restrictions of the server page concept. However, there is still need for functional decomposition on the server side. The developer should be able to create separate server pages which provide commonly used elements of several pages and reuse these pages. This desired behavior can be interpreted within the paradigm of special return values: if one server page A calls another server page B, this automatically implies that the special return value of B is textually included into the special return value of A and cannot be used in any other way by A. In NSP other pages can be called from a page with a special tag. Such calls follow the semantics just described with respect to the output. Similar inline commands like this one in

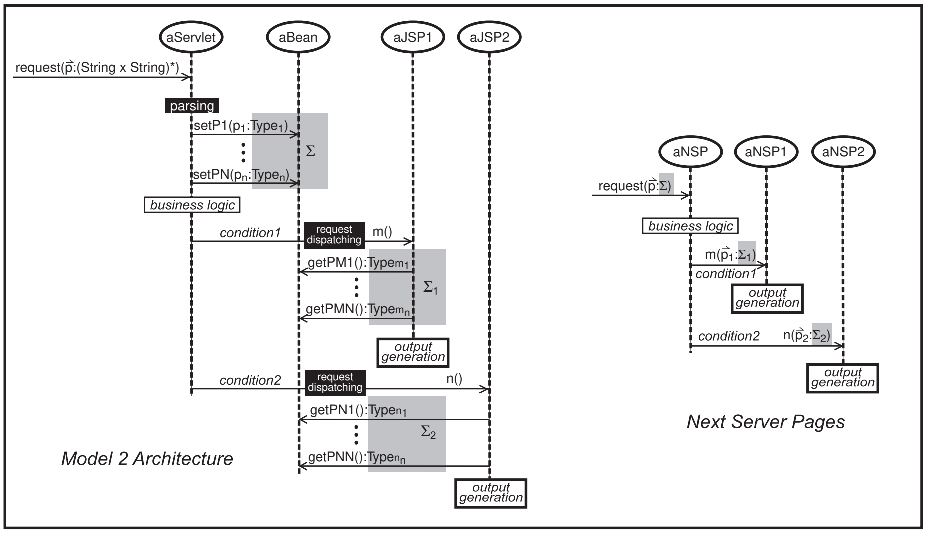

Figure 1: The figure shows a typical control and data flow in a Model 2 architecture system up to details of request dispatching and the improvement of a counterpart system build on Next Server Pages technology.

NSP are available in JSP and therefore in web application frameworks as well. However, the NSP mechanism is special with respect to the parameter passing mechanisms.

5.3 Parameter Passing to Server Pages

In JSP, 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 form 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 mechanism. 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 (Fig.1). This is the same semantics as it is known from the method call in Java. Consequently, in NSP calling a page across the net follows the remote method call semantics while calling on the server side follows the local method call semantics. In the section on implementation of NSP we outline that this transparency in the parameter passing mechanism comes at virtually no additional cost compared to the approaches in web application frameworks.

5.4 An Introductory NSP Example

The following example shows the implementation of a login dialogue. The dialogue starts with a login page. The customer enters her user identification, which is simply a number, and her password. If she enters a wrong combination of user identification and password, an error page shows up, which contains an error message and again the login capability. Otherwise the respective user is welcome. We use a shorthand notation for user defined types in the example.

```
Login {
  int userid;
  String passwd;
}

Customer {
  int userid;
  String passwd;
  String name;
  ...
}

<nsp name="Login" kind="presentation">
  <head><title>Login</title></head>
  <java>import myBusinessModel.CustomerBase;</java>
  <body>
     <form callee="Validate">
       <input widget="intfield" param="login.userid"></input>
       <input widget="textfield" param="login.passwd"></input>
       <submit></submit>
    </form>
  </body>
</nsp>

<nsp name="Validate" kind="business">
  <java>import myBusinessModel.CustomerBase;</java>
  <param name="login" type="Login"/>
  <body>
    <java>
    if (! (CustomerBase.validLogin(login))) {
      </java><call callee="Error"></call><java>
    } else {
      Customer c=CustomerBase.getCustomer(login);
      </java>
      <call callee="WelcomeCustomer">
        <actparam param="customer">
          c
        </actparam>
      </call>
      <java>
    }
    </java>
  </body>
</nsp>

<nsp name="Error" kind="presentation">
  <head><title>Error</title></head>
  <java>import myBusinessModel.CustomerBase;</java>
  <body>
     Invalid login ! Please try again !
     <call callee="Login"></call>
  </body>
</nsp>
```

```
<nsp name="WelcomeCustomer" kind="presentation">
  <head><title>Welcome</title></head>
  <java>import myBusinessModel.CustomerBase;</java>
  <param name="customer" type="Customer"/>
  <body>
     Hello Mr. <javaexp>customer.name</javaexp> !
     ...
  </body>
</nsp>
```

The above code exemplifies a selection of important NSP language constructs. The form of the `Login` page targets the `Validate` page. The `Validate` page has one parameter of user defined type `Login`. Path expressions are used in the form of the `Login` page to gather the fields of the `login` parameter. It is checked at compile time, if the form in its entirety matches the type of the targeted server page. In the form, a special NSP widget is used to gather the `integer` value. The widget dynamically checks the type of the entered value and prevents the form from being submitted if necessary. Therefore server side dynamic type errors cannot occur. In the `Validate` page, the `WelcomeCustomer` page is called. Like form input capabilities, an actual parameter of a call explicitly targets a formal parameter by referencing its name.

6 Application Architectures with NSP

NSP allows for arbitrary application architectures based on the NSP functional decomposition. NSP frees the developer from considering the implementation details of the parameter passing mechanisms. Hence all special runtime architecture which is needed in NSP to deliver the method call semantics is hidden from the developer.

6.1 Processing/Presentation Separation in NSP

As we have pointed out earlier, processing/presentation separation is in the first place a pattern for source code organization. NSP allows to solve the challenges in processing/presentation separation without considering system architecture. Instead in NSP the functional decomposition mechanism allows for the desired separation of concerns.

6.2 Example of Architectural Styles with NSP

In Fig.2 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.

7 Implementation of NSP

NSP is a server page language together with a semantics for functional decomposition. NSP comes with a type checker. The current reference implementation of NSP follows a generator approach which translates NSP into JSP. It does not imply a runtime library for the key method call mechanism.

The reference implementation of NSP maps a page header to a piece of generated Java code which makes the page parameters available for the following java code. To this purpose the generated code defines local variables which are initialized with the passed parameters. Parameter passing is realized in different ways for calls via HTTP and local calls. In case of local calls the parameters are passed as bean. In the case of calls over the net the parameters are retrieved from

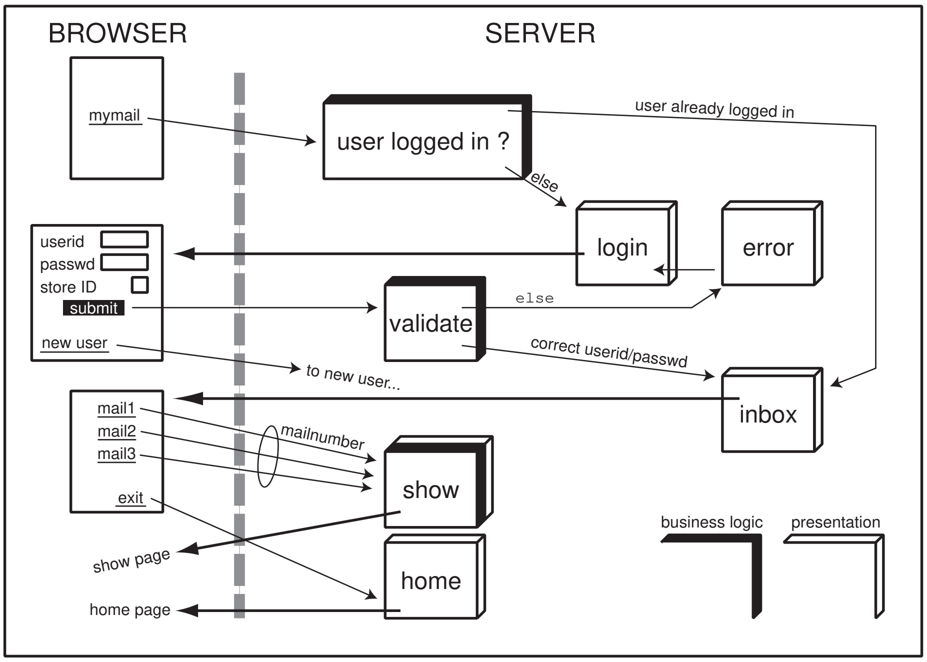

Figure 2: Example interaction diagram. The figure shows the login dialogue of a web based mail account. The user logs in and views her inbox. If she stores her password, for a certain time no login is necessary.

the request, then parsed and type checked. In both cases the parameters are accessible in the same manner for the java code in the page.

8 Conclusion

Our analysis of web application frameworks based the Model 2 architecture has shown:

- The subsumption of these frameworks under the Model View Controller paradigm is deeply flawed and based on a misunderstanding of Model View Controller.
- Within these frameworks a separation of concerns can be reached and the content parts as well as the presentation parts can be placed in different code units.
- The composition mechanism used for combining the different code units is proprietary and based on consideration of the runtime object structure of the system. No abstraction layer has been reached.
- The Model 2 approach unfolds complex architecture at a place where architecture is certainly not needed. Page request handling is simple because it has no inherent concurrency problems: The request is fully received, can be processed, and the result can be produced. This allows a functional decomposition in a straightforward way as it is rarely possible in today's complex system landscape. If a design problem for which the simple method call mechanism is the adequate solution is realized by a complex code architecture, this should be considered as bad design.

We recognized the following as crucial for improving web presentation layer architecture:

- We propose functional decomposition as sufficient and appropriate decomposition mechanism for server side programming, which allows, but is not restricted to the earlier discussed architectures.

- In the used server page mechanism, remote and local call have the same style, but different semantics.

References

[1] David Atkins, Thomas Ball, Glenn Bruns, and Kenneth Cox. Mawl: a domain-specific language for form-based services. In IEEE Transactions on Software Engineering, June 1999.

[2] Claus Brabrand, Anders Møller, Anders Sandholm, and Michael I. Schwartzbach. A runtime system for interactive Web services. Computer Networks, 31:1391-1401, 1999. Also in Proceedings of the Eighth International World Wide Web Conference.

[3] D. Draheim, G. Weber, "Strong complex typed dialogue-oriented Server Pages", Technical Report No B 02-05, Intitute of Computer Science, Free University Berlin, March 2002, http://www.inf.fu-berlin.de/inst/pubs/tr-b-02-05.ps.gz

[4] Malcolm, D. Struts, an open-source MVC implementation. In: IBM developerWorks, February 2001.

[5] Gamma, E. et al. Design Patterns - Elements of Reusable Object-Oriented Software. Addison-Wesley, 1995.

[6] Krasner, G.E., Pope, S.T. A Cookbook for Using the Model-View-Controller User Interface Paradigm in Smalltalk-80. In: Journal of Object-Oriented Programming, August/September 1988 26-49

[7] Pelegri-Llopart, E.; and Cable, L. Java Server Pages Specification, v.1.1. Sun Press, 1999.

[8] Webmacro. http://www.webmacro.org/, 2002.