Technische Universität Ilmenau
Fakultät für Informatik und Automatisierung
Institut für Praktische Informatik und Medieninformatik
Fachgebiet Telematik
Prof. Dr.-Ing. habil. Dietrich Reschke

Betreuer: Dipl. Inf. Thorsten Strufe

Hauptseminar Informatik
im SS 2002

Thema-Nr. 5

Java Web-Frameworks

vorgelegt von:

Alexander Löser
Max-Planck-Ring 4
98693 Ilmenau
03677/2081632
Alexander.Loeser@wi.stud.tu-ilmenau.de
Matrikel-Nr.: 27183

Inhaltsverzeichnis

Abkürzungsverzeichnis

API Application Program Interface.

DOM Document Object Model.

EJB Enterprise Java Beans.

EL Expression Language.

XML Extensible Markup Language.

XSLT Extensible Stylesheet Language Transformations.

H-MVC Hierarchical-MVC.

HTML Hypertext Markup Language.

HTTP Hypertext Transfer Protocol.

J2EE Java 2 Plattform Enterprise Edition.

JSP Java Server Pages.

MVC Model-View-Controller.

TDK Turbine Developement Kit.

VS Value Stack.

1 Einleitung

Oft wird die Wiederverwendung von Software als erstrebenswertes Ziel und erfolgversprechendes Instrument bei der Softwareentwicklung herausgestellt. Frameworks verfolgen genau diesen Aspekt, denn sie bestehen *„aus einer Menge von zusammen arbeitenden Klassen, die einen wiederverwendbaren Entwurf für eine bestimmte Klasse von Software darstellen“*[1] Neben der einfachen Wiederverwendung von Code dienen Frameworks also auch der Wiederverwendung von Design und sogar kompletter Services. Sie stellen branchenspezifische oder -übergreifende Lösungen für allgemein bekannte Problem dar. Dabei beschränken sie sich nicht auf reine Umsetzungsempfehlungen, wie sie Design-Patterns typischerweise zum Inhalt haben, sondern liefern fertige Implementierungen für ein bestimmtes Anwendungsgebiet.
Frameworks bestehen aus fertigen und halbfertigen Teilsystemen. Zum einen wird also die grundlegende Architektur für ein Softwaresystem bestimmt, auf der anderen Seite aber auch genug Raum für individuelle Erweiterungen gelassen.
Diese Arbeit beschäftigt sich mit Web-Frameworks. Darunter sollen Frameworks verstanden werden, die eine Architektur für Client-Server Anwendungen liefern, der client-seitige Zugriff erfolgt dabei durch einen Webbrowser. Besondere Bedeutung hat das Entwurfsmuster des Model-View-Controller erlangt, weshalb es in einführender Weise kurz vorgestellt werden soll. Im weiteren wird ein kurzer Überblick über die verfügbaren Frameworks gegeben, sowie die Auswahl der hier vorgestellten Frameworks begründet. Nach der einzelnen Beschreibung der Frameworks soll in einem abschließenden Fazit nochmals deren spezifische Eignung herausgestellt werden.

[1] [Gamma u. a. 1996, S. 37]

2 Das Model-View-Controller-Pattern

Um die Auswahl und den Nutzen der vorgestellten Frameworks besser verstehen zu können, ist es nötig einen Blick auf die zugrunde liegenden Design-Patterns zu werfen. In diesem Kapitel sollen jedoch nicht die klassischen von der *Gang of Four*[2] beschriebenen Design-Patterns vorgestellt werden, vielmehr soll das komplexere Konzept des Model-View-Controller (MVC) behandelt werden. Es ist ein häufig diskutiertes Entwurfsmuster, welches bei der Architektur von graphischen Benutzeroberflächen sowie von webgestützten Anwendungen besondere Bedeutung gewonnen hat.

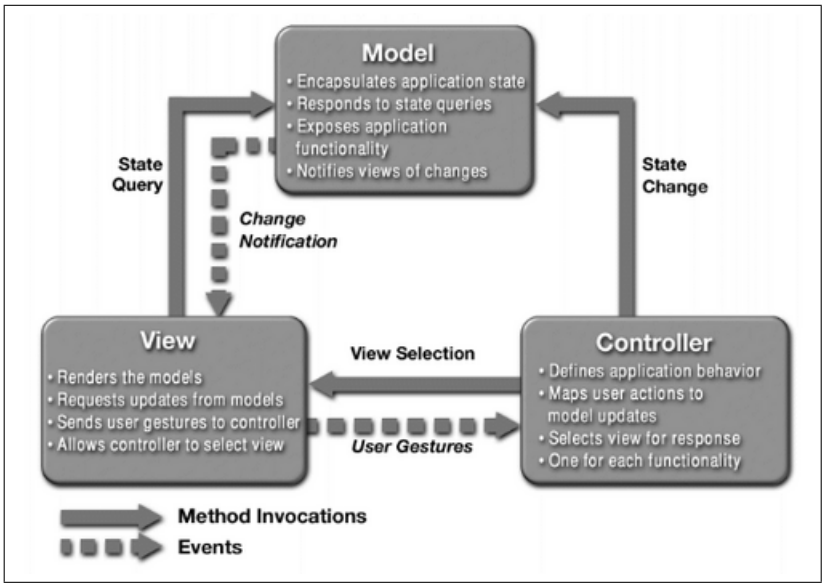

Abbildung 2-1: Model-View-Controller Architecture Vgl. [Sun 2001]

Das MVC-Muster hat seinen Ursprung in Smalltalk 80 und diente der Entwicklung von Benutzerschnittstellen.[3] Dabei zeichnet es sich vor allem durch die Trennung von Präsentation(View), Ablaufsteuerung(Controller) und Datenbereitstellung(Model) aus. Wichtig ist, dass ein Model keine Kenntnis von den View- und Controller-Komponenten hat und die Bestandteile somit separat entwickelt werden können. Sun Microsystems Inc. hat das MVC-Muster in die Welt der webbasierten Anwendungen übertragen. So wird es im Rahmen der Blueprints für die Entwicklung mit der Java 2 Plattform Enterprise Edition (J2EE) empfohlen.[4] Neben einem guten

[2] Vgl. [Gamma u. a. 1996]
[3] Vgl. [Yin 1995]
[4] Vgl. [Sun 2001]

Programmierstil profitiert man vor allem davon unterschiedlichste Clients mit den selben Daten versorgen zu können.
Im Allgemeinen stellen Java Beans den Model-Part dar – sie speichern die Daten und den Zustand der Anwendung. Bei der Entwicklung mit der J2EE wird dies entsprechend durch Entity-Beans und Session-Beans realisiert. Für die View empfiehlt Sun den Einsatz von Java Server Pages (JSP). Die anzuzeigenden Daten werden aus dem Model geholt und an Stelle von speziellen Tags in die Hypertext Markup Language (HTML)-Seite eingefügt. Diese Vorgehensweise wird auch als Pull-MVC bezeichnet. Bei der gegensätzlichen Push-Strategie kümmert sich das Model um die Aktualität der View. Die dritte Komponente bei MVC-Ansätzen – der Controller – wird meist durch ein Servlet repräsentiert. Es reagiert auf Nutzerinteraktionen mit der View, ruft entsprechende Methoden des Models auf und setzt eine neue View. Trennt man wie beschrieben die Einheiten View und Controller, so spricht man auch von einer Model-2-MVC. Weniger konsequent war die Model-1-MVC-Architektur bei der die beiden Einheiten anfänglich in einer JSP-Datei implementiert wurden.[5]
Bei großen Applikationen kann das Controller-Servlet schnell unübersichtliche und nicht wartbare Ausmaße annehmen. Deswegen nimmt man eine komponentenähnliche Unterteilung vor, die sich als baumähnliche Verknüpfung von Controller-Komponenten darstellt. Jeder Controller hat zwar seine eigenenen Views und Models, reicht aber nicht bekannte Ereignisse an die übergeordnete Ebene weiter. Diese Erweiterung wird auch als Hierarchical-MVC (H-MVC) bezeichnet.[6]

[5] Vgl. [Duffey 2000]
[6] Vgl. [Cai u. a. 2000]

3 Web-Frameworks

3.1 Überblick und Auswahl der Web-Frameworks

Es existiert eine Vielzahl an Frameworks zur Entwicklung von Web-Applikationen. Sie setzt neben dem MVC-Muster verschiedenste weitere Entwurfsmuster um und bieten mehr oder weniger Unterstützung für die Implementierung. Eine kleine Auswahl soll im Rahmen dieser Arbeit vorgestellt werden und in Bezug auf ihre Eignung verglichen werden. Im folgenden sollen die grundlegenden Richtlinien für diese Auswahl dargelegt werden.

Die hier betrachteten Frameworks genügen alle dem Anspruch domänenunabhängig zu sein – mit anderen Worten sie stellen Basis-Frameworks dar. Daneben ist die freie Verfügbarkeit und Nutzung ein wesentlicher Aspekt für die Betrachtung. Im Hinblick auf die universitäre Nutzung beschränken sich die ausgewählten Projekte somit auf Open-Source-Ansätze. Dabei ist zu beachten, dass sich die Projekte in einem Status produktiver Nutzbarkeit befinden sollten. Die erst in der Spezifikation befindlichen JavaServer-Faces[7] wurden demzufolge ebenso ausgeschlossen, wie zahlreiche Sourceforge-Projekte im Alpha-Stadium. Insgesamt soll die Auswahl einen Querschnitt durch die verschiedenen MVC-Architekturen darstellen und auf verschiedenste Möglichkeiten der Realisierung eingehen.

Bei der Fokussierung auf Open-Source-Ansätze wird man um einen Blick auf das Jakarta-Projekt der Apache Software Foundation nicht herum kommen. Dort sind die ersten beiden Frameworks *Struts* und *Turbine* beheimatet. Sie wurden unabhängig voneinander entwickelt und verfolgen unterschiedliche Ziele und Strategien. Während Struts sich auf ein solides MVC-Architektur beschränkt, stellt Turbine darüber hinaus zahlreiche weitere Services zur Verfügung. Gleichzeitig ist mit den beiden Frameworks eine JSP-basierte und eine template-basierte Lösung in der Auswahl vertreten. Einen gegensätzlichen aber dennoch interessanten Weg verfolgt *Barracuda*, weshalb es hier auch keinesfalls fehlen darf. Mit *Expresso* soll außerdem ein Turbine-ähnliches Framework vorgestellt werden, welches eine relativ hohe Verbreitung erlangt hat. Schließlich reiht sich das hierarchische MVC-Framework *Webwork* in die Auswahl ein. Es hat noch eine relativ geringe Verbreitung gefunden, besitzt aber dennoch ein recht großes Entwicklungspotential.

[7] Vgl. [Sun 2002]

3.2 Apache Struts

Eine strikte Umsetzung des Model-2-MVC ist das Open-Source-Framework Struts aus dem bekannten Jakarta-Projekt.[8] Die klassische Rollenverteilung auf JSPs, Servlets und Beans wurde beibehalten. Aus den genannten Komplexitäts-

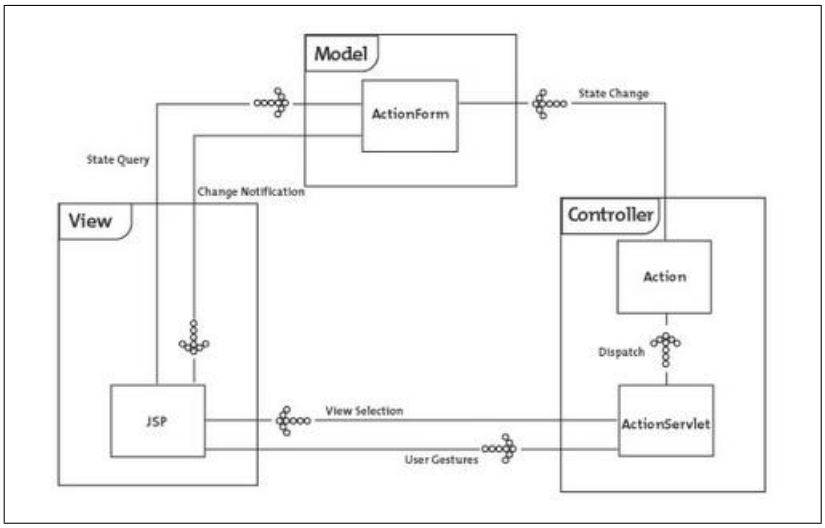

Abbildung 3-2: Struts MVC Implementation Vgl. [Sweeting u. a. 2002]

Gründen wurde der Controller ein wenig erweitert. Als zentrales Request-Ziel kommt ein so genanntes ActionServlet (*org.apache.struts.action.ActionServlet*) zum Einsatz. Von ihm werden ankommende Requests über ein Action-Mapping (*org.apache.struts.action.ActionMapping*) zu einer bestimmten Action (*org.apache.struts.action.Action*) weitergeleitet. Dieses Mapping wird flexibel in einer XML-Datei(*struts-config.xml*) konfiguriert. Ebenso wird dort die weitere Ablaufsteuerung festgelegt, indem je nach Erfolg einer Action auf verschiedene Views verwiesen werden kann.
Die Action-Klasse sollte nur in einfachsten Fällen Geschäftslogik enthalten. Für eine bessere Trennung ist deren Realisierung in separaten Business-Komponenten vorzunehmen. Rückgabewert einer Action ist ein *ActionForward*-Objekt, aus dem die nächste View bestimmt wird. Vor der Rückgabe sollten jedoch alle relevanten Daten im Model gespeichert werden, welches durch ein Session-spezifisches *ActionForm* repräsentiert wird. Es enthält lediglich getter- und setter-Methoden, sowie eine Methode zur Syntax-Prüfung (*validate()*) und eine *reset()* Methode zur Belegung mit Standardwerten.

[8] Vgl. [Apache 2002b]

In der View werden einfache JSPs verwendet gleichzeitig aber jedoch versucht sämtlichen Java-Code fern zu halten. Dazu kommen eigene Tag-Libraries zum Einsatz, deren einfache Erlernbarkeit es Web-Designern ermöglicht ohne Java-Kenntnisse mit den Programmierern zusammenzuarbeiten. Durch diese Tags werden die Views mit den Daten des Models angereichert und ermöglichen eine transparente Internationalisierung. Zum einen gibt es *struts-html.tld*, welche einen Großteil der normalen HTML-Tags nachbildet, dabei aber spezifische Einstellungsmöglichkeiten nachrüstet, beispielsweise die Bindung von Formularen an Actions. Durch *struts-bean.tld* wird vor allem Zugriff auf globale Messages, den Context oder Struts-spezifische Einstellungen gewährleistet. Die letzte Tag-Library *struts-logic.tld* dient der darstellungsbezogenen Programmlogik auf die auch in der View meist nicht verzichtet werden kann.[9]
Insgesamt wurde mit Struts ein MVC-Framework geschaffen, das die Komponenten-Trennung konsequent verwirklicht und durch den Einsatz von Tag-Libraries auch deren Umsetzung klar voneinander trennt. Allerdings wird man nicht zu dieser Konsequenz gezwungen. Wie immer, wenn man JSPs einsetzt, besteht die Möglichkeit diese mit Java-Code anzureichern und so die angestrebte Trennung zu zerstören.

[9] Vgl. [Muhammet 2002, S. 52-55]

3.3 Apache Turbine

Neben dem recht einfachen MVC-Framework Struts existiert mit Turbine ein weiteres in dem Jakarta-Projekt.[10] Dieses widmet sich auch der Umsetzung des MVC-Musters, ist jedoch deutlich komplexer gestaltet als Struts. Es verfügt über zusätzliche Services, welche die Entwicklung von Web-Applikationen im Bezug auf alle drei Komponenten der MVC-Architektur unterstützen. So ist beispielsweise ein OR-Mapping-Service[11] zur Modellerzeugung und ein Service zur Template-basierten Erzeugung von Views verfügbar. Ergänzend stehen ein ausgeklügeltes Benutzer-Konzept, Security-Services, Logging-Services, Pool-Services und weitere Integrationsmöglichkeiten zur Verfügung. Turbine besitzt damit die Fähigkeiten einer vollständigen Entwicklungsumgebung und steht in einem vorkonfigurierten Bundle inklusive der Servlet-Engine Tomcat als so genanntes Turbine Development Kit (TDK) zum Download bereit. Marketingwirksam bezeichnen die Apache Entwickler Turbine auch als Model 2+1 MVC,[12] was auf die Erweiterung des Controllers um *Actions* zurückgeht.

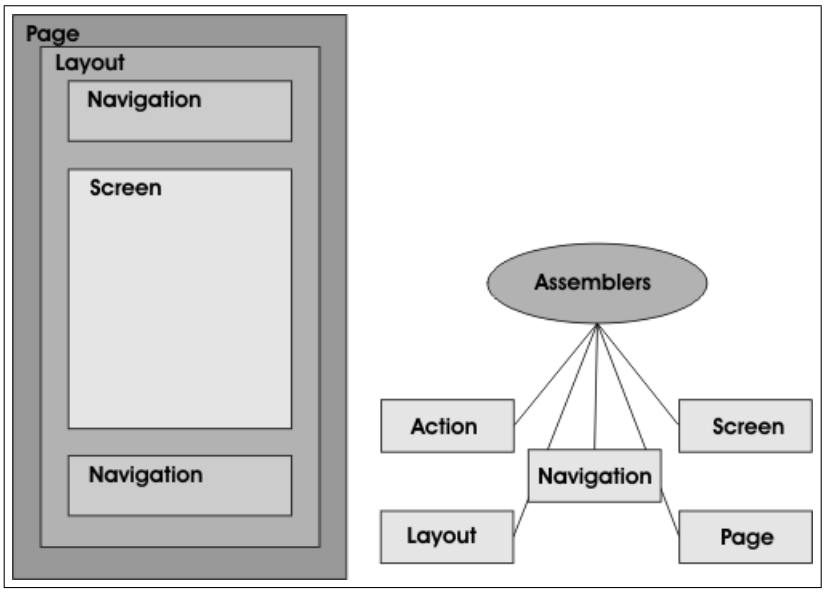

Abbildung 3-3: Turbine Specification Vgl. [Apache 2001]

Die Actions stellen eines von insgesamt fünf Modulen dar, aus denen sich Turbine zusammensetzt. Sie enthalten die eigentliche Business-Logik, z.B. der Verarbeitung

[10] Vgl. [Apache 2002c]
[11] Vgl. [Object Relational 2002]
[12] Vgl. [Stevens 2000]

von Formulardaten oder auch dem Aufruf von EJBs. Actions verarbeiten jedoch keine HTTP-Requests, sondern werden von dem *Page*-Modul aufgerufen. Durch diese Trennung können die Actions an unterschiedlichen Stellen der Applikation wiederverwendet werden. Das Page-Modul ist gleichzeitig ein Container für die restlichen Module *Layout*, *Screen* und *Navigation*. Es kümmert sich also um die ankommenden HTTP-Requests und führt die genannten Module nacheinander aus um letztendlich eine fertig formatierte HTML-Seite zu erhalten. Während im Screen die anzuzeigenden Informationen aufbereitet werden, stellt das Navigations-Modul austauschbare Navigationselemente bereit. In beiden Modulen besteht grundsätzlich die Möglichkeit externe Komponenten wie z.B. EJBs anzusprechen. Im Navigations-Modul werden Screen und Navigation miteinander verknüpft und das endgültige Erscheinungsbild der HTML-Seite bestimmt.
Für die Umsetzung der View, also von Layout, Screen und Navigation, empfehlen die Apache Entwickler den Einsatz von Template-Engines, speziell aber von *Velocity*, welches ebenso aus dem Jakarta-Projekt stammt.[13] Templates sind vollständige HTML-Seiten, die mit einfachen Macros und Variablen angereichert werden um zur Laufzeit dynamische Inhalte zu generieren. Da die Seiten zur Laufzeit geparsed werden verwirklichen sie eine Push-Strategie. Zu jedem Template wird eine Java-Klasse benötigt, um die View mit konkreten Daten zu versorgen.[14]
Auch bei Turbine übernimmt ein zentrales Servlet die Ablaufsteuerung. Aus dem Namen des angeforderten Templates wird ein äquivalente Java-Klasse gesucht. Wird eine solche gefunden, kommt diese zum Einsatz, anderenfalls wird auf Default-Klassen zurückgegriffen. In den Java-Klassen wird eine *Context*-Variable referenziert um dort in einer Art Hashmap Werte für die im Template verwendeten Variablen zu hinterlegen. Diese Variablen stehen dann für alle Screens in der Session zur Verfügung. Die Container-Funktionalität der Page wird ebenfalls durch Templates abgebildet. In Layout-Templates müssen die Variablen *screenplaceholder* und *navigation* verwendet werden um eine Art Include-Beziehung zu weiteren Screen- und Navigation-Templates herzustellen. Diese sind wiederum HTML-Fragmente mit Template-Syntax an die Java-Klassen gekoppelt sind. In den Screens wird eine neue Page gesetzt und optional eine auszuführende Action angegeben. Damit übernehmen

[13] Vgl. [Apache 2002d]
[14] Vgl. [Eschweiler 2002, S. 36-38]

sie selbst die Controller-Funktionalitäten. Aus den Actions heraus wird die Business-Logik ausgeführt und ebenfalls die Session mit Variablen gefüllt.[15]
Das Framework richtet sich nicht primär an J2EE-Entwickler, da es viele Funktionalitäten bereits von Hause aus mitbringt. Es bieten ich jedoch vielfältige Integrationsmöglichkeiten im Frontend-Bereich. Erfolgreich eingesetzt wurde Turbine in dem Portal-Framework Jetspeed.[16] Es profitiert dabei vor allem von der fein strukturierten Nutzerverwaltung und bietet eine Vielzahl spezifischer Portal-Funktionalitäten an. Auf eine detaillierte Vorstellung soll jedoch verzichtet werden, da lediglich Basis-Frameworks Gegenstand dieser Arbeit sind.

[15] Vgl. [Theis 2002, S. 76-77]
[16] Vgl. [Apache 2002a]

3.4 Barracuda

Das Barracuda-Framework aus dem Hause Enhydra[17] basiert ebenfalls auf einer Model-2-MVC-Architektur. Es distanziert sich jedoch entschieden von den JSP-basierten Pull-Ansätzen und setzt mit dem hauseigenen XMLC eine Push-Strategie ein. In der Mächtigkeit von JSP und Tag-Libraries sah man zunehmends einige Schwachpunkte: Zum einen muss der Web-Designer über geringe Java-Kenntnisse verfügen zumindest aber die neuen Tags der Tag-Library erlernen. Zwar wird durch die Verwendung der Tag-Libraries der Java-Code in den JSP-Seiten minimiert, man wird jedoch keinesfalls dazu gezwungen. So findet man häufig Formatierungen und Konvertierungen von Zahlen und Strings direkt zwischen den HTML-Tags wieder. Auch Template-Engines vollziehen diese Trennung nicht endgültig. Die Tag-Libraries verschwinden zu Gunsten einer spezifischen Makrosprache – für den View-Designer ändert dies relativ wenig. Auf elegante Weise löst XMLC das aufgeführte Problem der Rollentrennung der View. Aus HTML bzw. XML Dateien werden Java-Klassen generiert, die den exakten Inhalt als Document Object Model (DOM) enthalten. Zusätzlich werden jedoch spezielle Methoden generiert, die den Zugriff auf einzelne Elemente kapseln. Dazu ist es nötig, die Original-Tags mit speziellen Attributen zu versehen. Das Attribut *id="sampleTag"* würde beispielsweise die Methoden *getElementSampleTag()* und *setTextSampleText()* generiert. Weiterhin steht das Attribut *class="group"* zur Gruppierung und eventuellen Filterung von Tags zur Verfügung. Die erzeugten Java-Klassen werden nun von einem Servlet verwendet, um den Inhalt der ausgezeichneten Tags zu verändern und eine komplette HTML-Seite zurückzuliefern. Im übrigen besteht die Möglichkeit den DOM über den Localization Service zu internationalisieren. Der XMLC-Ansatz verwirklicht eine vollständige Trennung von Inhalt und Layout. Es wird sämtliche Programm-Logik aus den HTML-Seiten verbannt.[18]

Wenn Barracuda einen HTTP-Request empfängt, so wird zunächst der Typ des Clients festgestellt. So können unterschiedliche Clients entsprechend ihren Darstellungsfähigkeiten wie z.B. Scripting und Format mit Informationen versorgt werden. In einem nächsten Schritt, wird aus dem Request ein Event generiert und entsprechend weiterverarbeitet. Ein weiteres Package bietet ein automatisches Mapping und Validierung von HTML-Formularen, wodurch die Arbeit erheblich vereinfacht wird.

[17] Vgl. [Barracuda 2002b]
[18] Vgl. [Young 2002]

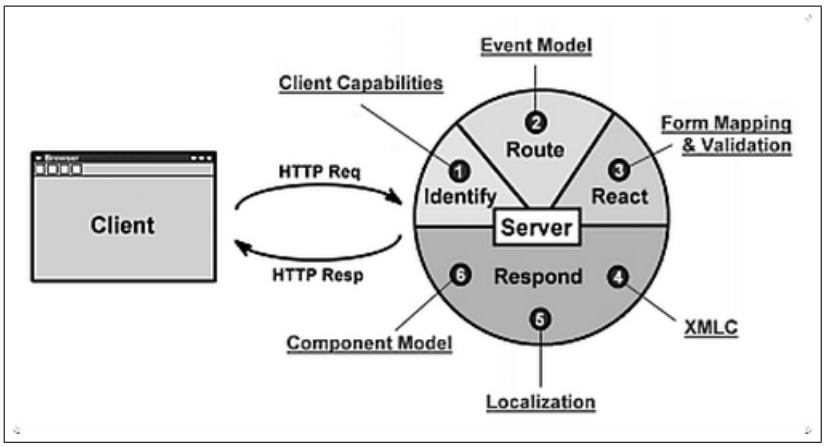

Abbildung 3-4: Barracuda Architektur Vgl. [Barracuda 2002a]

Die Elemente eines Formulars werden auf Java-Objekte abgebildet – bei Validierungsfehlern werden entsprechende Exceptions geworfen. Nun kann die eigentliche XMLC-basierte Generierung eines HTTP-Responses beginnen.
Die Barracuda-Architektur ist streng Komponenten-basiert, welches durch ein serverseitiges Event-Modell verbunden wird. Dadurch wird das übliche Request-Processing flexibler gestaltet. Barracuda sieht zwei Arten von Events vor die Request-Events und die Response-Events. Die korrespondierenden Listener der Request-Events können unter MVC-Gesichtspunkten als Controller aufgefasst werden, die Listener der Response-Events übernehmen die Aufgaben der View.

3.5 Expresso

Jcorporate Expresso[19] stellt ein ähnlich komplett ausgestattes Basis-Framework wie Turbine dar. Es besteht aus drei Kernkomponenten – den *Schemas*, den *Controllers* und den *DatabaseObjects*. Abgerundet wird die Architektur durch ein aus Security-Konzept und eine Reihe zusätzlicher Funktionen wie z.B. Logging, Caching, Connection-Pooling, etc.

Das Schema-Objekt ist als eine Art Anwendungs-Kontext zu verstehen – es registriert in einer von *com.jcorporate.expresso.core.dbobj.Schema* abgeleiteten Klasse die anwendungsspezifischen Einstellungen insbesondere aber natürlich die verwendeten DatabaseObjects und Controller. Mit den DatabaseObjects wurde ein einfacher OR-Mapping-Mechanismus geschaffen, der sich um die Persistenz der Daten kümmert. Sie greifen dabei auf DatabaseConnections zurück, die einen JDBC-unabhängigen Connection Pool bereit stellen. Ein *DBObject* bezieht sich auf genau ein Tabelle und verfügt über *add()*, *delete()* und *update()* Methoden sowie eine Reihe von Finder-Methoden. [20]

Eine zentrale Rolle nehmen die Controller ein. Sie sind als endliche Automaten modelliert, und führen bei jedem Zustandswechsel Actions aus. Damit übernehmen sie nicht nur die Aufgaben des MVC-Controllers, sondern architekturbedingt auch die des Models. Eine strikte Trennung der Rollen wird also in Expresso nicht verwirklicht. Sehr wohl können aber Controller mehrere User Interfaces bedienen neben JSPs und Template-Lösungen kann der Output auch als XML bzw. via XSLT transformiert zurückgegeben werden.[21] Ein integriertes Sicherheitskonzept ermöglicht die automatische Prüfung von Zugriffsrestriktionen. Dazu sind die speziellen Klassen *SecuredDBObject* und *DBController* zu verwenden.

Mit der Version 4.0 wurde das bereits beschriebene Framework Struts integriert, indem die Controller direkt von den Struts-Actions abgeleitet wurden und sich so in das Request-Mapping von Struts eingliedern. Man erweiterte damit das reine MVC-Framework Struts um die Expresso-Funktionalitäten um damit bessere Synergieeffekte in der Nutzergruppe der Web-Frameworks zu erzielen.[22]

[19] Vgl. [Expresso 2002]
[20] Vgl. [Nash 2001]
[21] Vgl. [Expresso 2001a]
[22] Vgl. [Pilgrim 2001]

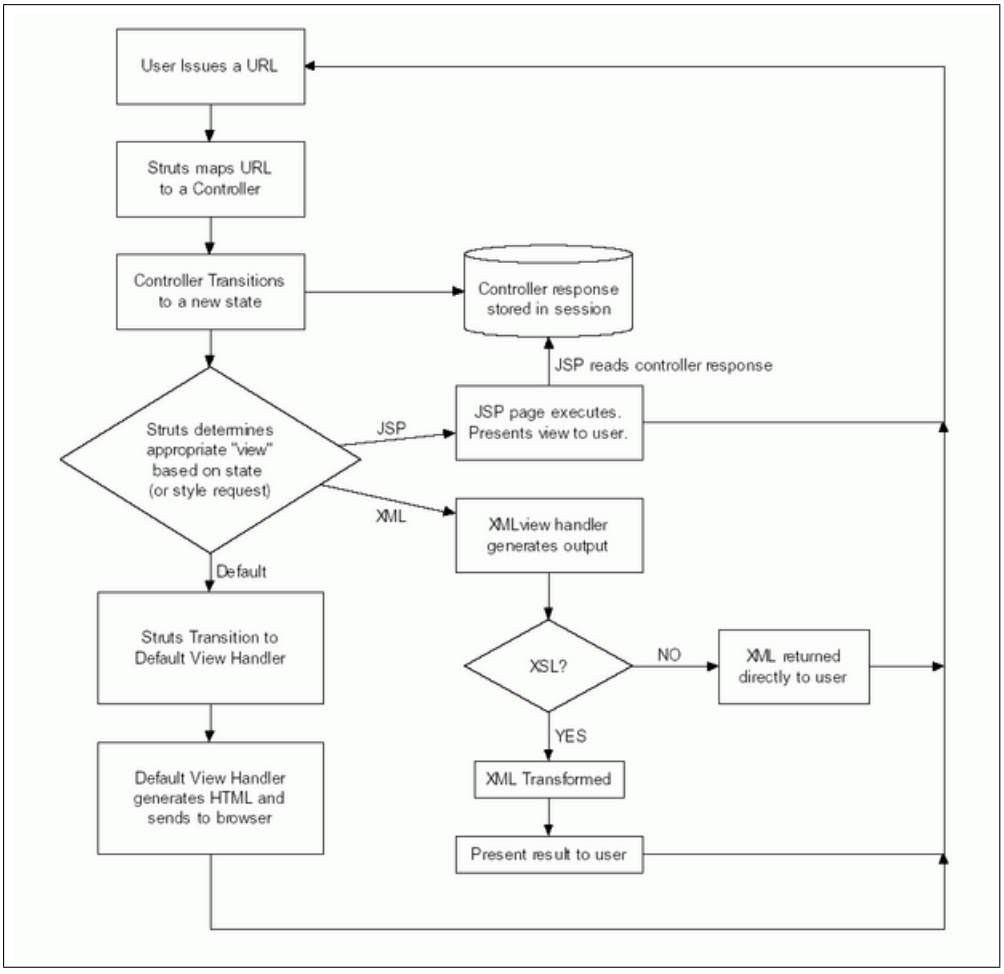

Abbildung 3-5: Expresso-Struts-Integration Vgl. [Expresso 2001b]

3.6 WebWork

WebWork ist ein Community-Projekt,[23] dass in mancher Hinsicht dem eingangs vorgestellten Framework Struts ähnelt, sich aber dennoch entschieden davon distanziert. Man will sich speziell durch die Einfachheit im Feld der Präsentations-Frameworks behaupten. Im Gegensatz zu den bisher vorgestellten Frameworks verwirklicht es ein hierarchisches MVC-Konzept. Kern des Frameworks ist die Action-API – sie ist die Grundlage für die Implementierung der H-MVC-Controller. Alle HTTP-Requests werden an ein Dispatcher-Servlet weitergeleitet, welches die passende Action sowie einen Kontext bestimmt. Der Kontext wird ausgehend von der *DefaultActionFactory* in einer Kette weiterer Factories bestimmt. Diese Factories bilden eine Hierarchie, wodurch die Ablaufsteuerung effizienter getrennt wird. Danach kann die Action zur Ausführung gebracht werden. Dazu muss sie mittels Settern initialisiert werden, und schliesslich mit *execute()* ausgeführt werden. Anschließend kann man mit Gettern auf die Variablen zugreifen. Ähnlich wie in Struts wird der Rückgabewert einer Action vom Dispatcher verwendet, um eine neue View zu bestimmen. Die Konfiguration von WebWork erfolgt über einfache Properties-Dateien.
Im Gegensatz zu Struts bringt WebWork verschiedenste Möglichkeiten der View-Gestaltung mit. Neben JSPs können auch Template-Engines angeschlossen, bzw. XML/XSLT-Transformationen eingesetzt werden. Die JSPs werden durch eine einfache aber leistungsfähige Tag-Library unterstützt. Sie profitiert vor allem von der Expression Language (EL) – einem mächtigen Werkzeug um Bedingungen zu formulieren und durch einen Pull-Mechanismus die Views mit Daten zu versorgen. Die EL kommuniziert ausschließlich mit dem Value Stack (VS). Auf dem VS werden Objekte abgelegt, die dann mittels der EL abgerufen werden können. Durch die Erzeugung von speziellen Properties-Dateien – den Ressource Bundles – werden sämtliche Abfragen transparent internationalisiert. WebWork bringt außerdem eine Unterstützung für das bekannte Form-Handling und die zugehörige Validierung mit. Insgesamt ist WebWork ein flexibleres Framework als Struts, da es nicht über solch gewachsene Strukturen verfügt. Bisher hat es sich jedoch nicht gegen die Marketing-Macht der Apache Software Foundation durchsetzen können.

[23] Vgl. [WebWork 2002]

4 Fazit

Der Einsatz von Frameworks ist immer mit einem anfänglichen Lernaufwand verbunden. Er lohnt sich aber dennoch in den meisten Fällen, da schnell beträchtliche Synergieeffekte auftreten. Die Anwendung von Anfang an auf einer soliden Architektur entwickelt die bei richtiger Anwendung auch wartbar bleibt. Gleichzeitig wird eine effiziente Rollentrennung zwischen Web-Designer und Java-Programmierer ermöglicht.

Welches der vorgestellten Frameworks nun tatsächlich am geeignetsten ist, kommt sicher auf den Einzelfall an. In erster Linie ist aber der Grad der notwendigen Flexibilität entscheidend. So bietet Struts aufgrund seiner Einfachheit relativ viel Raum für individuelle Anpassungen. Auf der anderen Seite kommen Turbine und Expresso von Hause aus mit zahlreichen Zusatz-Services daher, was besonders für kleinere Web-Anwendungen von Vorteil sein wird. Zu bedenken ist, dass der damit verbundene Einarbeitungsaufwand um so beträchtlicher wird. Es ist also abzuwägen, ob man die mitgelieferten Funktionen verwenden möchte, oder lieber auf eigene Persitenz-, Logging- und Security-Mechanismen zurück greift. Weiterhin spielen auch die bisherigen Erfahrungen und Vorkenntnisse der Projektmitarbeiter eine Rolle. Der angesprochene Lernaufwand minimiert sich bei dem Einsatz eines bekannten Frameworks. Mit der Entscheidung für ein Framework schränkt sich auch die Art der Gestaltung der Views ein. Hat man bei Expresso noch die Wahl zwischen JSP, Templates und XSLT, so kann Struts zweckmäßigerweise nur mit JSP, Turbine nur mit Templates und Barracuda nur mit XMLC eingesetzt werden. Inwieweit dies einen Einfluss auf die Auswahl des Frameworks hat, liegt eher in der Sympathie des Projektleiters. Den Web-Designer sollte die Entscheidung nur minimal treffen, da seine Unabhängigkeit gerade erklärtes Ziel eines jeden Frameworks ist. Grundsätzlich sollte man sich an den Leitsatz *„nie gegen ein Framework entwickeln“* halten.

Abschließend sei noch erwähnt, dass Struts im Jahr 2001 eines der populärsten Frameworks gewesen ist. Ausschlaggebend dafür wird die Tatsache sein, dass es als reines MVC-Framework konzipiert wurde. Dadurch verfügt es nicht über den Overhead an Zusatz-Services und ist zudem schneller erlernbar. Inwieweit es von hierarchischen Frameworks wie z.B. Webwork abgelöst werden kann, muss die Entwicklergemeinde entscheiden.

Literatur

[Apache 2001] Apache: *Turbine Specification*. 2001. – URL `http://jakarta.apache.org/turbine/turbine-2/fsd.html`. – Zugriffsdatum: 05.06.2002

[Apache 2002a] Apache: *JetSpeed*. 2002. – URL `http://jakarta.apache.org/jetspeed/`. – Zugriffsdatum: 05.06.2002

[Apache 2002b] Apache: *The Struts Web Application Framework*. 2002. – URL `http://jakarta.apache.org/struts/`. – Zugriffsdatum: 05.06.2002

[Apache 2002c] Apache: *Turbine*. 2002. – URL `http://jakarta.apache.org/turbine/turbine-2/`. – Zugriffsdatum: 05.06.2002

[Apache 2002d] Apache: *Velocity*. 2002. – URL `http://jakarta.apache.org/velocity/`. – Zugriffsdatum: 05.06.2002

[Barracuda 2002a] Barracuda: *Barracuda - Framework Comparisons*. 2002. – URL `http://barracuda.enhydra.org/cvs_source/Barracuda/docs/barracuda_vs_struts.html`. – Zugriffsdatum: 05.06.2002

[Barracuda 2002b] Barracuda: *Barracuda - MVC Presentation Framework for Web Applications*. 2002. – URL `http://barracuda.enhydra.org/`. – Zugriffsdatum: 05.06.2002

[Cai u. a. 2000] Cai, Jason ; Kapila, Ranjit ; Pal, Gaurav: *HMVC: The layered pattern for developing strong client tiers*. 2000. – URL `http://www.javaworld.com/javaworld/jw-07-2000/jw-0721-hmvc.html`. – Zugriffsdatum: 02.06.2002

[Duffey 2000] Duffey, Kevin: *Kevin on Model 1, Model 1.5 and Model 2*. 2000. – URL `http://www.brainopolis.com/jsp/mvc/KDuffey_MVC.html`. – Zugriffsdatum: 05.06.2002

[Eschweiler 2002] Eschweiler, Sebastian: Web-Frameworks Webmacro, Velocity und Cameleon im Vergleich. In: *Javamagazin* 03 (2002), S. 34–40

[Expresso 2001a] Expresso: *Expresso Controller Objects*. 2001. – URL `http://www.jcorporate.com/econtent/Content.do?state=template&template=2&resource=636&db=default`. – Zugriffsdatum: 10.06.2002

[Expresso 2001b] Expresso: *Struts / Controller execution in Expresso*. 2001. – URL `http://www.jcorporate.com/expresso/doc/struts1.pdf`. – Zugriffsdatum: 10.06.2002

[Expresso 2002] Expresso: *Expresso - Application Development Framework*. 2002. – URL `http://www.jcorporate.com/html/products/expresso.html`. – Zugriffsdatum: 05.06.2002

[Gamma u. a. 1996] Gamma, Erich ; Helm, Richard ; Johnson, Ralph ; Vlissides, john: *Entwurfsmuster*. Bd. 5., korrigierter Nachdruck. Addison Wesley Verlag, 1996. – ISBN 6-8273-1862-9

[Muhammet 2002] Muhammet, Öztürk: Das Web-Framework Apache Struts in Version 1.0. In: *Javamagazin* 04 (2002), S. 48–57

[Nash 2001] Nash, Michael: *Expresso Developer's Guide*. : Jcorporate Ltd. (Veranst.), 10 2001. – URL `http://www.jcorporate.com/expresso/doc/edg/edg.html`. – Zugriffsdatum: 10.06.2002

[Object Relational 2002] Object Relational: *Transparent persistence in object-relational mapping*. 2002. – URL `http://www.object-relational.com/articles/transparent_persistence.html`. – Zugriffsdatum: 11.06.2002

[Pilgrim 2001] Pilgrim, Peter: Best Practice with Expresso Framework: Using a framework to create a web application. In: *TheServerSide Developer Ressources* TheServerSide (2001). – URL `http://www.theserverside.com/resources/article.jsp?l=Expresso`. – Zugriffsdatum: 10.06.2002

[Stevens 2000] Stevens, Jon: *What is model 2+1?* 2000. – URL `http://jakarta.apache.org/turbine/turbine-2/model2+1.html`. – Zugriffsdatum: 05.06.2002

[Sun 2001] Sun: Model-View-Controller Architecture. In: *J2EE Design Patterns* (2001). – URL `http://java.sun.com/blueprints/patterns/j2ee_patterns/model_view_controller/index.html`. – Zugriffsdatum: 05.06.2002

[Sun 2002] Sun: *JavaServer Faces Technology*. 2002. – URL `http://java.sun.com/j2ee/javaserverfaces/`. – Zugriffsdatum: 05.06.2002

[Sweeting u. a. 2002] Sweeting, Steven ; Jones, Clive ; Rustad, Aaron: Integrating and Mapping a Web Application MVC Pattern. In: *Java Developer's Journal* (2002), Nr. 06. – URL `http://www.sys-con.com/java/archives3/0610/sweeting/fig2.jpg`. – Zugriffsdatum: 05.06.2002

[Theis 2002] Theis, Fabian J.: Turbine - das Apache Web Application Framework, Teil 1. In: *Javamagazin* 06 (2002), S. 73–77

[WebWork 2002] WebWork: *WebWork Documentation*. 2002. – URL `http://opensymphony.com/webwork/`. – Zugriffsdatum: 18.07.2002

[Yin 1995] Yin, Tong S.: *The Smalltalk MVC paradigm with pluggable views*. 1995. – URL `http://citeseer.nj.nec.com/331003.html`. – Zugriffsdatum: 05.06.2002

[Young 2002] Young, David H.: *A Friendly Game of Tug War: XMLC vs JSP*. 2002. – URL `http://www.theserverside.com/resources/article.jsp?l=XMLCvsJSP`. – Zugriffsdatum: 05.06.2002