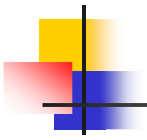
Software Design Patterns

Model View Controller (MVC)

Quoc Nghi Duong
qd01@hdm-stuttgart.de
Taufan Zimmer
tz03@hdm-stuttgart.de



Software Design Patterns - MVC

Agenda

- Was ist MVC ?
- Warum MVC
- MVC Architektur
- MVC Implementierungsbeispiel
- Links

•Was ist MVC genau
- •Geschtlichen Hintergrund aufzeigen
- •Die Ziele nennen, die MVC verfolgt
- •Kurz die einzelnen Komponenten anreißen und welche Aufgaben diese Komponenten besitzen...

•Warum MVC
- •Anhand eines weniger guten JSP Codebeispiel die Vorteile von MVC aufzeigen
- • und warum MVC dadurch Sinnvoll ist besonders in größeren Projekten

•Zur Halbzeit etwa, möchten wir detaillierter auf die MVC Architektur eingehen.
- •Und Anhand von JSP und dem Struts Framework das Zusammenspiel der einzelnen Komponenten erläutern
- •MVC Pattern kann auf große n-tier Web Applikationen betrachtet werden aber auch auf einen kleinen Scrollbalken wie unter Swing
- •Wir haben uns für die Darstellung der Architektur für JSP und Struts entschieden weil es für uns an diesen Beispielen und Diagrammen am leichtesten verständlich war
- •Vorweg möchte ich erwähnen, dass In früheren JSP Spezifikationen zwei philosophisch unterschiedliche Auffassungen befürwortet wurden um Applikationen zu bauen.
- •Diese Auffassungen bezeichnen die JSP Model 1 und Model 2 Architektur. Sie unterscheiden sich durch den Ort wo die Request-Verarbeitung ausgeführt wird.
- •Diese Dinge werden wir dort ansprechen

•Beim vorletzten Punkt möchten wir auf ein Struts Codebeispiel eingehen.

•und dann abschließend einige Links im anführen

Was ist MVC ?

Was ist MVC ?

The mantra of every experienced web application developer:

"Thou shalt separate business logic from display."

Was ist MVC ?

MVC Geschichte

- Wurde von Xerox in den 80ern für Benutzeroberflächen in Smalltalk entwickelt.
- Die Idee ist die Trennung des Programms in die drei Einheiten: Geschäftslogik (Model), Präsentation (View) und Interaktion (Controller).
- Erste Veröffentlichung erschienen über MVC: "A Cookbook for Using the Model-View-Controller User Interface Paradigm in Smalltalk-80" (Glenn Krasner, Stephen Pope) im *Journal of Object-Oriented Programming (JOOP)* August/September 1988.
- Andere Namen für MVC:
  - Presentation-Abstraction-Control (PAC)
  - Interface-Control-Model (ICM)

"A Cookbook for Using the Model-View-Controller User Interface Paradigm in Smalltalk -80", by Glenn Krasner and Stephen Pope, was published in the August/September 1988 JOOP. It defined MVC as follows:

"Model-View-Controller (MVC) programming is the application of this three-way factoring whereby objects of different classes take over the operations related to the application domain (the model), the display of the application's state (the view), and the user interaction with the model and the view (The controller)."

"Models -- The model of an application is the domain-specific software simulation or implementation of the application's central structure."

"Views -- In this metaphor, views deal with everything graphical: they request data from their model and display the data."

"Controllers -- Controllers contain the interface between their associated models and views and the input devices (e.g., keyboard, pointing device, time)."

PAC

Presentation-Abstraction-Control, which maps (roughly) to the notions of View/Controller, Model, and Mediator, first proposed in a paper by Joelle Coutaz 1987 and published as a pattern in „Pattern-Oriented Software Architecture - A System Of Patterns" by FrankBuschmann, RegineMeunier, HansRohnert, PeterSommerlad, and MichaelStal (The Siemens Gang), 1995.

Ivar Jacobson, wrote about a similar, but not identical architecture to MVC in his book "Object Oriented Software Engineering: A Use-case driven approach" in 1992.

ICM

Interface-Control-Model

In 1992 Greg Hendley and Eric Smith wrote a series of articles in the Smalltalk Report describing what they referred to as the "Interface-Control-Model (ICM)" architecture.

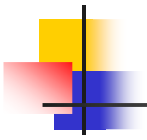
The ICM somewhat similar to Jacobson's concept and nearly identical to the PAC architecture but not as.

Links:

http://c2.com/cgi/wiki?ModelViewControllerHistory

http://c2.com/cgi/wiki?WhatsaControllerAnyway

http://c2.com/cgi/wiki?ModelModelViewController



Was ist MVC ?

Ziel

- flexibles Programmdesign um u.a. eine spätere Änderung oder Erweiterung einfach zu halten.
- sorgt für eine gewisse Übersicht und Ordnung in großen Anwendungen
- Rollenverteilung, z.B.:
  - der Web-Designer erstellt das Erscheinungsbild,
  - der Programmierer erstellt die nötige Geschäftslogik

Die Idee des Modells ist die Trennung des Programms in die drei Einheiten Geschäftslogik, Präsentation und Interaktion.

Als eindeutige Bezeichnung der Einheiten werden Model, View und Control verwendet.

Ziel des Modells ist ein flexibles Programmdesign um u.a. eine spätere Änderung oder Erweiterung einfach zu halten.

Außerdem sorgt das Modell für eine gewisse Übersicht und Ordnung in großen Anwendungen.

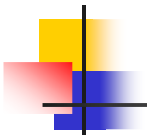
Gleichzeitig bringt die Trennung auch eine Rollenverteilung mit sich.

Fachkräfte können so optimal, ihrer Fähigkeit entsprechend, eingesetzt werden:

Der Web-Designer erstellt das Erscheinungsbild,

der Programmierer erstellt die nötige Geschäftslogik,

Datenbankexperten kümmert sich um die optimale Datenverwaltung usw.



Was ist MVC ?

Model

- Das Datenmodell
- enthält die dauerhaften (persistenten) Daten der Anwendung.
- hat Zugriff auf diverse Backend-Speicher wie zum Beispiel Datenbanken.
- realisiert die eigentliche Geschäftsintelligenz.

Das Model

enthält die dauerhaften (persistenten) Daten der Anwendung.

Das Model hat in der Regel Zugriff auf diverse Backend-Speicher wie zum Beispiel Datenbanken.

Das Model realisiert die eigentliche Geschäftsintelligenz.

Das Model sollte die View nicht kennen, d.h. es weiß nicht auf welche Art und Weise diese Daten letztendlich dargestellt werden

•Das Model kennt die Daten, die angezeigt werden können

•Das Model kennt die Operation, die auf die Daten ausgeführt werden können

•Das Model weiß aber nicht auf welche Art und Weise diese Daten dargestellt werden

Was ist MVC ?

View

- Die Darstellungsschicht
- präsentiert die Daten
- Die Programmlogik sollte aus der View entfernt werden und enthält auch keine Businesslogic

•Die View soll eine reine Darstellungsschicht sein

•Sie soll im Grunde genommen keine Intelligenz beisitzen und nur Daten präsentieren

•Die View soll keine Ablauflogik oder irgendeine Businesslogik enthalten.

•Vorzugsweise sollte die View simple gestaltet sein, d.h. keine Vermischung von unterschiedlichen Programmiersprachen

•Dazu jedoch später mehr

Was ist MVC ?

Controller

- Die Steuerungsschicht.
- Nimmt requests / events an
- steuert die Ablauflogik
- Entscheidet welche View aufgerufen wird

•Der Contoller stellt die Steuerungsschicht dar

•Der Controller ist dafür verantwortlich die request bzw. events anzunehmen

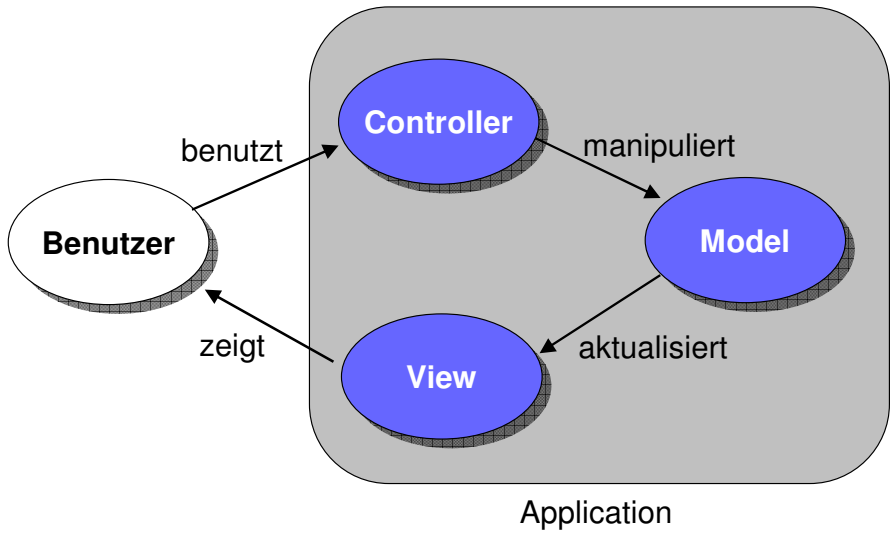
- •durch Mouseklicks an
- •durch Keyboard eingaben
- •Oder Webrequest
- •Die Eingaben werden vom Controller so ausgewertet und transformiert, dass es vom Model verstanden werden können

•Der Controller steuert die Ablauflogik in einer Applikation.

- •D.h Wenn ein spezielles Ereignis eingetreten ist, dann ruft er die dafür verantwortliche View auf

Was ist MVC ?

MVC Interaktionsdiagramm



•Hat die View Interesse an Änderungsvorgängen im Model kann es sein Interesse beim Model bekunden und sich dort registrieren

•Bei Zustandsänderungen werden über ein Callback alle Views benachrichtigt.

•Falls kein Interesse besteht kann die View sich auch wieder deregistrieren.

Beziehung zwischen Views und Models: **Observer Pattern**

Controller sind **Strategie** von Views

Views formen einen Baum mit **Composite Pattern**

Views beinhalten meist **Decorators** um zusätzliche visuelle Eigenschaften hinzuzufügen.

Das Model ist oft ein **Mediator**

Model-View-Controller is made up of a number of smaller design patterns. The views form a tree using the Composite Pattern. The relationship between views and models is the Observer Pattern. The controllers are strategies of the views. Document View is more popular now than true Model View Controller, and it consists of Composite and Observer, but not Strategy.

Although Composite Pattern, Observer Pattern and Strategy Pattern are the core of Model View Controller, it uses more patterns. The model is often a Mediator (The Smalltalk Browser is a good example). The tree of views often contains Decorators to add properties to views. A view often uses an Adaptor to convert the model to a standard interface that the view can use. Views create controllers using a Factory Method.

Model View Controller is hard to learn because it is complex. It is much easier to understand if you learn the patterns one at a time and then fit them together.

Links:

http://c2.com/cgi/wiki?ModelViewControllerAsAnAggregateDesignPattern

Warum MVC

```
<%@ page import="java.sql.*" %>
<HTML>
<HEAD><TITLE>Example without MVC</TITLE></HEAD>
<script language="Javascript" src="script.js" />
<BODY BGCOLOR="#FFFFFF">
<CENTER>
<% Connection conn = null;
   try {
        Class.forName("oracle.jdbc.driver.OracleDriver");
        conn = DriverManager.getConnection("jdbc:oracle:thin:@db.hdm.de:1521:fuju","myName","myPWD");
        Statement stmt = conn.createStatement();
        ResultSet rs = stmt.executeQuery("SELECT * FROM hdm.Employees");
%> <TABLE border=1> <%
        //Loop through results of query.
        while(rs.next()) {
        %> <TR>
             <TD class="even"> <%=rs.getString("vorname")%> </TD>
             <TD class="uneven"> <%=rs.getString("nachname")%> </TD>
             <TD class="even"> <%
                if(rs.getString("status").equals("silver")) {
                %> <a href="silverMember.html" onclick="javascript:member('silver')">Member</a> <%
                } else {
                %> <a href="goldMember.html" onclick="javascript:member('gold')">Member</a> <%
                } %>
           </TD>
         </TR> <%
        } %>
    </TABLE> <%
    } catch(SQLException e) {
    %> <bold>SQLException:</bold> <%=e.getMessage()%> <BR> <%
    } %>
</CENTER>
</BODY>
</HTML>
```


•An diesem Codebeispiel können die Nachteile von IN -- Anführungszeichen -- herkömmlichen Programmen ersichtlich werden

•Es ist zuerst auffällig

- •HTML Tags
- •Scriptlets
- •JSP Code
- •Javascript
- •Vermischt sind, das dient nicht gerade zur guten Übersicht.

•Weiter ist zu sehen, das direkt aus der JSP heraus eine Datenbankverbindung hergestellt wird

- •Wenn das in vielen JSP Seiten einer Webapplikation geschieht und sich das Backend ändern sollte, dann müssen überall dort wo Datenbankverbindungen hergestellt werden, Änderungen durchgeführt werden
- •Hier stehen die SQL Statements Hardcodiert
- •Die Codierung des Workflows ist in den Seiten eingebaut. Wenn ein neuer Platinmember hinzukommt müssen möglicherweise an vielen Anderen JSP Seiten angepasst werden
- •Und dann wird hier auch noch die Fehlermeldung dem Benutzer bekannt gegeben.
- •Eine willkommene Einladung für einen Hacker.

•Das Problem hier sicherlich auch, dass viele Technologien wie beispielsweise JSPs sehr Mächtig sind um Sie nur als View einzusetzten.

•Der Programmier muss gewissenhaft mit den Sprachwerkzeugen umgehen die ihm durch die die Technologie gegeben ist

•Aber es ist eben doch mal schnell gewisse Logik eingebaut.

•Das wird beispielsweise beim Strutsframework durch die Einführung von Strutseigenen Taglibraries versucht zu unterbinden.

Warum MVC

Nachteile

- HTML und Java sind eng miteinander gekoppelt
- Verwendung von Java-Script
- Die Codierung des Workflows ist in den Seiten eingebaut
- Enge Kopplung der Geschäftslogik
- Schwer zu debuggen

HTML und Java sind eng miteinander gekoppelt

-Der Entwickler einer JSP Seite muss trotz Vereinfachung immer noch Java und Graphik-Design können (v. a. bei nicht-trivialen Tag-Libraries)

-Der resultierende Code ergibt entweder schlechtes Java oder schlechte Seite Seiten oder auch beides

Verwendung von Java-Script

-Zur ersten, Client-Seitigen Feld-Prüfung

-Erhöht die Komplexität des Codes und macht diesen schlecht wartbar („Nun haben wir schon 3 Sprachen in einem Quell-File“)

-Debuggung wird eminent schwierig – Fehlersuche wird teuer!

Die Codierung des Worklows ist in den Seiten eingebaut

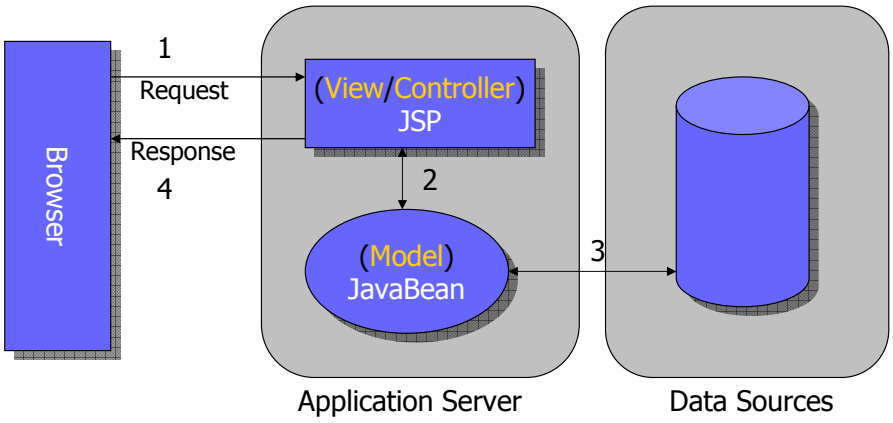
-Eine Umstellung ist schwierig (Spaghetti Code), Verständnis problematisch

Enge Kopplung der Geschäftslogik

-Daten mit dem GUI provoziert gegenseitige Änderung von GUI & Logik bei einer Änderung des jeweiligen Gegenparts. Im schlimmsten Fall betrifft dies eine Vielzahl von Seiten!

MVC Architektur

Model 1 Implementierung am Beispiel von JSP



Aus diesen ganzen Nachteilen die in der vorigen Folie genannt wurden, haben sich dann schlaue Köpfe letztendlich die Trennung von Model und View ausgedacht.

Hier an diesem Diagram, am Beispiel von JSP.

Hier sieht man eine saubere Trennung zwischen Model und der letztendlichen Darstellungs-Ebene die mit JSP realisiert wird.

Das Model sind hier JavaBeans die die gesamte Bussinesslogik beinhalten und u.a. mit der Datenbank kommunizieren. Sie brauchen sich überhaupt nicht kümmern wie ihre Daten dann letztendlich präsentiert werden.

Die View kümmert sich mit Hilfe des Controllers wie die Daten dargestellt werden.

In der JSP ist es oft so, oder der Programmiere wird oft dazu „verleitet“, dass die ganzen Controllerfunktionalitäten gleich in der JSP mit eingabaut werden.

Im Model 1 nimmt JSP die Funktionalität von View und Controller an.

Hier kann man also immer noch die Nachteile einer mächtigen Templatesprache, wie JSP, sehen (Terrence Parr)

Über den Controller gab es schon etliche Diskussionen, was jetzt Controller genau ist, für was Controlller genau zuständig sind usw...

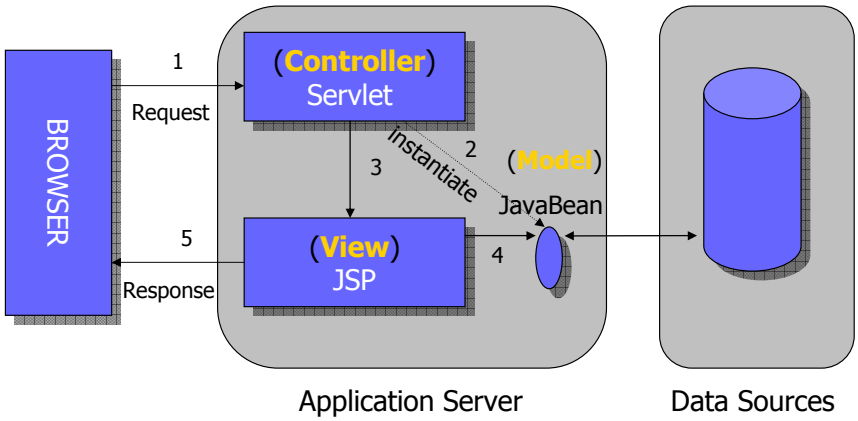
Swing vereinfacht MVC zur Model-UI-Delegate-Architektur (Model 1)

In komplexeren Komponenten ist die MVC Trennung jedoch sichtbar (z.B. Tabellen und Bäumen)

Die Beziehung zwischen Model und View wird durch das Observer Pattern implementiert

MVC Architektur

Model 2 am Beispiel von JSP



Im Laufe der Zeit hat man dann weitere Erfahrungen gesammelt...
Der Benutzer setzt einen Request ab und der Request wird nun vom einem Servlet abgefangen.

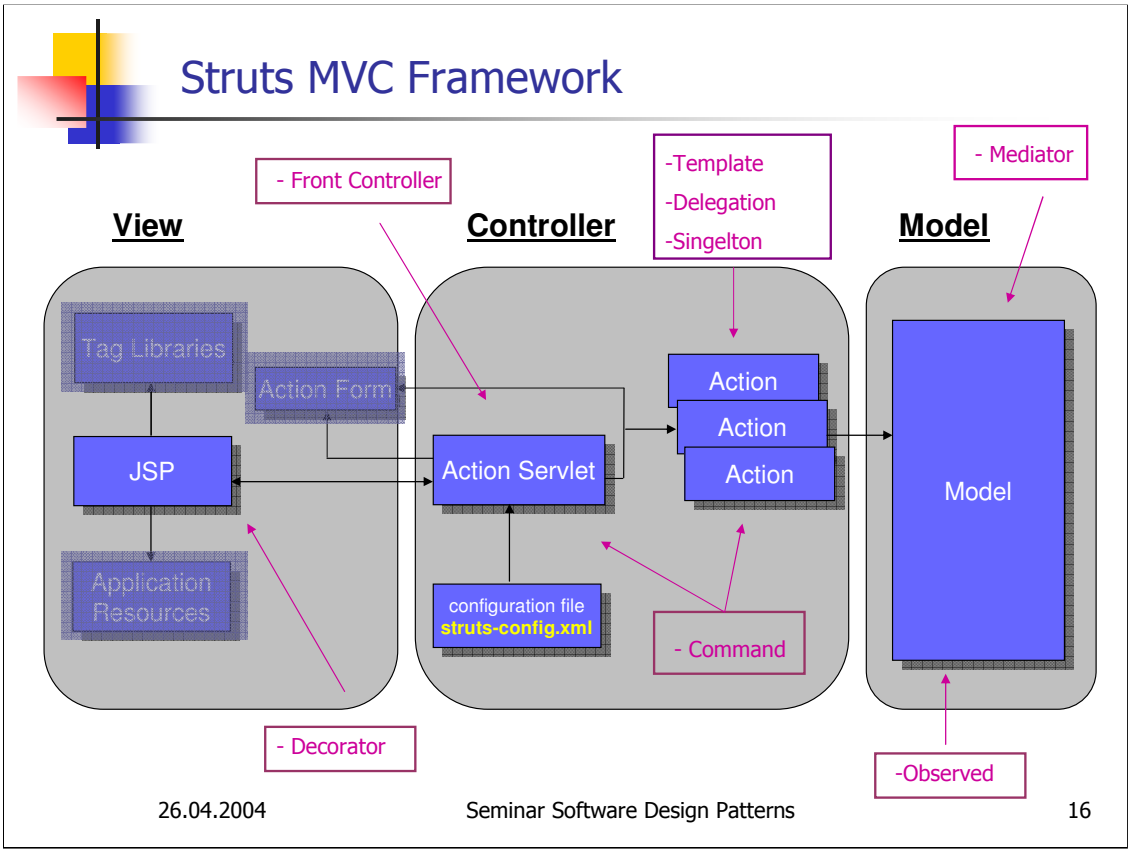
- Es hat sich als Vorteil erwiesen das der Controller als Servlet abgespalten wird.
  - Serverseitig wird ein Servlet von einem Servlet-Container wie Tomcat schneller vearbeitet
  - Das Servlet übernimmt die Verarbeitungsintensiven Aufgaben.
  - JSPs würden im Grunde genommen intern wieder in Servlets umgewandelt werden. Diesen Schritt möchte man sich ersparen
  - Workflowlogic kann zentralisiert werden und ist nicht auf mehrere JSPs verstreut

- Das Model kennt die Daten, die angezeigt werden können
  - Das Model kennt die Operation, die auf die Daten ausgeführt werden können
  - Das Model weiß aber nicht auf welche Art und Weise diese Daten dargestellt werden
  - Das Model kann jetzt auf Datenressourcen zurückgreifen und gibt nach der Verarbeitung der Daten die Kontrolle zurück an den Controller

- Der Controller entscheidet nun welche View aufgerufen werden soll
  - Und forwardet den Benutzer auf die bestimmte Seite.
  - Die View hat die Möglichkeit lesend Daten von der JavaBean abzugreifen

- Diese saubere Trennung von Model View und Controller führt wiederrum zu einer klaren Skizierung der Rollen und
  - Den Verantwortlichkeiten zwischen dem Programmierer und dem Page Designer
  - Letztenendes kann gesagt werden, das Je komplexer die Applikation ist desto mehr profitiert man von der Model 2 Architektur

(Nicken)



•So, auf diesem Diagramm ist das Strutsframework zu sehen. Es basiert auf dem Model 2 MVC Pattern.

•Wie man sieht ist das Struts Framework ist ein Konglomerat von ganz vielen Patterns. Wahrscheinlich verbergen sich noch viel mehr Patterns dahinter.

•Die meisten Patterns sind uns bereits bekannt und wurden schon vorgetragen

•Wenn man all die Patterns miteinander kombiniert, dann sag ich jetzt mal, schimmert das MVC Pattern hindurch

•Ich möchte nicht weiter auf die einzelnen Patterns eingehen. Laßt es einfach kurz auf euch einwirken....

•Das was hier abgedeckt ist, ist für uns nicht wichtig

- •JSP , ActionServlet fein granular , Actios dünne Model Wrapper die einen Request weiter an eine EJB delegieren
- •Model sind EJBs und enthalten Geschäftslogik

•Struts selber kennt laut Doku keine Modelkomponenten. Die müssen extern bezogen werden.

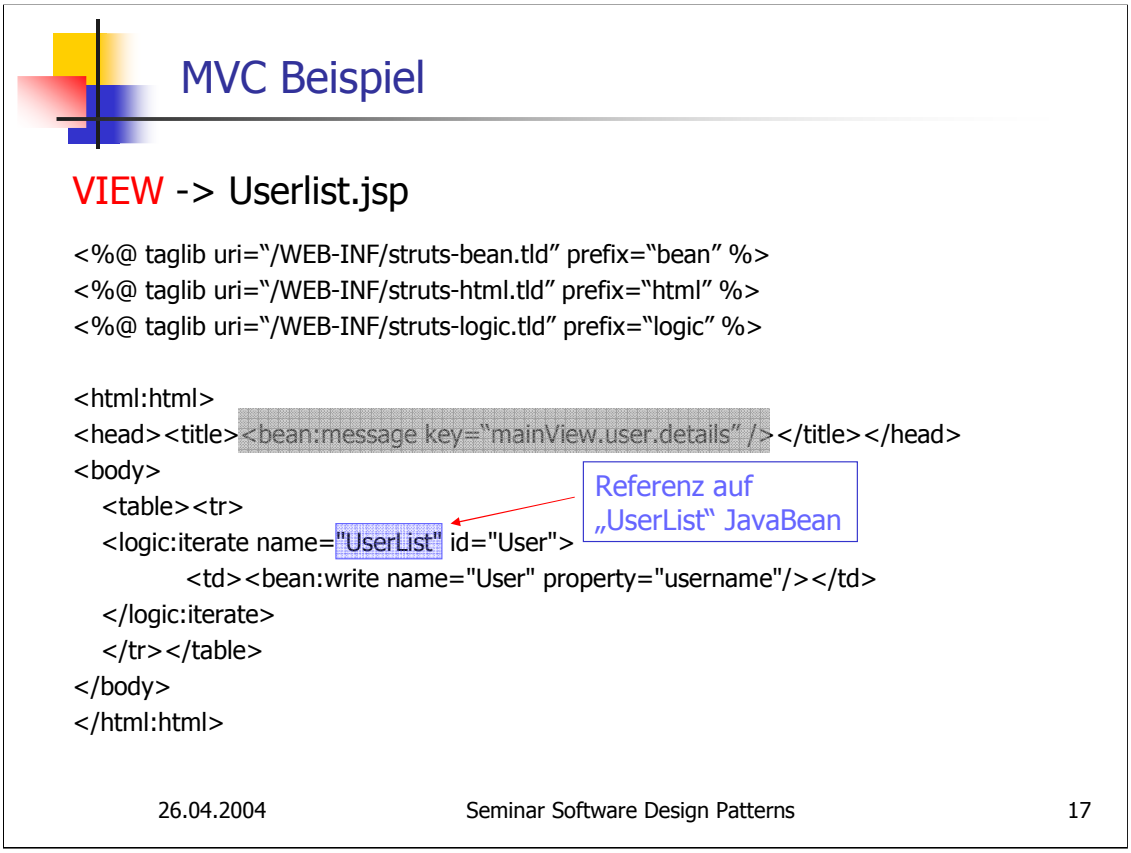
•Das Struts-Framework hat sich zu einem beliebten MVC Framework entwickelt.

•Es ist zwar nicht Perfekt aber kommt dem Wunsch der optimalen Trennung von Model View Und Controller sehr nah.

•Deshalb jetzt auch ein kleines MVC Codespiel anhand von Struts

•------- Warum nicht Perfekt?

- •Viele Inderektionen



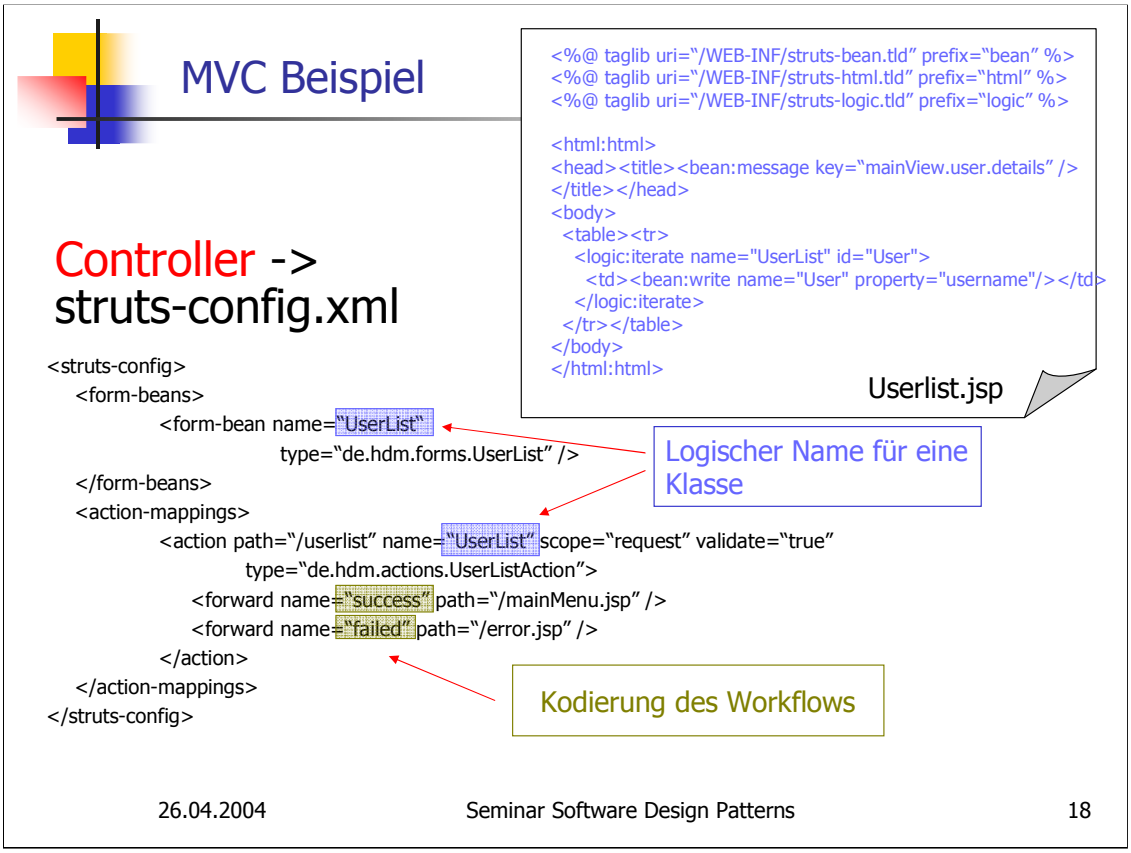
Hier ist ein simples Beispiel wie eine klare Trennung der View zur Busissesslogik (Model) im Struts Framework realisiert wird

Die View soll laut Struts lediglich über Struts-eigene Taglibraries auf JavaBeans zugreifen sprich Daten vom Model auslesen

Keine JSP-Scriptlets

Kein Java-Script für überprüfung von z.b. Eingaben im Textfeld bzw. ob dort etwas eingegeben wird oder Leerstrings,

Einfach um den Code sauber und übersichtlich zu halten und die strikte Trennung zum Model zu gewährleisten.



•OK, hier ist die XML – Basierte Konfigurationsdatei für den Controller von Struts zu sehen

•Das hier ist die Folie die Taufan gerade beschrieben hat

•Und jetzt kann man erkennen, dass UserList im Grunde genommen eine Logische Bezeichnung für den eigentlichen Klassennamen.

•Die Klassen kann in Ihrer Funktion verändert werden, Die Klassen können umbenannt werden, ohne das es der Page Designer merkt

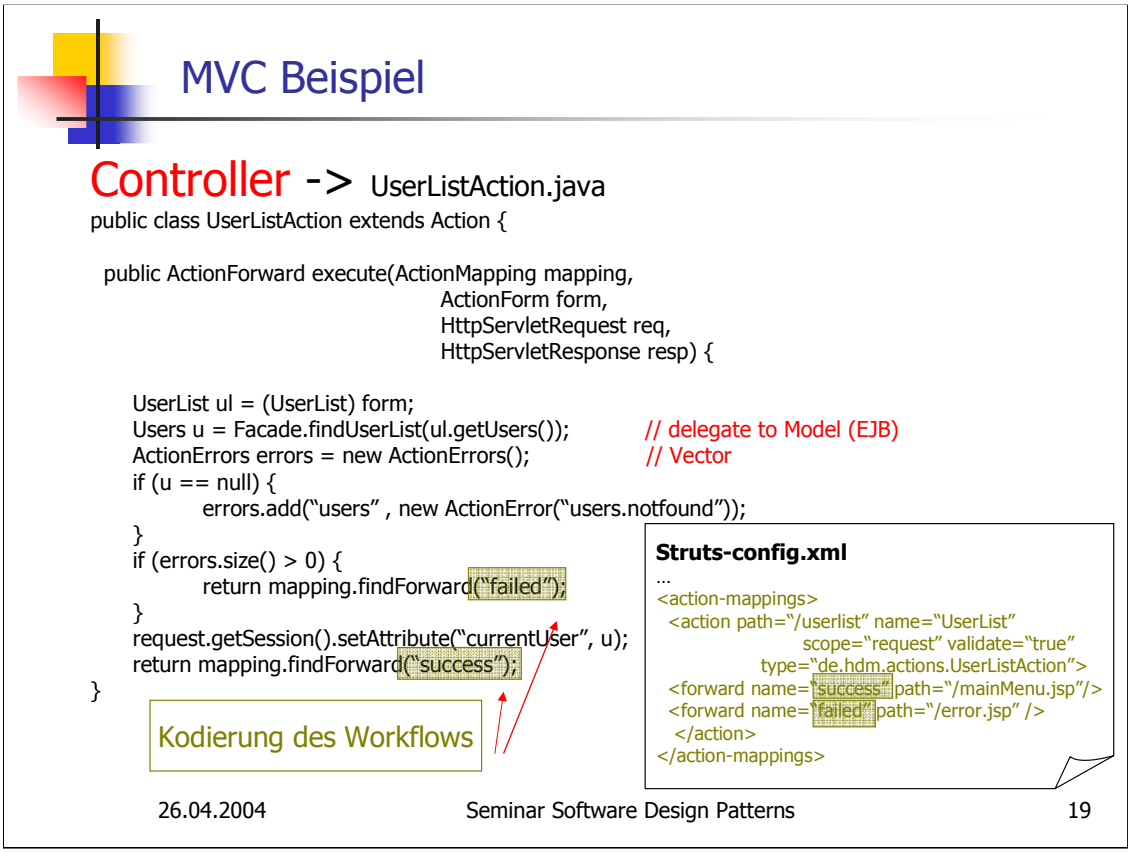
•Der Zugriff auf die UserList ist für den Page Designer wie man so schön sagt völlig transparent.

•Der Bergriff UserList ist irgendwo im Design niedergeschrieben und der Programmier und der JSP Coder halten sich an diese Abmachung

•Hier ist zu sehen, dass der Aufruf der UserListAction Klasse zu zwei forwardings führen kann

- •Nämlich zur mainMenu.jsp wenn die UserListAction Klasse ein success zurückgibt, ICH zeige euch gleich den Code dazu
- •Und zu einer error Seite wenn die Klasse UserListAction failed zurückliefert

•Das bedeutet also, das sich die Kodierung des Workflows hier zentral in der struts-config.xml befindet und nicht auf mehrer JSPs verteilt ist



Auf dieser Folie sehen wir die UserListAction die von Action erbt und ein Teil des Controllers ist

•Ich möchte nicht tiefer auf die Rolle von Actions in dem Strutsframework eingehen da es für uns nicht relevant ist

•Was für uns wichtig ist, ist das hier der request über eine Facade an eine EJB weiter delegiert

• Der Rückgabewert wird in einer Instanz der Klasser Users gespeichert

•ActionErrors ist ein Vektor in dem Fehlermeldungen gespeichert werden können

•Wenn jetzt u gleich null ist, dann wird ein neues Objekt ActionError gebaut ohne S und dem Vektor hinzugefügt...

•Das JSP Seite die aufgerufen werden soll steht nicht Hardcodiert in dem Code.

•Die Klasse kann kompiliert werden und muss nicht mehr angefasst werden.

•Falls Änderungen durchgeführt werden sollen geschieht das hier zentral an dieser Stelle.

•D.h das Model hat jetzt irgendeinen Wert zurückgeliefert, weis aber nicht welche View für die Darstellung verantwortlich ist

•Das weiß nur der Controller, das Model ist somit nicht abhängig von der View und könnte als Komponente für anderer Geschäftsbereiche eingesetzt werden

Links

Links

- Understanding JSP Model 2 architecture
  - http://java.sun.com/blueprints/guidelines/designing_enterprise_applications_2e/web-tier/web-tier5.html
- C2.com / Wiki
  - http://c2.com/cgi/wiki?ModelViewController
- IBM Struts, an open-source MVC Implementation
  - http://www-106.ibm.com/developerworks/java/library/j-struts/
- New way to learn MVC. View a sing-song
  - http://home.in.tum.de/pittenau/ModelViewController.mp3

MVC

Model View Controller

Danke für eure Zeit