# Projektgruppe: Generierung von Web Anwendungen aus visuellen Spezifikationen – Entwurfsmuster für Web-Anwendungen

Bastian Cramer

Universität Paderborn

**Zusammenfassung** In diesem Paper sollen die aktuellen Entwicklungen für Entwurfsmuster für Web-Anwendungen gezeigt werden. Im ersten Teil werden dazu zunächst einige Begriffe, insbesondere der Begriff Entwurfsmuster (engl. Pattern) erläutert. Danach wird beispielhaft der systematische Entwurf einer Webapplikation anhand von Pattern gezeigt. Zum Schluß wird konkret auf einige Entwurfsmuster für Webanwendungen und spezielle Entwurfsmuster für e-Commerce und Shopsysteme eingegangen.

## 1 Was sind Entwurfsmuster und wozu benötigt man sie?

Der erste, der den Begriff des Entwurfsmusters (engl. Pattern) erwähnte, war 1977 Christopher Alexander [12]. Ein Entwurfsmuster ist die Abstraktion eines Problems in einem bestimmten Kontext und die Repräsentation einer universellen Lösung, die immer wieder angewendet werden kann, oder mit Alexanders Worten:

*„Each pattern describes a problem which occurs over and over again in our environment, and then describes the core of the solution to that problem, in such a way that you can use this solution a million times over, without ever doing it the same way twice.“*

Heutige Webanwendungen sind äußerst komplex. Sie sind geprägt durch verteilte Prozesse, heterogene Systeme, große Datenmengen, einem Mix von Navigation, Layout und Inhalt, und sie haben zudem eine kurze Lebensdauer. Trotz dieser Eigenschaften werden Webanwendungen häufig relativ chaotisch erstellt. Es ist oft kein wirklicher Entwicklungsprozess vorhanden, und die Anwendungen werden ad-hoc erstellt. Dies steht im krassen Gegensatz zur Erstellung von herkömmlicher Software, die einen feingranularen Entwicklungsprozess durchläuft. Durch die Quick and Dirty Implementierung von Webanwendungen entstehen oft viele Nachteile [4]:

### 1.1 Nachteile herkömmlich entstandener Webanwendungen

- Schlechte Wartbarkeit: Wegen des Fehlens eines objektorientierten Designs und der Verstrickung von Logik und Layout, z.B. JSPs mit viel Java Code
- Keine Trennung von Layout und Logik: Designer und Programmierer kommen sich häufig in die Quere.
- Schlechte Testbarkeit des frontends: JSP Seiten sind schwer zu testen
- Schlechte Testbarkeit des backends: Da JSP Seiten das backend aufrufen, kann die Logik nicht alleine getestet werden.
- Schlechte Portierbarkeit auf unterschiedliche Client Geräte: unterschiedliche Clients benötigen auch unterschiedliche Anzeigeroutinen, z.B. WAP
- Schlechte Wiederverwendbarkeit, da keine Modularisierung vorhanden ist.

Sicherlich ist eine Herangehensweise, wie sie in dieser Arbeit später gezeigt wird, nur für größere Webanwendungen mit mehreren Duzend generierten Seiten interessant. Eine kleine Anwendung, wie z.B. ein Umfrageskript, benötigt sicherlich nicht diese Vorgehensweise.

Die Idee von Entwurfsmustern ist es also, die oben genannten Probleme zu umgehen.

### 1.2 Die Komponente

Oft wird im Zusammenhang mit Entwurfsmustern auch der Begriff der Komponente erwähnt. Eine Komponente wird hierzu definiert [2] als eine wiederverwendbare, innerhalb eines Komponentensystems unabhängige und vermarktbare Software, die über dokumentierte Schnittstellen verfügt und in Kombination mit anderen Komponenten eingesetzt wird. Eine Webanwendung kann also als eine Sammlung von individuell angepassten Komponenten angesehen werden.

### 1.3 Ein komponentenbasierter Ansatz: WebComposition

In [2] wird mit WebComposition ein komponentenbasierter Ansatz zur Modellierung von Webanwendungen vorgeschlagen: Dabei gibt es einen sogenannten Komponentenspeicher (Component Store) in dem persistente Komponenten liegen, auf die zugegriffen werden kann. Diese Komponenten können dabei einerseits sehr feingranular sein, also zum Beispiel kann eine Komponente nur die Schriftgröße manipulieren oder sie kann komplexe Aufgaben erledigen. Der WebComposition Ansatz ist dabei objektorientiert, es werden Vererbung und Aggregation angeboten. Im Component Store von WebComposition wird als Komponentenbeschreibungssprache die WebComposition Markup Language (WCML) [14] benutzt.

### 1.4 Das MVC Pattern als Grundlage

Wenn man über Entwurfsmuster spricht, dann kommt man um den Begriff MVC, was für Model-View-Controller steht, nicht herum. Das MVC Pattern beschreibt dabei die Aufteilung der Webanwendung in drei Bereiche. Das Model beschreibt Zustand und Daten der Webanwendung (siehe Abb. 1). Die View beschreibt die Sichten auf das Model. Wenn sich Zustände oder Informationen ändern, dann ändert sich auch die View. Der Controller bekommt die Benutzereingaben und verarbeitet sie. Danach verändert er das Model und erneuert schließlich die View. Vorteile des MVC Patterns sind, dass es zwischen Layout und Logik trennt. Dies ist für Webdesigner und Programmierer vorteilhaft, da ein Webdesigner nicht programmieren und der Programmierer kein Künstler sein muß. Zudem kann die Präsentation verändert werden, ohne die dahinterstehende Geschäftslogik zu ändern. Dies bringt auch Vorteile bei verschiedenen Endgeräten, wie Organizer, Handys oder unterschiedliche Browsertypen, die natürlich unterschiedliche grafische Oberflächen besitzen.

Das Jakarta Struts Framework [13] ist dabei einer der bekanntesten Systeme, die das MVC Pattern einsetzen. Nun noch ein Beispiel einer Iteration, einmal in einer JSP, also ohne MVC und einmal in Struts:

```
<%@ page language="java" contentType="text/html" %>
<%@ page import = "meinpackage.Datenbank" %>
<html>
<head>
 <title>JSP Iteration ohne MVC</title>
```

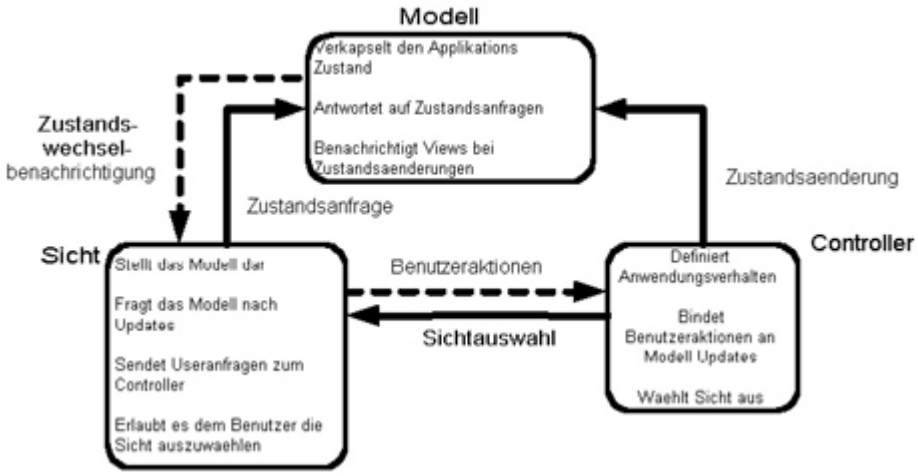



**Abbildung 1.** MVC Pattern

```
</head>
<body>
 <%
  Vector users  = Datenbank.getUsersVector();
  for (int i=0; i < users.size(); i++)
 out.println((String)users.get(i));
 %>
</body>
</html>
```

Jetzt dasselbe mit Struts:

```
<%@ page language="java" %>
<%@ taglib uri="../WEB-INF/struts-bean.tld" prefix="bean" %>
<%@ taglib uri="../WEB-INF/struts-html.tld" prefix="html" %>
<%@ taglib uri="../WEB-INF/struts-logic.tld" prefix="logic" %>
<html:html locale="true">
<head>
 <title>Struts Iteration</title>
 <html:base/>
</head>
<body>
 <logic:iterate id="user" name="Users">
      <bean:write name="user" property="name" />
 </logic:iterate>
</body>
</html>
```

Hier kann man erkennen, dass in der JSP Variante viel Logik in der Sicht implementiert werden muß. Das MVC Gestaltungsmuster wird also verletzt. Dieses Beispiel ist noch recht übersichtlich, werden jedoch Tabellen verwendet, so wird die JSP Variante immer unübersichtlicher, während die Struts View sich nicht wesentlich ändert.

### 1.5 Model 1 und Model 2 Architektur

In Bezug auf die J2EE Technologie [15] werden häufig die Termini „Model 1“ und „Model 2“ für zwei verschiedene Architekturarten erwähnt. In der „Model 1“ Architektur werden alle eingehenden und alle ausgehenden Anfragen direkt in der JSP Seite verarbeitet. Datenbankanfragen werden in einer oder mehreren Java Beans verarbeitet. Die JSPs verarbeiten dann die Daten und zeigen sie an. Die „Model 1“ Architektur ist nur für kleinere Webanwendungen geeignet, weil sie zu einer Verstrickung von JSPs mit Code führt und so für Designer sehr schlecht zu warten ist.

Bei der „Model 2“ Architektur verbindet man Servlets mit JSPs. Das Servlet steht dabei für die ausführende Ebene (Processing Tier), es behandelt alle ankommenden Requests und instanziiert die Java Beans. Die JSPs stellen nur die Sichten her (Presentation Layer). Es existiert kein Code mehr in der JSP. Dieser Ansatz kommt dem MVC Paradigma sehr nah.

### 1.6 Mehrschichten Architektur

Wenn man von einer mehrschichtigen Architektur (engl.: Multi Tier Architektur) spricht, meint man die Trennung der Applikation in mehrere Ebenen. Die eben erwähnten „Model 1“ und „Model 2“ Architekturen sind beides 3 schichtige Anwendungen. Die erste Ebene ist dabei die Sicht, evtl. auf unterschiedlichen Browsern, also das *Front-end*. Die zweite Ebene ist der *Application Server* mit der Logik und die dritte Ebene ist die Datenbank, also das *Back-end*. In der J2EE Technologie hat man eine 4 Schichten Architektur. Zum einen das Frontend, dann den WebContainer mit JSPs und Servlets. Dann den EJB Container, der für Transaktionsmanagement, Lifecycle Management und Ressource Management zuständig ist, sowie das Back-end. In der Literatur ist die Unterteilung der Schichten allerdings sehr unterschiedlich. Eine konkrete Definition gibt es nicht.

## 2 Ein systematischer Entwurf einer Webanwendung

Komplexere Webanwendungen, also solche mit Dutzenden von Seiten, sollten nicht ad hoc entworfen werden. Ein systematischer Entwurf hat hier letztendlich den Vorteil, dass er einfacher zu warten und zu erweitern ist. Zudem können objektorientierte Entwurfsmuster helfen, nicht im Chaos zu enden. In [5] wird hierzu ein Ansatz erläutert, dessen Schritte hier nachvollzogen werden sollen.

### 2.1 Transition Table

Zuerst sollte anhand einer sogenannten „Transition Table“ das Navigationsdesign der Webanwendung erstellt werden. Diese Transition Tables oder auch Storyboards sollen die interaktiven Szenarien der Anwendung wiederspiegeln. Sie haben starke Ähnlichkeit mit endlichen Automaten oder dem Konzept der Janssen Netze [17]. Ohne eine Spezifikation der Navigationslogik wäre die einzige Möglichkeit herauszufinden, wie man zu bestimmten Teilen der Website gelangt, sich den Quellcode anzuschauen. Dies ist für Erweiterungen des Systems natürlich nicht vorteilhaft. Deshalb werden in [5] Transitionsregeln vorgeschlagen. Sie haben die Form:

```
<action> => <next action 1> | <next action 2> | ...
```

Auf der linken Seite steht der Name der Benutzeraktion. Auf der rechten Seite stehen die Aktionen, die als nächstes gewählt werden können. Die Behandlung dieser Aktionen sollte in einem zentralen Verteiler (Dispatcher) geschehen. Dies wird im nächsten Teil dieser Arbeit gezeigt.

### 2.2 Site Map

Die Site Map soll die Organisation der Seiten darstellen. Auch andere wichtige Informationen wie zugriffsgeschützte Seiten oder dynamisch generierte Seiten sollen dargestellt werden. Es werden jedoch nicht alle Links dargestellt, sonder nur die wichtigsten.

### 2.3 Wireframes

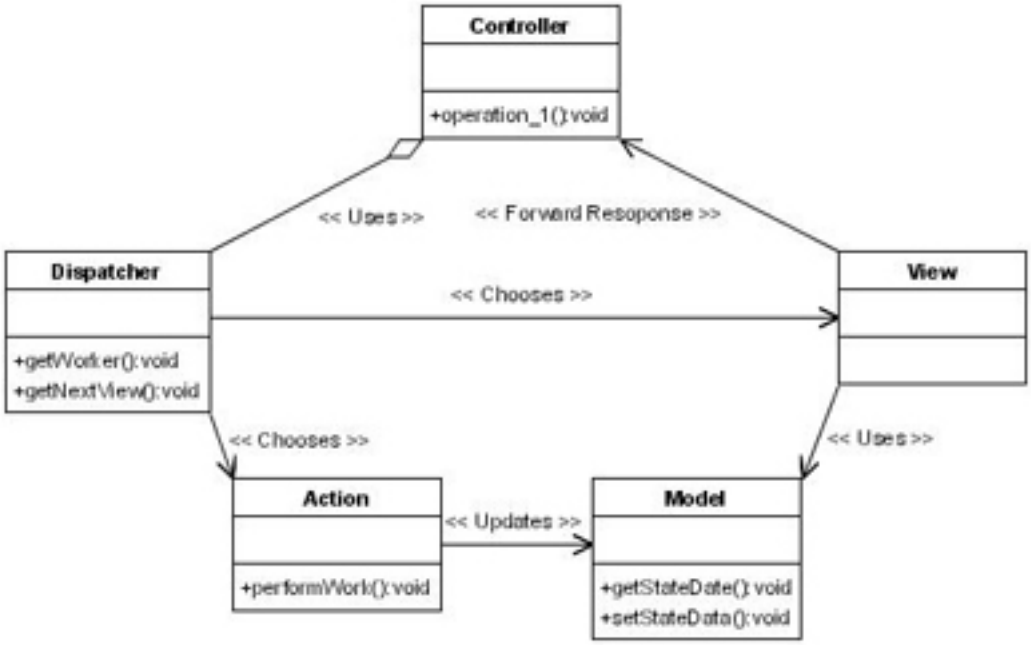
Wireframes zeigen bereits das grobe Layout einer Seite. Es werden wichtige Elemente der Seiten gezeigt, wie Suchboxen und Links, die in der Site Map ggf. noch nicht auftauchten.

## 3 Konkrete Entwurfsmuster

In diesem Teil der Arbeit sollen nun konkrete elementare Entwurfsmuster gezeigt werden. Die Entwurfsmuster lassen sich dabei noch in Kategorien unterteilen. So gibt es Entwurfsmuster für das Layout der Sichten und für die Navigationsstruktur. Hier werden besonders Aspekte der Usability beachtet. Für eCommerce Anwendungen gibt es spezielle Entwurfsmuster, die einen Benutzer möglichst lange auf der Website halten und zum Kauf animieren sollen. Der folgende Abschnitt ist jedoch auf Entwurfsmuster beschränkt, die für die Implementierung der Logik zuständig sind. Sie sind also insbesondere für den Entwickler gedacht und nicht für den Designer.

### 3.1 Zentraler Verteiler / Controller Servlet

Bei diesem Ansatz fungiert ein zentrales Servlet (Controller Servlet) als zentraler Einstiegspunkt (engl.: „Single-Point-of-Entry“). Dies steht im Kontrast zum herkömmlichen Weg, immer ein Skript für eine Anforderung zu benutzen und den Webserver quasi als Dispatcher zu benutzen. Dies ist bei kleinen Problemen sicherlich einfach und schnell. Bei größeren Anwendungen führt es jedoch zu Code Duplikaten und zu mehr Wartungsaufwand. In [11] wird dabei der in der Abbildung




gezeigte Vorschlag gemacht. Das Controller Servlet benutzt dabei eine Dispatcher Klasse, die je nach Anforderung verschiedene Aktionen ausführt, das Model aktualisiert und schließlich eine View aussucht. Dieser Ansatz erfüllt auch die Anforderungen, die sich ergeben, falls man eine Webanwendung für unterschiedliche Client Geräte benutzbar machen möchte. Der Dispatcher wählt einfach eine View aus, die für das Endgerät optimiert ist. Zudem wird hier strikt zwischen Logik und Präsentation getrennt. Die Controller Klasse kann dabei wie folgt implementiert werden, falls entsprechende Buttons mit Namen existieren:

```
my $q= new CGI;
my $action = $q->param("‘action"’) || "start";

# Dispatching
```

```
if ($action eq "start"){
start($q, ...);
}
elseif ($action eq "about"){
about ($q, ...);
}
...
```

Das Struts Framework benutzt bereits von Haus aus einen zentralen Dispatcher. Hier wird in der Datei struts-config.xml für jeden Request eine Action angelegt:

```
<action    path="/userLogin" //Mapping Name
    type="de.myPackage.UserLoginAction" //mit
     Mapping assoziierte Klasse
    validate="true"   // Eingabedaten validieren
    name="loginForm"  // assoziierte Java Bean
    scope="session"   // Gültigkeitsbereich
    input="/client/index.jsp">  // assoziierte JSP
    <forward name="success" path="/client/success.jsp"/>
      // Forwarding bei Erfolg
    <forward name="login" path="/client/index.jsp"/>
      // Forwarding bei Mißerfolg
</action>
```

Struts ist somit sehr komfortabel und übersichtlich.

### 3.2 Kontinuität

Ein Aspekt in komplexen Webanwendungen ist es, Kontinuität zu erreichen. Informationen müssen dabei von einer zur anderen Seite mitgenommen werden. In [5] werden dafür drei unterschiedliche Methoden vorgeschlagen:

1. Parameter in der URL kodieren: <a href=“skript.cgi?action=about“>About us</> Dieser Ansatz ist nur für relativ einfache URLs geeignet.
2. Speichern der Parameter in hidden fields: <form action=“skript.cgi?action=add> <input type=“hidden“ name=“isbn“ value=“05454447“> </form> Dieser Ansatz ist leichter zu lesen und man kann zwischen Parametern in den hidden fields und Aktionen in der URL unterscheiden.
3. Buttons mit unterschiedlichen Namen versehen: <input type=“submit“ name=“delete-0525454“/> <input type=“submit“ name=“delete-74469“/>

Ein anderer Weg, Daten in Java geschriebenen Webanwendungen zwischenzuspeichern ist es, das Session Attribut zu benutzen, hier mit Hilfe des Struts Frameworks:

```
HttpSession session = request.getSession();
// Session Objekt anfordern
session.setAttribute("USER",username);
  // username unter Key USER speichern
```

```
if (session.getAttribute("USER")==null){
// ist USER Objekt vorhanden?
    target="login";
    return(mapping.findForward(target));
  }
```

**Abstract Form**

In [8] wird mit dem Abstract Form Pattern eine weitere Möglichkeit vorgeschlagen Kontinuität zu erreichen. Hier werdene Daten validiert und über mehrere Formulare „mitgenommen“. Das Servlet generiert dabei „on the fly“ immer das nächste Formular, wobei in der anderen Richtung ein immer neues Servlet aufgerufen wird. Ein weiterer Punkt ist dabei das Validieren der Parameter, das durch einige wenige formale Regeln in der Object Constraint Language (OCL) [18] erreicht wird.

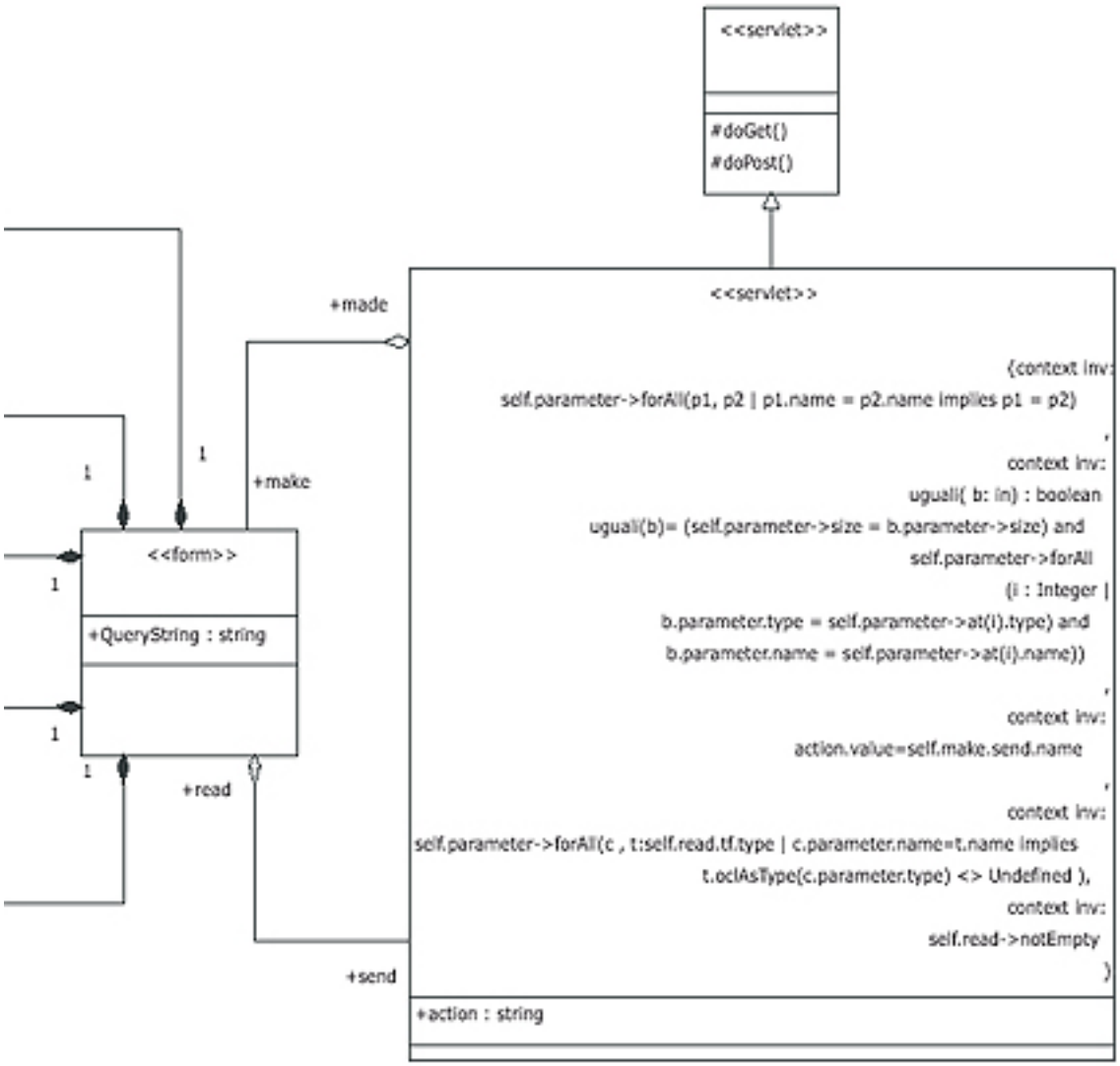
**Three Java Bean**

Anwendungen, die es erfordern Daten zu validieren, haben das Problem, dass drei verschiedene Arten von Daten auftreten können, zum einen Daten, die der User in Formularen eingibt, Daten, die in Eingabeformulare voreingetragen werden und Fehlermeldungen, falls Formulardaten nicht korrekt validiert werden konnten. In Java Webanwendungen gibt es nun die Möglichkeit, diese Daten in drei verschiedenen Beans abzuspeichern, wie in [4] vorgeschlagen. Dies hat den Vorteil, dass alle Daten wohl definiert sind und leicht verarbeitet werden können. Falls der Programmierer einen Fehler macht und Beans falsch zurückgibt, so wird dies beim ersten Testlauf erkannt. Programmierer und Anwender müssen sich nur einig über die Namensgebung der Beans sein. Dies kann leicht zu Beginn festgelegt werden. Dieses Pattern wird oft zusammen mit dem Controller Servlet/ zentraler Verteiler Pattern benutzt. Das Struts Framework benutzt auch dieses Pattern. Mit jeder Action wird eine standardisierte Java Bean assoziiert. Fehlermeldungen werden ebenfalls in einer Bean zurückgegeben.

### 3.3 Inhalt von Präsentation trennen

Wie bereits erwähnt ist es wichtig, den Inhalt vom Layout und Präsentation zu trennen. Falls man nicht ohnehin bereits ein MVC Framework wie Struts benutzt, gibt es einige Möglichkeiten [5] trotzdem eine solche Trennung zu erreichen. Der einfachste Ansatz ist es, Cascading Stylesheets zu verwenden. Allerdings können CSS nur definieren, wie der Inhalt formatiert wird.

Ein anderer Ansatz ist es, die Daten im XML Format vorliegen zu haben und diese dann mittels XSLT zu präsentieren. Leider ist das on the fly generieren mit XSLT zeitaufwändiger als z.B. Templates zu benutzen, die einfach ersetzt werden.




### 3.4 Sitzungsmanagment

In Webanwendungen ist es wichtig den Zustand des Systems festzuhalten. Das HTTP Protokoll ist jedoch zustandslos. Möglichkeiten Parameter mitzunehmen wurden bereits im Abschnitt „Kontinuität“ besprochen, allerdings sind die nicht immer ausreichend. So kann es passieren, dass Daten verloren gehen oder aus Sicherheitsgründen eine Speicherung innerhalb von beispielsweise hidden fields nicht möglich ist. Beim Online Einkauf kann es sein, dass Benutzer zusätzlich mit dem Back Button des Browsers navigieren. Die Einkaufsliste soll trotzdem übernommen werden. Eine

Möglichkeit ist hier die Benutzung von Cookies. In den Cookies sollten allerdings keine sicherheitsrelevanten Daten gespeichert werden. Diese sollten nur auf dem Server abgelegt werden. Im Cookie wird nur eine ID gespeichert am Besten verschlüsselt. Zusätzlich sollten in der ID Informationen über den Client wie IP Adresse, User Agent o.ä. gespeichert werden. Dies ist notwendig, da ein User die Session ID eines anderen Users erraten könnte und so unerwünschte Aktionen auslösen könnte. Dies muß vor allem in Shop Anwendungen gewährleistet sein. Ein Weg ein Sitzungsmanagent zu imlementieren findet sich mit dem Bus Class Form in [8]. Dabei melden sich die Servlets bei einem zentralen Manager an, der jedem Servlet einen eigenen Zustand anlegt.

### 3.5 Automatische Aktualisierung

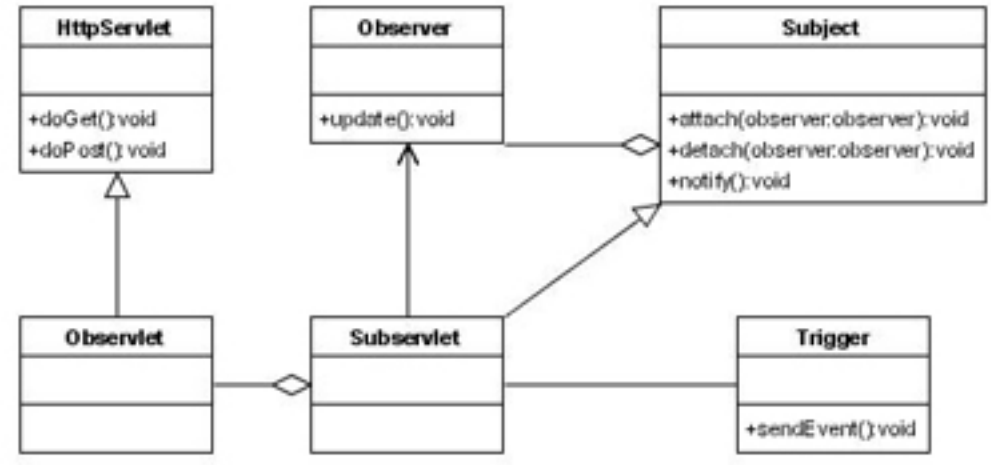



**Abbildung 2.** Active View Pattern

Bei manchen Webanwendungen kann es sein, dass ständig eine bestimmte View aktualisiert werden muß, obwohl der Benutzer eigentlich keine Aktion vornimmt. Ein ständiger Reload wäre nicht die eleganteste Lösung. In [8] wird mit dem Active View Pattern ein Vorschlag gemacht, dieses Problem zu umgehen. Verschiedene Subjects werden dabei von Observern beobachtet. Falls sich ein Subject ändert, werden die entsprechenden Observer benachrichtigt und die wiederum triggern die Events der Clients. Die Kommunikation zwischen Observlet und Subservlet könnte dabei über ein Standardprotokoll wie Java RMI geschehen.

### 3.6 Leistung

#### Client übernimmt Arbeit

Um schnelles Feedback vom Server zu erreichen, ist es von Vorteil, möglichst viel Arbeit dem Client zu überlassen. So kann man die Eingabevalidierung dem Client überlassen. Hier kann beispielsweise JavaSkript eingesetzt werden, um herauszufinden, ob Felder korrekt ausgefüllt wurden. Diese Variante ist natürlich nicht sicher, deshalb muß der Server trotzdem die Eingabedaten validieren. Man erspart dem Server jedoch einige unnötige Anfragen.

#### Active Proxy

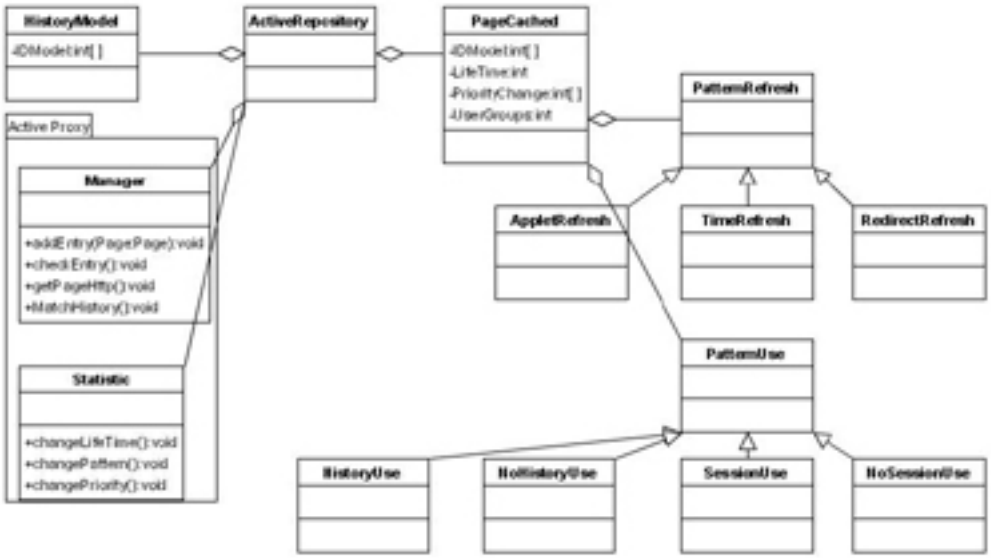



**Abbildung 3.** Active Proxy Pattern

Wegen des immer höher werdenden Interesses an eCommerce Anwendungen und Shop Systemen, wird auch die Serverlast immer größer. In [8] wird mit dem Active Proxy Pattern ein System vorgeschlagen, um die Serverlast möglichst niedrig zu halten. Es basiert dabei darauf, typische Seiten einer Anforderung bereits vorzuspeichern und bei Bedarf einfach auszuliefern. Es wird also eine Art Proxy implementiert. Dazu benötigt man eine Statistik Komponente, die ermittelt, wann eine Seite veraltet ist und diese dann aktualisiert.

### 3.7 Landestypische Formatierungen

Webanwendungen werden in der Regel auch im Ausland eingesetzt. Die Formatierungsregeln für Zahlen und Nummern können jedoch unterschiedlich sein. In [4] wird hierfür ein Pattern vorgeschlagen, das es ermöglicht, Code aus der JSP zu halten, was die Seite testbar hält. Die JSP erstellt dabei den *ObjectFormatter*, der

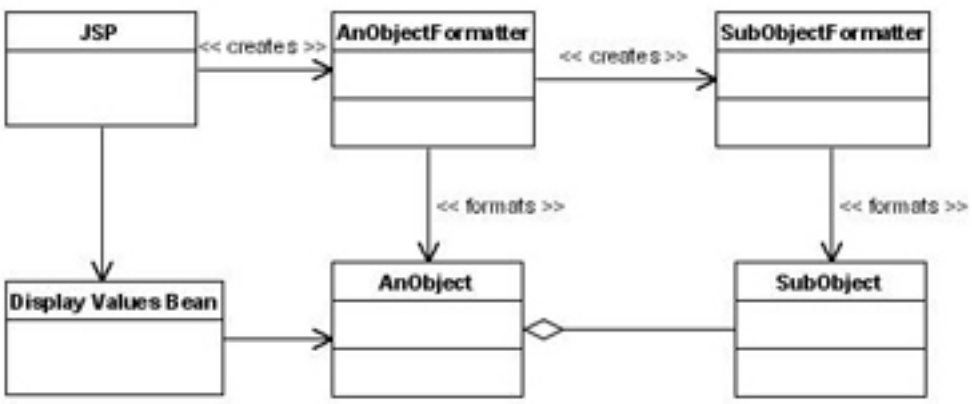



**Abbildung 4.** Landesspezifische Formatierungen

wiederum landestypische *SubObjectFormatter* generiert. Diese formatieren das gewünschte Objekt und liefert es an den ObjectFormatter zurück. Struts bietet für unterschiedliche Zielsprachen eine zusätzliche Bean an, die automatisch generiert wird:

```
<bean:message key="title"/>
```

Dieses Tag sucht in der Message Bean nach einem Key/Value Paar mit dem Key „title“.

## 4 Weiterführendes

Für e-Commerce Anwendungen und insbesondere Internet Shops exisitieren noch eine ganze Reihe anderer Patterns, um den Benutzer so lange wie möglich auf der Website zu halten und ihn zum Kaufen zu animieren, in [7, 6, 9] werden diese näher erläutert. Zudem ist es in Webanwendungen mit Bezahlvorgängen wichtig, auf Sicherheitsaspekte zu achten. Ein „stehlen“ von Sitzungsdaten muß ausgeschlossen werden. Hierfür ist es nötig Sitzungsdaten und insbesondere Session IDs zu verschlüsseln. Dies ist in der Regel sowohl Client- als auch serverseitig zu tun.

In [10] wird besonders auf mobile Webanwendungen eingegangen. Hierbei sind unterschiedliche Views zu gestalten. Eine Erkennung von unterschiedlichen Endgeräten wird hier erläutert

Unter [16] sind eine ganze Reihe von Patterns gelistet, die sich hauptsächlich auf das Layout beschränken. Es werden optimale Positionen von Eingabemasken etc. besprochen.

In [11] wird besonders auf Patterns für die J2EE eingegangen. Es sind dort viele praktische Implementierungsbeispiele zu finden.

## Literatur


1. Radi, H.: Creating Multi-Tier Web Applications with PHP, Paper, 2002
2. Gaedke, M.: Wiederverwendung von Komponenten in Web-Anwendungen, Paper, 1999
3. Hettel, J.: „Best-Practice“-Architekturen für Web-Anwendungen,Paper, Hänchen u Partner Beratungsgesellschaft für Wirtschaftsinformatik Böblingen
4. Richardson, C.: A Pattern Language for J2EE Web Component Development,Buch, 2001
5. Weiss, M.: Patterns for Web Applications, Paper, 2003, Carleton University, Ottawa, Canada
6. Rossi, G et al.: Patterns for E-commerce applications, Paper, Departamento de Informática, PUC-Rio, Brazil
7. Fernandez et al.: Patterns for Internet shops, Paper, Florida Atlantic University, Boca Raton, FL 33431
8. Bonura et al.: Paterns for Web Applications, Paper, 2002, Department of Mathematics and Computer Science University of Camerino, Italy
9. Koch et al.: Paterns for adaptive Web Applications, Paper, Institut für Informatik, Ludwig-Maximilians-Universität München and F.A.S.T. Applied Software Technology GmbH, München, Germany
10. Dabkowski et al.: Model-View-Controller Design Pattern for Mobile and Desktop-based Applications, Paper 2003, European University Viadrina
11. Crawford, W.: J2EE design patterns. - 1. ed., Buch, 2003
12. Alexander, C. et al.: A Pattern Language, New York 1977,Oxford University Press,
13. Apache Jakarta Sruts Projekt, Web Application Framework, Website: http://jakarta.apache.org/struts/
14. WebComposition Markup Language: Website: http://www.teco.edu/ gaedke/webe
15. Java 2 Enterprise Edition: Website: http://java.sun.com/j2ee/
16. Welie, M.: Interaction Design Patterns, Website: http://www.welie.com/patterns/
17. Janssen, C.: Fraunhofer-Institut für Arbeitswissenschaft und Organisation, Stuttgart, 1993
18. Einführung in OCL, Website: http://www.informatik.uni-bonn.de/III/lehre/AG/IntelligenteDatenbanken/ Projektgruppe/SS03/Seminar/Song-OCL.pdf