Frameworks zur Implementierung von Hypertext-Anwendungen

von Thorsten Berger[1]

Problemseminar
- Neue Entwicklungen in der Telematik -
Dr. Sami Beydeda

26.01.2004

[1]mail@thorsten-berger.net

Inhaltsverzeichnis

1. Einleitung

Das World Wide Web hat unser Leben in den letzten 10 Jahren nachhaltig geprägt. Es ist ein Informationsmedium, dessen Ausmaße wohl niemand geahnt hat. Tagtäglich werden wir mit ihm konfrontiert, wir nehmen aktiv oder passiv daran teil. Sei es um uns zu informieren, zu unterhalten, zu shoppen oder einfach nur zu „surfen“, es ist einfach nicht mehr wegzudenken. Kaum eine Firma kann es sich mehr leisten, nicht im WWW präsent zu sein. Dabei reichen schon lange nicht mehr nur rein statische Seiten, vielmehr entstehen hochkomplexe dynamische Websites, deren Erstellung eine besondere Herausforderung an Softwareentwickler und nicht zuletzt an die finanziellen Möglichkeiten des Auftraggebers ist. Professionelle Auftritte im Web verschlingen eine Unmenge an Zeit und Geld. Selten sind dem Benutzer die dahinterliegenden Ressourcen überhaupt bewusst.
Zur Entwicklung solcher Hypertext-Anwendungen stehen unzählige Hilfsmittel, freie und kommerzielle, zur Verfügung. In diesem Dokument will ich besonders auf das Open Source Framework Struts aus dem Apache Jakarta Projekt eingehen und später noch das Framework XForms für Formulare im Web vorstellen, das seit kurzem ein W3C-Standard ist.

1.1. Was ist ein Framework?

Eine allgemeine Definition für den Begriff des Frameworks zu finden fällt schwer. Helmut Balzert liefert uns in [1] folgendes:

„Ein Rahmenwerk (framework) ist ein durch einen Software-Entwickler anpassbares oder erweiterbares System kooperierender Klassen, die einen wiederverwendbaren Entwurf für einen bestimmten Anwendungsbereich implementieren.“

Ein Framework ist also die Grundlage unserer zu entwickelnden Anwendung, das, getreu dem Komponentengedanken, uns verschiedene Bausteine zur Verfügung stellt, die immer wieder verwendet werden können und nach einem bestimmten Designmuster zusammengefügt werden müssen. Frameworks bieten damit eine gewisse Abstraktion über dem eigentlichen Problem, d..h. der Entwickler muss sich nicht mit allen Einzelheiten der Technologie beschäftigen, die dem Framework zu Grunde liegt. Schließlich kann man sagen, dass sie die Entwicklungszeit und auch die damit verbunden Entwicklungskosten deutlich reduzieren sollen.

1.2. Was ist eine Hypertext-Anwendung?

Zunächst versteht man darunter eine Anwendung, die ihre Darstellung (also die graphische Oberfläche) im World Wide Web hat. Sie ist dabei nicht auf bestimmte Techniken/Notationen zur Visualisierung festgelegt, sondern es stehen ihr viele verschiedene Anzeigeformate zur Verfügung, z.B. HTML, WML, XML usw. Genauso existieren eine Unzahl an Technologien zur Implementierung des sog. Backends, also der Applikationslogik, welche die Oberfläche steuert.
Hypertext-Anwendungen finden auch zunehmend Verbreitung in Intranets, also unternehmensinternen Netzen, die schwere proprietäre Software durch leichtgewichtige, standardisierte Web-Technik ersetzen soll.

2. Grundlagen

Frameworks setzen auf bereits bestehenden Technologien auf und abstrahieren von Details der Implementierung. Diese Technologien zur Entwicklung von Hypertext-Anwendungen will ich nun kurz vorstellen.
Alles begann mit dem „Common Gateway Interface“, kurz CGI. Es war die erste definierte Schnittstelle zwischen dem eigentlichen Webserver, der bekanntlich nur statische Seiten ausliefert und Applikationsprogrammen, die auf derselben Plattform laufen. Der Webserver leitet Anfragen, mit allen Request-Informationen, an das Programm weiter, indem er einen neuen Prozess auf dem Server startet. Die Ausgaben werden durch dieselbe Schnittstelle wieder an den Client vermittelt. Diese Vorgehensweise hat zwar den Vorteil, dass jedes auf der Server-Plattform ausführbare Programm genutzt werden kann, aber es muss auch immer ein neuer Prozess gestartet („fork“) werden, was, besonders bei vielen Requests, viele Ressourcen kostet. Außerdem stehen keine professionellen web-spezifischen APIs zur Verfügung, z.B. Session- und Security-Funktionen. Aufgrund dieser Nachteile wird CGI heute nur noch für kleinere Projekte verwendet.
Diese Nachteile versuchten Web-Server-Hersteller durch spezielle „Server Extensions“ zu beseitigen. Dabei existieren exakte Interface-Definitionen, die nur ausprogrammiert werden mussten. Unter anderem waren das die NSAPI für Netscapes Server und ISAPI für den Microsoft IIS. Die Performance war zwar deutlich besser, dafür sind die Erweiterungen allerdings auch inkompatibel zueinander, d.h. ein Entwickler musste sich immer auf eine Plattform festlegen.
Einen anderen Ansatz verfolgen serverseitige Scriptsprachen, eingeführt von Netscape mit den „Server Side Includes“ (SSI). Ein Interpreter führt in den HTML-Quelltext eingebetteten Script-Code bei jedem Request aus und liefert das Resultat als HTML an den Client. Das proprietäre SSI ist heute fast komplett von modernen Scriptsprachen, wie PHP, JSP, ASP verdrängt worden, nicht zuletzt wegen einer mangelhaften Mächtigkeit der dahinterliegenden API.
Noch einen anderen Weg gehen Java-Servlets. In der J2EE spezifiziert sind sie eine Weiterentwicklung der Server-Extensions. Sie sind, da sie auf Java basieren, plattformunabhängig, sehr effizient und performant, da sie in kompilierter Form vorliegen und die JVM nicht bei jedem Request gestartet werden muss. Weiterhin liegt ihnen die komplette Mächtigkeit der Sprache Java zu Grunde, die nicht zuletzt zur Entwicklung sicherer und stabiler Programme beiträgt. Servlets sind auch nicht nur auf Hypertext-Anwendungen beschränkt. JSP, als Scriptsprache, basiert auf der Servlettechnologie, und zwar werden sie zur Laufzeit in Servlets übersetzt, müssen also nicht bei jedem Request neu interpretiert werden, wodurch sich ein nicht unerheblicher Geschwindigkeitsvorteil ergibt.
Ich selbst arbeite zur Zeit an einem Open Source Projekt, das eine Online-Übungsverwaltung basierend auf Servlets entwickelt, die zu Vorlesungen am Lehrstuhl AIS (am Institut für Informatik) der Universität Leipzig eingesetzt wird. Die Projektseite befindet sich auf http://uebman.sourceforge.net.

Das waren, kurz zusammengefasst, die Technologien, die uns derzeit zur Entwicklung von Hypertext-Anwendungen zur Verfügung stellen. Ich beschränke mich im folgenden auf die Java-Plattform.

2.1. Architekturen und Modelle

Hypertext-Applikationen sind verteilte Anwendungen. Nach Sun sollte die Architektur in drei Schichten erfolgen:

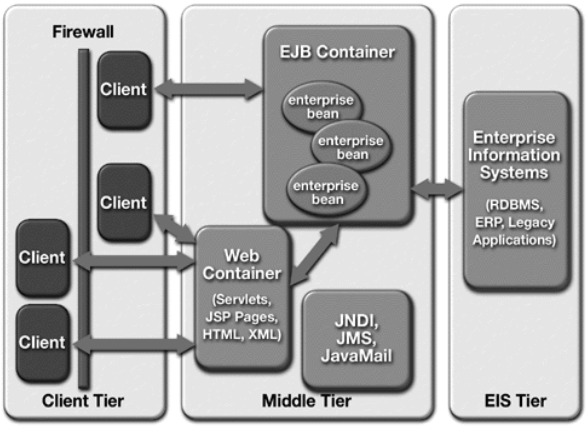

Abbildung 1: „Multi-Tier"-Architektur, Quelle: J2EE auf java.sun.com

Die Client-Tier ist einfach der Browser des Endbenutzers, der die vom Middle-Tier gelieferten Daten (HTML, WML, XML ...) darstellt. Diese wiederum besteht aus dem Web-Server, dem Container, und der eigentlichen Applikation. Die kann nun auf die „Enterprise Information Systems" (EIS), in der Datenbanken und andere internen Ressourcen enthalten sind, zugreifen.
Uns interessiert im folgenden aber nur die mittlere Schicht.

Für einfache Anwendungen ist es möglich, sofort mit der Implementierung mit Hilfe der im letzten Abschnitt beschriebenen Technologien anzufangen. Viele Projekte, die klein angefangen haben, entwickelten sich schnell, bedingt durch Erweiterungen und neue Anforderungen, zu größeren Applikationen. Spätestens da zeigt sich das Problem, dass man entweder in Servlets nicht mehr den Code durch zuviel in „println()"-Anweisungen eingebettetes HTML oder umgekehrt in JSP kein HTML vor lauter Java-Code erkennen kann. Nur strukturiertes Programmieren und Abstrahieren hilft hier weiter, ansonsten explodieren die Kosten für die Wartung und Erweiterung solcher Projekte.

Die Stichwörter heißen Modularisierung sowie Trennung von Applikationslogik und visueller Darstellung. Für ersteres gibt es etablierte Komponententechnologien, z.B. JavaBeans bzw. Enterprise JavaBeans (EJB) oder COM/COM+ von Microsoft.
Der Standardansatz für letzteres Problem lautet MVC: „Model-View-Controller", das ursprünglich zur Entwicklung von graphischen Oberflächen in Smalltalk-80 gedacht war.
Die Applikation wird in drei Komponenten aufgeteilt, die so gut wie möglich gegeneinander gekapselt werden sollen.

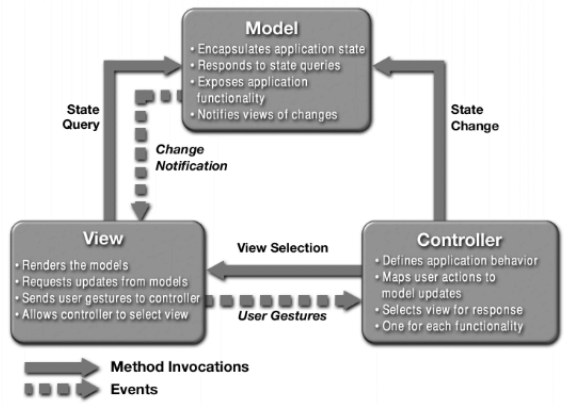

Abbildung 2: MVC, Quelle: J2EE auf java.sun.com

Das Model repräsentiert die Applikationslogik und ihre Datenbasis, also die eigentliche Anwendung. Die View ist die visuelle Darstellung der Anwendung und der Controller steuert den zentralen Kontrollfluss, er reagiert auf Aktionen des Benutzers während dieser mit der Anwendung kommuniziert. Verhindert werden dadurch Code-Konstruktionen, die Logik und Darstellung vermischen.
Neben diesen Vorteilen sollen aber auch nicht die Nachteile von MVC verschwiegen werden. Nach Allen Holubs Meinung [2] ist dieser Ansatz ungeeignet um von der Implementierung des Models zu abstrahieren. Des weiteren ist eine nicht unerhebliche Zahl von Transaktionen zwischen den einzelnen Komponenten erforderlich, die leicht ausarten kann, man denke hier nur an die Unzahl von Listenern, die es in Java für die unterschiedlichsten Zwecke gibt.

Dass solche Modelle zur Entwicklung größerer Anwendungen für die Servlet Plattform fehlen hat auch Sun schnell bemerkt und definierte zwei Architekturen für Hypertext-Anwendungen: „Model 1“, das lediglich die alleinige Verwendung von JSP vorschreibt und „Model 2“, das auf MVC basiert und bei dem das Model mit EJB, die View mit JSP und der Controller mit Servlets realisiert werden soll. MVC und „Model 2“ werden heute (fälschlicherweise) fast synonym verwendet. Zu beachten ist, dass „Model 2“ eine Abwandlung des originalen MVC sind, denn es ist in einer Hypertext-Anwendung aufgrund des zustandslosen HTTP-Protokolls u.a. nicht möglich, dass das Model die View über Änderungen informiert. Vielmehr findet der Kontrollfluss innerhalb eines Request/Response-Zyklus statt.

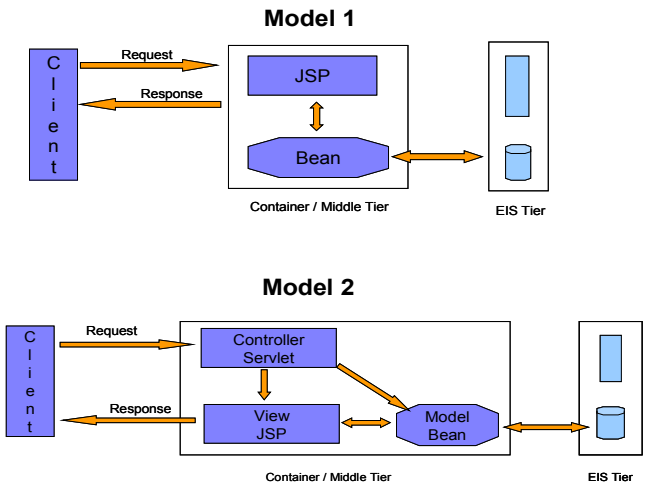

Abbildung 3: Model 1 und 2

3. Struts

Nach aller Theorie kommen wir jetzt zu dem sehr erfolgreichen Framework Struts, das seit 1999 Teil des Apache Jakarta Projektes ist und von IBM sowie Sun unterstützt wird. Nicht zuletzt wird die Tatsache, dass Struts Open Source, plattformunabhängig und sehr gut dokumentiert ist, zu seiner weiten Verbreitung beigetragen haben.
Struts ist ein modernes Framework, das auf J2EE basiert und das „Model 2“ umsetzt. Als Container kommt häufig der auch im Jakarta Projekt entwickelte Tomcat[1] zum Einsatz.
Struts verfolgt eine relativ lose Kopplung der drei Komponenten des MVC Modells, wodurch ein hoher Grad an Wiederverwendbarkeit erreicht werden soll. Das Framework unterstützt in den mit JSPs realisierten Views die Internationalisierung („i18n“) basierend auf verschiedenen Internationalisierungsfunktionen von Java (Stichwort „Locale“). Als Controller dient ein flexibel konfigurierbares Servlet, das alle Requests entgegennimmt und die Reaktion darauf steuert. Das Model wird in Front-End- und Back-End-Model getrennt. Ersteres besteht hauptsächlich aus sog. ActionForm-Beans, während das Back-End-Model in Struts nicht näher spezifiziert ist, zur Umsetzung können beliebige Technologien, wie z.B. JavaBeans oder EJB verwendet werden.

3.1. Übersicht

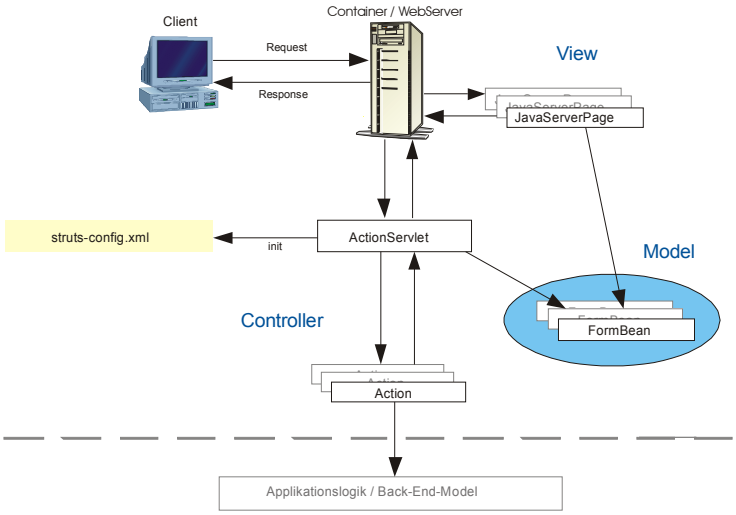

Abbildung 4: Struts Architektur und Ablauf eines Request/Response-Zyklus

Der Ablauf einer Anfrage an eine mit Struts entwickelte Applikation ist folgender:
Die Anfrage des Client löst die Ausführung des zentralen Controllers „ActionServlet“ aus. Dieses initialisiert sich evtl. (Singleton-Pattern) mit Hilfe der Struts Konfigurationsdatei *struts-config.xml*. In dieser stehen sog. Mappings zu Action-Objekten, die von ihm mit Hilfe einer Auswertung von Anfrage-Parametern ausgewählt und aufgerufen werden. In diesen Actions ist weitere spezielle Controller-Funktionalität enthalten. Das aufgerufene Action-Objekt kann entweder den Request selbst bearbeiten oder ihn an ein anderes weiterleiten. Wählt es ersteres, so ist es für die Ausführung des Models (durch z.B. Aufruf entspr. Bean-Methoden) und das Auswählen der View zuständig.

[1] http://jakarta.apache.org/tomcat

Eine wichtige Rolle spielen Beans in Struts, sowohl im Model als auch im Controller. Sie können z.B. zum flexiblen Einsatz von Formularen eingesetzt werden. Diese sog. ActionForm-Bean enthält die eingegebenen Formulardaten und behält sie auch über mehrere Requests hinweg, um z.B. die Validierung und die Korrektur von Teilen der Formulardaten zu ermöglichen. JSPs (also die View Komponente) sammeln Daten für die ActionForm-Bean, welche diese auch verifizieren können, bevor eine entsprechende Action ausgeführt wird. Bei Fehlern existieren Mechanismen zur Anzeige von zielgerichteten Fehlermeldungen und der erneuten Abfrage der fehlerhaften Daten. Struts bringt u.a. dafür sein eigenes Tag-Set für JSPs mit. Diese Tags sorgen für die logische Verbindung zwischen einzelnen Formularfeldern, d.h. sie können u.a. Initialisierungswerte eintragen oder Fehlermeldungen einbetten. Zu beachten ist, dass hier normale HTML-Formulare verwendet werden. Ein eigenes Framework speziell für Formulare wird mit XForms später vorgestellt werden.
Die Internationalisierung von Anwendungen ist durch die Sprache Java bereits gewährleistet. Der Vorteil davon sind nicht nur landesspezifische numerische Formate (Währungen, Datumsangaben), sondern auch die Möglichkeit bestimmte Nachrichten und Beschriftungen zentral in einem Language Pack zu verwalten.

Im folgenden werden die einzelnen Komponenten detaillierter erläutert und auch Beispiele zur Implementierung angegeben. Diese werden sich auf Beispiele aus „The Struts User's Guide" [4] stützen.

3.2. Controller

Der Controller besteht in Struts aus zwei Komponenten: der Klasse ActionServlet, einem zentralen Servlet, das den Request entgegennimmt und verschiedenen Actions, die der Entwickler implementieren muss.

3.2.1. ActionServlet

Das ActionServlet spielt eine zentrale Rolle im Workflow der gesamten Applikation. Wie bereits erwähnt nimmt es alle Requests von Clients entgegen und wertet diese anhand verschiedener Request-Parameter aus, z.B. die RequestURI, verschiedene QueryStrings usw. Das Resultat davon ist ein bestimmtes Action-Objekt, an das die Kontrolle übergeben wird. Erfahrene Anwender erkennen hier das Factory-Pattern. Die Request-Parameter werden in ActionForm-Beans gespeichert.
Die Informationen über das Mapping erhält ActionServlet dabei aus Instanzen der ActionMapping-Klasse. Damit man diese nicht per Hand schreiben muss, kann Struts selbst diese ActionMappings aus bestimmten Abschnitten aus der struts-config.xml erzeugen. In dieser Konfigurationsdatei wird auch die Zuordnung zu ActionForm-Beans gespeichert.

Sehen wir uns eine beispielhafte struts-config einmal an:

```
<struts-config>

   <form-beans>
      <form-bean name="logonForm" type="org.apache.struts.example.LogonForm"/>
   </form-beans>

   <global-forwards type="org.apache.struts.action.ActionForward"/>
      <forward name="logon" path="/logon.jsp" redirect="false"/>
   </global-forwards>

   <action-mappings>
      <action
            path="/logon"
            type="org.apache.struts.example.LogonAction"
            name="logonForm"
            scope="request"
            input="/logon.jsp"
            unknown="false"
            validate="true"/>
   </action-mappings>

</struts-config>
```

Das „form-beans“-Tag definiert spezifische ActionForm-Beans, im Beispiel ist es ein „LogonForm“, das Daten eines Login-Formulars enthält und den logischen Namen „logonForm“ bekommt.
Im „global-forwards“-Bereich können Aliase für RequestURIs angegeben werden, welche von jeder Action jederzeit mit ihrem Aliasnamen angesprochen werden können.
Schließlich findet sich innerhalb der „action-mapping“-Tags für jeden möglichen Request an das ActionServlet ein ActionMapping:

path:	RequestURI, bei welcher die Action ausgeführt wird
type:	auszuführendes Action-Objekt
name:	die zu verwendende ActionForm-Bean
unknown:	soll diese Action standardmäßig ausgeführt werden wenn keine andere passt? also z.B. die RequestURI vom Client ungültig ist.
validate:	„validate()“-Methode der ActionForm-Bean aufrufen und auswerten?

3.2.2. Actions

Actions sind Unterklassen von „Action“ und haben die Aufgabe, die eigentliche Applikationslogik auszuführen, das Front-End-Model, d.h. die ActionForm-Beans zu verändern und ein bestimmtes JSP zur Anzeige auszuwählen.
Von der Action wird bei einem zu ihm gehörigen Request die „execute()“-Methode aufgerufen und darin das dazugehörige ActionForm-Bean sowie versch. andere Informationen übergeben.

```
public ActionForward execute(ActionMapping mapping,
                             ActionForm form,
                             HttpServletRequest request,
                             HttpServletResponse response)
```

Anhand dieser Daten kann die Action nun die Applikationslogik, das Backend, ausführen. Es ist verlockend, diese gleich mit in die Action zu integrieren. Bei kleineren Projekten dürfte das noch funktionieren, aber prinzipiell sollte man von so einer Mischung Abstand nehmen.

Eine Implementierung von Actions sollten den folgenden typischen Ablauf gewährleisten:

- Wenn nötig Validierung des Status der aktuellen User-Session. Falls der Benutzer nicht eingeloggt ist erfolgt eine Weiterleitung zur Login-Seite
- Falls die Validierung eventueller Formulardaten durch die ActionForm-Bean nicht ausreicht, sollte das spätestens jetzt passieren.
- Ausführung der Applikationslogik durch Bean-Methodenaufrufe

- Aktualisieren verschiedener Beans, die das Verhalten im weiteren Verlauf der Anwendung beeinflussen, also insb. Beans mit Request-Scope und Session-Scope (siehe 3.4.)
- Rückgabe eines ActionForward-Objektes, das die anzuzeigende JSP identifiziert.

Zu beachten ist hier noch, dass nur eine Instanz der Action gebildet wird, welche von mehreren Threads (je einer pro Request) mit Eingang in der „execute()"-Methode benutzt werden. Das impliziert die Notwendigkeit von threadsicherem Code in der Action-Klasse. Insbesondere globale Instanzvariablen verbieten sich daher von vornherein.

3.3. View

Wie bereits erwähnt erfolgt die Realisierung der View mit JSPs. Struts unterstützt den Entwickler bei der Internationalisierung, bei Formularen sowie mit verschiedenen Erweiterungen von JSP, so z.B. eigene Tag-Libs oder dem Include-Statement.

3.3.1. Internationalisierung

Hypertext-Anwendungen im globalen World Wide Web sind zumeist internationalisiert, denn wer viele Besucher haben will, sollte sich nicht auf eine Nationalität beschränken. Auch die Wiederverwendbarkeit steigt mit zunehmender Internationalisierung. Java bietet uns hier bereits komfortable Hilfestellung:
Die „Locale"-Klasse repräsentiert eine Nationalität mitsamt seinen Eigenschaften, also z.B. Sprache, Datums- und Zahlenformate. Eine weitere wichtige Klasse ist „ResourceBundle", die wiederum Locale-spezifische Objekte enthält. Auf diese Art und Weise können u.a. Nachrichten in mehreren Sprachen unterstützt werden.

Der Struts-Anwendung können eigene ResourceBundles im Deployment-Descriptor (normalerweise die web.xml) angegeben werden:

```
<servlet>
   <servlet-name>action</servlet-name>
   <servlet-class>org.apache.struts.action.ActionServlet</servlet-class>
   <init-param>
      <param-name>application</param-name>
      <param-value>com.mycompany.mypackage.MyResources</param-value>
   </init-param>
 ...
</servlet>
```

3.3.2. Formulare und ActionForm-Beans

Bis XForms wirklich praktisch einzusetzen sind (mal abgesehen von serverseitigen Implementierungen), wird es wohl noch einige Zeit dauern. Bis dahin gibt es die Formularunterstützung von Struts.

Struts stellt eine eigene Tag-Lib für Formulare in JSP bereit. Ein einfaches Textfeld sieht folgendermaßen aus:

```
<html:text property="username"/>
```

Ich betrachte als nächstes ein Login-Formular, das Benutzernamen und Passwort abfragen soll. Hier zunächst der komplette JSP-Quelltext:

```
<%@ page language="java" %>

<!-- Definition des Tag-Library-Descriptors -->
<%@ taglib uri="/WEB-INF/struts-html.tld" prefix="html" %>
<%@ taglib uri="/WEB-INF/struts-bean.tld" prefix="bean" %>

<html:html>

<head>
<title>
   <bean:message key="logon.title"/>
</title>
</head>

<body bgcolor="white">
<html:errors/>
<html:form name="logonForm" action="logon.do"
      type="org.apache.struts.example.LogonForm">
<table border="0" width="100%">
   <tr>
      <th align="right">
         <html:message key="prompt.username"/>
      </th>
      <td align="left">
         <html:text property="username" size="16"/>
      </td>
   </tr>
   <tr>
      <th align="right">
         <html:message key="prompt.password"/>
      </th>
      <td align="left">
         <html:password property="password" size="16"/>
      </td>
   </tr>
   <tr>
      <td align="right">
         <html:submit>
            <bean:message key="button.submit"/>
         </html:submit>
      </td>
      <td align="right">
         <html:reset>
            <bean:message key="button.reset"/>
         </html:reset>
      </td>
   </tr>
</table>
</html:form>
</body>
</html:html>
```

Wie wir sehen können wird das „message“-Tag zur Internationalisierung der Formularbeschriftung genommen. „Error“-Tags zeigen eventuelle Fehlermeldungen an. Das „form“-Tag beinhaltet alle Formular-Elemente und stellt die Verbindung zu einem ActionForm-Bean (hier „logonForm“) her. Zur Initialisierung der Formularfelder werden die korrespondieren Eigenschaften der Bean verwendet.
Es wäre auch problemlos möglich, im Formular einen Datei-Upload zu realisieren. Man muss lediglich ein Feld mit dem Typ „file“ angeben und Struts erstellt ein MultiPart-Formular.

Erstellen wir jetzt noch die entsprechende ActionForm-Bean, welche die dem Formular zu Grunde liegende Datenstruktur definiert:

```
import org.apache.struts.action.ActionForm;

public class LogonForm extends ActionForm
{
   protected String username;
   protected String password;

   public void setUsername(String user)
   {
      username = text;
   }

   public String getUsername()
   {
      return username;
   }

   public void setPassword(String pwd)
   {
      password = pwd;
   }

   public String getPassword()
   {
      return password;
   }
}
```

Eine sinnvolle Anwendung dieser ActionForm-Bean liegt zudem in der Validierung von eingegebenen Formulardaten. Sie muss lediglich die Methode

```
public ActionErrors validate(ActionMapping mapping, HttpServletRequest request);
```

überschreiben, welche vom Controller aufgerufen wird bevor die eigentliche Action ausgeführt wird. Werden Fehler entdeckt kann eine Instanz der Klasse „ActionErrors“ mit genauen Informationen zum Fehler zurückgegeben werden.

3.3.3. Custom Tags

Neben der bereits in Struts enthaltenen Tag-Library ist es möglich, auch eigene Tags zu definieren. Darauf will ich aber nicht weiter eingehen.

3.3.4. Include

Include bezeichnet die Möglichkeit, während der Laufzeit, mehrere logisch getrennte „Seiten“ in eine andere Seite einzufügen. Auch hier sei auf den „Struts User’s Guide“ verwiesen.

3.4. Model

Das Model ist die der Applikation zu Grunde liegende Logik mitsamt den zu verarbeitenden Daten. Unterteilt wird es typischerweise in zwei Bereiche: Zustand und Funktionalität. Wesentliche Grundlage davon sind JavaBeans, die Einzelheiten des Zustandes repräsentieren.

Aufgrund der zentralen Rolle von Beans will ich noch auf ein paar web-spezifische Besonderheiten eingehen. Besonders zu betrachten wäre der Gültigkeitsbereich („Scope“), sprich die Lebensdauer von Beans. Aufgrund des besonderen Request/Response-Zyklus, der Hypertext-Anwendungen prägt, existieren für Beans verschiedene Gültigkeitsbereiche:

- PAGE:
  Die Bean gehört nur zu einer JSP-Seite während eines einzigen Requests.

- REQUEST:
  Die Bean kann zu mehreren Seiten gehören während eines einzigen Requests.
- SESSION:
  Session Tracking ermöglicht die Verfolgung mehrerer zusammenhängender Requests eines Clients. Die Bean existiert in diesem Zustand solange die Session aktiv ist.
- APPLICATION:
  Die Lebensdauer entspricht der Laufzeit der Applikation.

Zur Umsetzung der Persistenz dieser Beans über einen Request/Response-Zyklus hinaus bietet Struts verschiedene Container-Lösungen, in denen die Speicherung erfolgt.
Die Benutzung einer Bean (mit Gültigkeit „request“) in JSP sieht folgendermaßen aus:

```
<jsp:useBean id="cart" scope="request" class="com.mycompany.MyApp.MyCart"/>
```

3.4.1. ActionForm-Bean

Diese Beans sind ein Hauptbestandteil des Front-End-Modells, das die Datenbasis für die Views darstellt. Sie enthalten abgesendete Formulardaten und bilden so die Grundlage für Formularunterstützung in Struts. Sie haben darüberhinaus noch die Möglichkeit, Formulardaten zu verifizieren noch bevor die entsprechende Action ausgeführt wird. Details dazu wurden aber schon im Abschnitt zur View erläutert.

3.4.2. Systemzustands-Beans

Wie bereits erwähnt wird der aktuelle Zustand der Applikation in Beans gespeichert. Bei kleineren Anwendungen reichen wahrscheinlich JavaBeans, die transiente Daten repräsentieren oder mit bestimmten Datensätzen/-objekten aus Datenbanken assoziiert sind. Bei großen Enterprise-Anwendungen, deren Benutzerzahlen von vorneherein nicht begrenzt sind und sie evtl. noch hohe Anforderungen an Ausfallsicherheit und dergleichen stellen, geht wahrscheinlich kein Weg an EJB vorbei. Ob das eigene Projekt so skalierbar zu sein hat muss jeder selbst entscheiden. EJB stellen auf jeden Fall einen nicht unerheblichen Mehraufwand dar. Meistens reichen auch normale Beans, die auf ihre eigene Art mit einer Datenbank verbunden sind, völlig aus.

3.4.3. Applikations-Logik-Beans

Genau wie der Zustand sollten auch die funktionalen Aspekte mit Beans modelliert und implementiert werden. Man kann sie entweder zusammen mit dem Zustand erstellen oder eigene Beans mit nur funktionalen Aspekten entwerfen.
Beachten sollte man im Sinne der Wiederverwendbarkeit die Trennung zum restlichen Struts Framework, so ist zu vermeiden, Servlet- bzw. JSP-spezifische Klassen in irgendeiner Form zu verwenden. Stattdessen sollten Adapter zur Umwandlung der Datentypen bereitgestellt werden. Ein Beispiel wären hier zum Beispiel Request-Parameter aus der Klasse HttpServletRequest.

3.5. Fazit

Der Hauptteil von Struts besteht aus nicht viel mehr als ca. 10 Klassen, was Struts relativ übersichtlich wirken lässt. Für den Einsteiger ergeben sich nicht allzu große Hürden, sowohl beim Verständnis des Frameworks als auch bei der Implementierung einfacher Anwendungen. Dem Anwendungsentwickler werden außerdem viele Freiräume gelassen, d.h. er kann zum einen viele mit Struts gelieferte Klassen erweitern und zum anderen ist er nicht auf deren

Verwendung beschränkt. Das birgt aber auch die Gefahr einer Implementierung, die nicht im Sinne von Struts sein könnte, z.B. eingeschränkte Wiederverwendbarkeit oder aufgeweichte Trennung von Model, View und Controller. Entwickler sollten sich deshalb zunächst mit der Philosophie der von Struts verwirklichten „Model 2“-Architektur vertraut machen.
Ansonsten steht mit Struts ein durchdachtes und flexibles Framework zur Verfügung, das sich hinter kommerziellen Lösungen nicht verstecken muss.

4. XForms

Der Einsatz von Formularen in Webseiten ist heutzutage so selbstverständlich, dass interaktive Anwendungen in Hypertext-Appplikationen ohne sie undenkbar wären. Mit steigender Interaktivität steigt auch die Benutzung von Formularen. Sie bieten ein standardisiertes Interface zur Kommunikation zwischen dem Benutzer und dem dahinterliegenden Server. Nun gibt es Formulare bereits seit 1993, damals vom W3C[1] als integraler Bestandteil von HTML veröffentlicht und für damalige Anforderungen konzipiert. Jetzt, 10 Jahre später, zeigen sich die Nachteile dieses Standards. Als Endgeräte für Hypertext-Anwendungen kommen nicht mehr nur Browser auf PCs/Workstations zum Einsatz, sondern es verbreitet sich zunehmend auch die Web-Nutzung durch andere Geräte, wie z.B. PDAs, Set-Top-Boxen, Mobiltelefone usw.

Sehen wir uns ein herkömmliches HTML-Formular an:

```
<form name="SubscriptionForm" method="post"
  action="http://localhost:8080/swt/manager.UebManager?action=SubscribeStudent">
Matrikelnummer:
    <input type="text" name="login" maxlength="12" size="12">
Vorname:
    <input type="text" name="fname" size="30">
Name:
    <input type="text" name="name" size="30">
Semester:
    <input type="text" name="semester" size="30">
E-Mail:
    <input type="text" name="email" size="30">
<input type="submit" name="submit" value="Registrieren">
```

Wie wir sehen geht es um ein Formular zur Einschreibung eines Studenten in eine Vorlesung. Die Formulardaten werden an einen Server geschickt, der sie auswertet und die eingegebenen Werte überprüft. Das Problem hierbei ist offensichtlich, dass der Server genau wissen muss, was man ihm hier vorsetzt, um das ganze auswerten zu können, er kann nicht auf ein Datenmodell zurückgreifen, denn das wird ja im HTML-Quelltext definiert, zusammen mit der Deklaration der visuellen Formular-Ansicht. Ein weiteres Problem bekommen wir, wenn ein Benutzer z.B. im Feld "Semester" keinen numerischen Wert eingibt, auch diesen Fall muss der Server abfangen. Eine Alternative wäre Client-Scripting mit etablierten, in HTML eingebetteten Skriptsprachen, z.B. JavaScript, aber das endet damit, dass die HTML-Seite bald nicht mehr überschaubar und v.a. wartbar wird. Davon, dass solche Formulare kaum wiederverwendbar und in der angegebenen Deklaration womöglich auf anderen mobilen Clients gar nicht darstellbar wären, wollen wir noch nicht einmal reden. Was fehlt ist eine Trennung von Formular-Datenmodell und Darstellung.

4.1. Ziele

Diese Unzulänglichkeiten versuchen XForms als neue Generation von Formularen zu beseitigen. XForms 1.0 sind seit Oktober 2003 offizieller W3C-Standard, was bedeuten soll, dass die Spezifikation ausgereift und zum Einsatz im Web empfohlen wird. Die hauptsächlichen Vorteile sind dabei folgende:

- **Wiederverwendbarkeit**:
  Ein erstelltes Formular kann in weiteren Anwendungen problemlos wieder eingesetzt werden, z.B. obiges Einschreibungsformular in versch. Seminarverwaltungen, unabhängig ihrer eigentlichen Daten.

[1] http://www.w3c.org

- **Plattformunabhängigkeit**:
  Die visuelle Darstellung ist abstrakt, d.h. sie kann adaptiv an verschiedene Endgeräte angepasst werden, völlig transparent vom Datenmodell.
- **Universalität**:
  Durch die Trennung von Datenmodell und Darstellung sind die Formulare portabel zum Einsatz in anderen Anwendungen, die evtl. Formulare auf ihre eigene Art und Weise einsetzen, man denke hier nur an Spracherkennung usw. Das wird u.a. durch die Bereitstellung von Metadaten, z.B. Beschriftungen, erreicht.

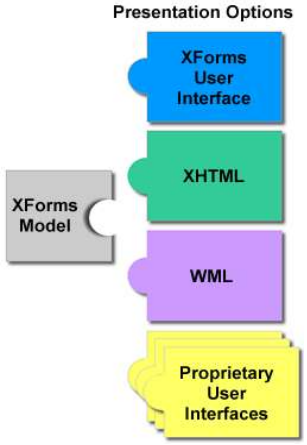

Abbildung 5: Ein Modell – mehrere Interfaces
Quelle: www.w3.org

XForms ist kein eigenständiger neuer Dokumententyp, sondern wird in andere Markup-Sprachen integriert, so z.B. als Bestandteil des zukünftigen Standards XHTML 2.0. Diese Version ist keine wirkliche Weiterentwicklung von XHTML 1.0 und 1.1, bei denen die Abwärtskompatibilität zu HTML, mehr oder weniger erfolgreich, im Vordergrund stand, sondern sie bricht mit der Tradition und ist nicht mehr kompatibel zu den älteren Standards. Wichtige Neurungen sind z.B. Modularität, striktere Trennung zw. Struktur und Darstellung und natürlich die konsequente Integration von XForms.
XForms ist eine XML-Anwendung, d.h. sie verwendet XML als Beschreibungssprache für das Datenmodell, die eigentlichen Daten und die visuelle Darstellung des Formulars. Letzteres basiert auf der Transformation mittels XSLT von XML in andere Formate, z.B. HTML, WML, VoiceXML. Der Vorteil von XML liegt in der Standardisierung, Wiederverwendbarkeit, Austauschbarkeit und der wesentlich leistungsfähigeren Datenrepräsentation als bei herkömmlichen Formularen, deren Daten „flach“ in „Variable/Wert“-Paaren gespeichert werden. Nicht zu verachten ist auch, dass XML einfach zu handhaben ist und selbst beschreibende Datenstrukturen bietet. Die Kommunikation zwischen Client und Server ist weiterhin nicht auf einen Request/Response-Zyklus beschränkt. Es ist möglich, dass ein Formular, das dem Absenden eines Beitrags in einem Forum dient, zwischendurch einen oder mehrere Moderatoren passieren muss, die entweder die Formulardaten bestätigen oder ändern können. Bei jeder Zwischenstation werden die Formulardaten erneut visualisiert und abgeschickt.

4.2. Architektur

Einen Überblick über XForms gibt folgende Abbildung:

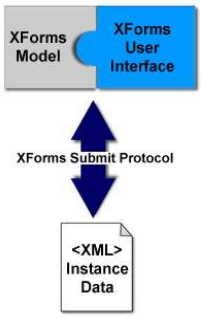

Abbildung 6,
Quelle: www.w3.org

Die drei Hauptbestandteile sind:

- **Model**:
  Beschreibt das dem Formular zu Grunde liegende Datenmodell, also alle vom Client zu erfragenden Daten und die Art und Weise ihrer Strukturierung. Es ist sogar möglich, Abhängigkeiten zw. versch. Daten zu modellieren.

- **User-Interface**:
  Erledigt die visuelle Darstellung des Formulars für den Endanwender. Je nach Endgerät werden verschiedene Oberflächen angeboten.

- **Instance-Data**:
  Repräsentieren die vom Client übermittelten Formulardaten serialisiert in XML. Sie besitzen dieselbe Struktur, die auch im Model definiert worden ist. Die Adressierung einzelner Elemente erfolgt per XPath, also mit Pfaden, die sich an der Notation herkömmlicher Dateisysteme orientieren. XPath stellt außerdem noch Methoden zur Stringmanipulation bereit und kann sogar Rechenoperationen ausführen.

4.3. Praxis

Da XForms ein noch sehr neuer Standard ist und noch kaum ein Browser ihn unterstützt, führt derzeit kein Weg an einer serverseitigen Implementierung vorbei. Eine solche ist z.B. Chiba[1], die auf Java-Servlets basiert und einen Großteil der XForms 1.0 Spezifikation unterstützt. Obwohl XForms wohl eher eine Client-Technologie ist, wird sie sich wahrscheinlich vorerst mehr als Serveranwendung etablieren. Es dürfte noch einige Zeit dauern, bis die etablierten Browser XForms unterstützen. Auf Chiba will ich allerdings auch nicht weiter eingehen, sondern als nächstes ein Beispiel zeigen, wie man theoretisch mit XForms ein einfaches Formular erstellt. Der Aufwand mag relativ hoch erscheinen, doch er amortisiert sich schnell bei komplexeren Formularen.

Betrachten wir wieder unser Einschreibungs-Formular, diesmal mit reduziertem Umfang:

[1] http://chiba.sourceforge.net

Unsere Daten (und das Verhalten des Formulars, dazu später mehr) definieren wir im XForms-Model. Wir benötigen den Namen des Studenten, seine Matrikel-Nr. und seinen angestrebten Abschluss. Das wird deklariert im *model*-Element von XForms und wäre in XHTML typischerweise im *head*-Abschnitt. Zwischen dem *xforms:instance*-Tag befindet sich das Skelett des Datenmodells, das wir abfragen wollen. Genau so wird auch später die *Instance Data* aussehen.

```
<xforms:model>
  <xforms:instance>
    <subscription>
      <name/>
      <matrikel/>
      <abschluss/>
    </subscription>
  </xforms:instance>
  <xforms:submission
     action="http://localhost:8080/swt/manager.UebManager?action=SubscribeStudent"
     method="post" id="submit"/>
</xforms:model>
```

Als nächstes definieren wir das Interface, das die Repräsentation für den Endbenutzer beschreibt, insbesondere die Metadaten, ohne die man wohl nicht wissen würde, was man denn wirklich ausfüllt.
Das ganze kommt bei XHTML typischerweise in den *body*-Abschnitt. Die Assoziation zw. Interface und Model geschieht hier mit einem XForms-eigenen "binding"-Mechanismus, in diesem Fall das *ref*-Attribut. Im Gegensatz zu unseren herkömmlichen Formularen befindet sich die Deklaration nicht in einem umfassenden *form*-Tag, stattdessen könnte man, wenn man mehrere Formulare auf einer Seite benötigt, die Model-Bezeichnung als Attribut angeben.
Zu beachten ist außerdem, dass diese Deklaration nicht zwingend beim Client auch Textfelder und Radio-Buttons erzeugt. Ein Browser wird das wohl so darstellen, aber ein Handy oder ein Client mit Spracheingabe könnte das eher anders handhaben.

```
<input ref="name">
  <label>Name</label>
</input>
<input ref="matrikel">
  <label>Matrikel-Nr.</label>
</input>
<select1 ref="abschluss">
  <label>Abschluss</label>
  <item>
    <label>Diplom</label>
    <value>diplom</value>
  </item>
  <item>
    <label>Master</label>
    <value>master</value>
  </item>
</select1>
<submit submission="submit">
  <label>Einschreiben</label>
</submit>
```

Damit ist das Formular einsatzbereit und kann eingesetzt werden. Ein XForms-Processor übernimmt die Ausführung und liefert sofort die serialisierten XML-Daten *(Instance Data)* an den Server:

```
<subscription>
  <name>Franz Matschenkow</name>
  <matrikel>1234567</matrikel>
  <abschluss>diplom</abschluss>
</subscription>
```

Zugegeben, das hätte man problemlos auch mit einem herkömmlichen Formular machen können, daher jetzt noch ein paar Features im Beispiel, die nur mit XForms möglich sind:

Es ist möglich, Namespaces und im Model (!) Standardwerte anzugeben:

```
...
    <subscription xmlns="de.myUniversity.myChair">
      <name/>
      <matrikel/>
      <abschluss>diplom</abschluss>
    </subscription>
...
```

Wir könnten auch mit Hilfe von XML Schema sicherstellen, dass die Matrikelnummer eine Zahl ist und 7 bis 8 Stellen besitzt:

```
  <xsd:schema ...>
    ...
    <xsd:simpleType name="matrikel">
      <xsd:restriction base="xsd:string">
        <xsd:pattern value="\d{7,8}"/>
      </xsd:restriction>
    </xsd:simpleType>
    ...
  </xsd:schema>
```

4.4. Fazit

XForms stellen ein leistungsfähiges Framework für die nächste Generation von Formularen bereit. Zur Zeit kommt man aber nicht an serverseitigen Lösungen, wie Chiba, vorbei. Bis sie wirklich herkömmliche Formulare ablösen können werden wohl noch Jahre vergehen. Das hängt nicht zuletzt von der Unterstützung durch den Internet Explorer und Netscape ab. Ein Problem sehe ich außerdem noch in einem nicht zu unterschätzenden Lern- und Zeitaufwand von Webmastern, denen es wahrscheinlich anfangs nicht leicht fallen wird, die Notwendigkeit von XForms zu verstehen, denn zugegeben, bis nicht genügend professionelle Tools zur Unterstützung zur Verfügung stehen, sind XForms, gerade bei kleinen bis mittleren Formulargrößen nicht so schnell geschrieben wie unsere derzeit etablierten „einfach gestrickten“ Formulare.

5. Weitere Apache Frameworks

5.1. Turbine

Turbine ist ebenfalls wie Struts seit 1999 ein Teil des Apache Jakarta Projekts. Die Entwickler nennen das zu Grunde liegende Modell „Model2+1“, da es, ähnlich Struts, einen hierarchischen Controller unterstützt. Das Framework stellt dem Entwickler in allen Teilen der MVC-Architektur Hilfen bereit, so u.a. ein Service-Framework (Fulcrum), das verteilte Java Caching System (JCS), ein Projektverwaltungs- und Dokumentationstool (Maven) sowie das objektrelationale Tool Torque. Besonders letzteres vereinfacht die Erstellung des Models indem es aus einem in XML spezifizierten Datenbankschema Teile der Applikationslogik erstellen kann.
Für die View kann man JSP verwenden. Die Turbine-Entwickler bevorzugen allerdings reine Templatelösungen wie Velocity, WebMacro oder Freemarker. Eine nennenswerte Besonderheit wäre noch die Möglichkeit, dass man für jedes View Template eine eigene Screen-Klasse implementieren kann, die verschiedene Daten aus dem Model zur Ansicht aufbereiten kann.
Wer schnell mit Turbine entwickeln will, dem wird außerdem das Turbine Development Kit (TDK) mit Sicherheit eine große Hilfe sein.

5.2. Jetspeed

Jetspeed als ein weiteres Jakarta Framework ist vor allem zur Entwicklung von Portalen ausgerichtet. Die Konzeption sieht vor, die Entwicklung so weit wie möglich ohne Programmierarbeit nur durch Konfiguration von Inhalt durchzuführen. Es basiert auf Turbine und verwendet somit grundlegende Konzepte davon. Eine Portalseite besteht aus sog. mehreren Portlets, die jeweils für sich eine Informationseinheit repräsentieren, z.B. Newsticker, Webmail usw. und flexibel auf den Webseiten eingesetzt werden können. Jetspeed stellt für Portlets auch transparente Funktionen der Personalisierung bereit, wodurch Besucher das Portal für eigene Bedürfnisse anpassen können.

5.3. Cocoon

Apache Cocoon ist ein sehr beliebtes Portal-Framework, das komplett auf der Konfiguration durch XML basiert. Eine besonders erwähnenswerte Eigenschaft von Cocoon ist die Sitemap, in der „Pipelines“ definiert werden können, die die Bearbeitung von Requests beeinflussen können. In ihnen werden verschiedene XML-Technologien hintereinander ausgeführt. Das führt dazu, dass Webseiten je nach Client völlig transparent unterschiedlich aufbereitet werden können. Zur Darstellung der View wird XSLT verwendet, das XML in verschiedene Ausgabeformate transformieren kann.

6. Zusammenfassung

Alle hier vorgestellten Frameworks unterstützen die Entwicklung von Hypertext-Anwendungen indem sie dem Entwickler verschiedene Modelle und wiederverwendbare Komponenten zur Verfügung stellen. Dabei zielen sie auf unterschiedlichste Ausprägungen solcher Anwendungen ab.
Struts stellt ein Basis-Framework für jegliche Art von Webapplikationen dar und bietet insbesondere Funktionen für die Präsentationsschicht, welche aus JSPs besteht, während das Modell nicht weiter spezifiziert ist.
Turbine konzentriert sich schon eher auf Anwendungen, die verstärkt Rollen spezifische Dienste anbieten und bringt dafür ein ausgereiftes Benutzer-Management mit. Zur Darstellung setzt Turbine auf Template-Lösungen, die zwar eine geringere Mächtigkeit als JSPs besitzen, aber mit eigenen Scriptsprachen wahrscheinlich für den Designer eher zu verstehen sind. Ausserdem sind aus dem Turbine-Projekt weitere, inzwischen eigenständige Entwicklungen hervorgegangen, wobei das wichtigste wohl der objektrelationale Mapper Torque darstellt.
Speziell für Portale existieren die beiden Frameworks Jetspeed und Cocoon, die dem Entwickler die Programmierung in Java größtenteils abnehmen wollen. Ersetzt wird das, speziell bei Cocoon, durch eine Konfiguration, die komplett auf XML-Technologien basiert. Beide Frameworks stellen weiterhin flexible und mächtige Funktionen zum Content-Management von Web-Portalen bereit.
Schließlich ist der vorgestellte W3C-Standard XForms weniger ein Framework als eine Spezifikation, die künftig von sog. XForms-Prozessoren umgesetzt werden wird. Genauso ist davon auszugehen, dass XForms irgendwann auch Einzug in die bereits vorgestellten Webapplikations-Frameworks Einzug finden wird.
Dieser Artikel hat sich auf Open Source Frameworks für J2EE beschränkt. Das soll nicht heißen, dass es nicht auch leistungsfähige kommerzielle Entwicklungen gibt. Zu nennen wären da u.a. die Agility Suite[1], Hawk[2], HyperQBS[3] oder die Swinglets[4]. Letztere versuchen ein Swing-ähnliches Komponenten-Modell für Webapplikationen bereitzustellen.
Was die ASP.NET Plattform angeht, so wird man in naher Zukunft sehen, ob diese eine der J2EE entsprechende Bedeutung bei den Webapplikationen erreichen kann.

[1] http://www.netdecisions.com
[2] http://www.xored.com/hawk.html
[3] http://www.hyperqbs.com
[4] http://www.swinglets.com

A. Anhang

A.1. Homepages

Struts:
http://jakarta.apache.org/struts

XForms:
http://www.w3.org/MarkUp/Forms

Turbine:
http://jakarta.apache.org/turbine

Jetspeed:
http://jakarta.apache.org/jetspeed

Cocoon:
http://cocoon.apache.org

A.2. Referenzen

[1] H. Balzert „Lehrbuch der Software-Technologie“, 2002

[2] A. Holub „Building user interfaces for object-oriented systems“, http://www.holub.com, 1999

[3] E. Armstrong, J. Ball, S. Bodoff et al. „The J2EE 1.4 Tutorial“ http://java.sun.com/j2ee/1.4/docs/tutorial/doc/index.html, 2003

[4] „The Struts User’s Guide“, http://jakarta.apache.org/struts/userGuide

[5] Micah Dubinko, "XForms Essentials", O'Reilly 2003

[6] M. Dubinko, L. Klotz, R. Merrik, T. Raman „W3C Candidate Recommendation“, http://www.w3.org/TR/2002/CR-xforms-20021112/