Fachhochschule Konstanz
Fachbereich Informatik

Einsatz des J2EE Frameworks Jakarta Struts

Diplomarbeit

Markus Herrmann

Zürich, 31. Januar 2003

Diplomarbeit

zur Erlangung des akademischen Grades

Diplom-Informatiker (FH)

an der

Fachhochschule Konstanz

Hochschule für Technik, Wirtschaft und Gestaltung

Fachbereich

Informatik / Wirtschaftsinformatik

Thema :	**Einsatz des J2EE Frameworks Jakarta Struts**
Diplomand:	Markus Herrmann Röschibachstr. 24, CH-8037 Zürich
Firma:	 Zürcher Kantonalbank, Neue Hard 9, CH-8010 Zürich
Betreuer:	Professor Dr. Eduard Klein, Fachhochschule Konstanz Dipl.-Ing. Bernd Reichert, Zürcher Kantonalbank
Abgabetermin:	31. Januar 2003

Zusammenfassung

Thema:	Einsatz des J2EE Frameworks Jakarta Struts
Diplomand:	Markus Herrmann
	Röschibachstr. 24
	CH-8037 Zürich
Firma:	Zürcher Kantonalbank
	Neue Hard 9
	CH-8010 Zürich
Betreuer:	Prof. Dr. Eduard Klein, Fachhochschule Konstanz
	Dipl. Ing. Bernd Reichert, Zürcher Kantonalbank
Abgabetermin:	31. Januar 2003
Schlagworte :	Jakarta Struts , MVC, J2EE, Framework, Java Server Page, Servlet, Tag Library ,Web-Applikation, IBM WebSphere Application Server, Tomcat

Die Arbeit befasst sich mit dem J2EE Framework Jakarta Struts. Hauptziel ist es, den Einsatz von Struts in der ZKB zu prüfen. Es soll festgestellt werden, ob Struts in Zukunft für die Entwicklung grösserer Web-Applikationen in der ZKB eingesetzt werden kann. Dazu wird eine bereits existierende ASP Web-Applikation zuerst zu einer JSP-Applikation (Model 1) und anschliessend zu einer Struts-Applikation (Model 2) portiert. Danach werden die beiden Versionen bzgl. Entwicklungsaufwand, Funktionsumfang, Performance und Wartbarkeit miteinander verglichen. Darüber hinaus werden Fähigkeiten des Struts Frameworks beleuchtet, die Architektur des Frameworks beschrieben und überprüft, inwiefern das Framework den Entwickler entlasten kann. Entwickler ohne Erfahrung mit Struts finden hier ausserdem eine verständliche Einführung an einem überschaubaren Beispiel.

Nach der Einleitung in Kapitel 1 werden im zweiten Kapitel die Struts zu Grunde liegenden Technologien der Java 2 Plattform Enterprise Edition (J2EE) von Sun beschrieben. Im dritten Kapitel wird detailliert auf Struts eingegangen. Um dem Leser den Einstieg in Struts zu erleichtern, wird vor der Beschreibung der Struts Komponenten die Architektur und der Programmablauf erläutert. Eine Anleitung zum Erstellen einer kleinen Struts Applikation erklärt Struts an einem praktischen Beispiel. Vergleichbare Frameworks werden am Ende des Kapitels vorgestellt. Im vierten Kapitel erläutere ich die Entwicklung der von mir mit Struts erstellten Web-Applikation. Die beiden letzten Kapitel enthalten die Erkenntnisse aus meiner Arbeit mit Struts und versuchen eine Entscheidungsgrundlage für oder gegen den Einsatz des Frameworks zu liefern.

Inhaltsverzeichnis

Vorwort

Ich möchte der Zürcher Kantonalbank für die Möglichkeit danken, meine Diplomarbeit in einem professionellen Umfeld erstellen zu können. Besonderer Dank gilt dabei Herrn Bernd Reichert für die gute Betreuung und Unterstützung sowie der ganzen Abteilung LIXA unter der Leitung von Herrn Dr. Walter Haas. Es war die Idee von Herrn Reichert, das Framework Struts in einer Diplomarbeit zu untersuchen.

Ebenfalls danken möchte ich Herrn Prof. Dr. E. Klein an der Fachhochschule in Konstanz für die kompetente Hilfe bei der Erstellung dieser Arbeit.

Ich möchte mich ausserdem bei Kerstin Winkler und meinen Eltern für das Korrekturlesen und die Anregungen bedanken.

Der ideale Leser dieser Diplomarbeit sollte ein gutes Verständnis von der Arbeitsweise des Internets und von Client-Server Architekturen haben. Java-, JSP-, HTML- sowie XML-Kenntnisse werden vorausgesetzt.

Abkürzungsverzeichnis

ADB	Abfrageorientierte Datenbasis (ZKB Data Warehouse)
API	Application Programming Interface
ASP	Active Server Pages
EIS	Enterprise Information Systems
EJB	Enterprise Java Bean
ERP	Enterprise Resource Planning
GUI	Graphical User Interface
IIS	Internet Information Server
J2EE	Java 2 Enterprise Edition
JAR	Java Archiv
JDBC	Java Database Connectivity
JMS	Java Message Service
JNDI	Java Naming and Directory Interface
JSF	Java Server Faces
JSP	Java Server Pages
JVM	Java Virtual Machine
MVC	Model-View-Controller
RDBMS	Relational Database Management System
SEU	Software Entwicklungsumgebung in der ZKB
WAR	Java Archiv für Web-Applikationen
WAS	IBM WebSphere Application Server
WML	Wireless Markup Language
WSAD	IBM WebSphere Application Developer Studio
XML	Extensible Markup Language
ZKB	Zürcher Kantonalbank

Darstellungsverzeichnis

Abbildungen

Quelltexte

1 Einleitung

Diese Diplomarbeit entstand während meiner sechsmonatigen Tätigkeit in der Systemarchitektur (SAR) der Zürcher Kantonalbank (ZKB) in Zürich. Die ZKB ist mit rund 4300 Mitarbeitern nach der UBS und der Credit Suisse Group die drittgrösste Bank in der Schweiz und zugleich grösste Kantonalbank und führende Bank im Zürcher Wirtschaftsraum. Mit Depotvermögen und Kundengeldern von über CHF 100 Mrd. gehört die ZKB zu den grössten Vermögensverwaltern der Schweiz.

Wie für alle Banken besitzt die Informatik auch für die ZKB einen hohen Stellenwert. Zur Zeit arbeiten ca. 560 Mitarbeiter im Informatikbereich. Die ZKB bezeichnet sich, was die Strategie in der Informatik angeht, als "smart follower" und nicht als "early adaptor". Das bedeutet, dass die Bank nicht jede neue Technologie sofort einsetzt, sondern zuerst den Markt beobachtet um zu prüfen was zum Standard wird. Das Ziel der Bank ist es einen möglichst grossen Teil der eingesetzten Software durch Standardsoftware extern zu beziehen und so wenig wie möglich selbst zu entwickeln. Aufgaben der Informatik in der ZKB sind u.a. Aufbau, Unterhalt und Weiterentwicklung der Rechenzentren, Netzwerke und Arbeitsplätze, sowie der produktiven Banksysteme. Dazu gehören beispielsweise die Online Bank, Kundeninformationssysteme, Stammdatenverwaltung, Vertriebssysteme, Kundenbeziehungs-management, Kartenverwaltungssysteme und Geldausgabe- und Einzahlautomaten.

Eine wichtige Rolle spielt dabei die Systemarchitektur, die mit fünf erfahrenen Informatikern und Ingenieuren Richtlinien und Grundsätze für den Einsatz von Soft- und Hardware erarbeitet und deren Einhaltung überwacht. Die SAR erarbeitet den technischen Bauplan der Informatiksysteme. Sie beschreibt das Zusammenwirken der bestehenden und neuen Technologien und erstellt Grundsätze betreffend der Bauweise und dem Einsatz von bestehenden und neuen Technologien und Systemen, sowie der Integration von Systemen. Ziele der SAR sind die Einschränkung der Lösungsvielfalt (beherrschbare Komplexität), kürzere Entwicklungszeiten (time to market), niedrigere Kosten durch Wiederverwendung bestehender Infrastrukturen, die Vereinheitlichung der Infrastruktur (Basis für den effizienten Betrieb), die Verbesserung des Kosten-Nutzen Verhältnisses und die Unterstützung der Business Strategie. In diesem Zusammenhang werden von der SAR auch neue Technologien, die für die Bank relevant sein könnten, untersucht und diesbezüglich erarbeitete Grundsätze verabschiedet.[1]

[1] vgl. [Busc02]

1.1 Ausgangslage

Oft wird die Wiederverwendung von Software als erstrebenswertes Ziel und erfolgversprechendes Instrument bei der Softwareentwicklung angesehen. Frameworks verfolgen genau diesen Aspekt, denn sie bestehen "*aus einer Menge von zusammenarbeitenden Klassen, die einen wiederverwendbaren Entwurf für eine bestimmte Klasse von Software darstellen*".[1] Neben der einfachen Wiederverwendung von Code dienen Frameworks auch der Wiederverwendung von Design und sogar kompletter Services. Sie stellen branchenspezifische oder -übergreifende Lösungen für allgemein bekannte Probleme dar. Dabei beschränken sie sich nicht wie die Entwurfsmuster auf reine Umsetzungsempfehlungen, sondern liefern fertige Implementierungen für ein bestimmtes Anwendungsgebiet.
Frameworks bestehen aus fertigen und halbfertigen Teilsystemen. Auf der einen Seite wird also die grundlegende Architektur für ein Softwaresystem bestimmt, auf der anderen Seite aber auch genug Raum für individuelle Erweiterungen gelassen.

In dieser Arbeit wird das Web-Framework Jakarta Struts - ein Open-Source Framework, welches unter dem Dach der Apache Software Foundation entstanden ist - betrachtet. Das Framework soll das Erstellen grösserer Web-Applikationen unter Einsatz von J2EE-Technologien vereinfachen. Struts gehört zur Gruppe der Web-Frameworks, die eine Architektur für Client-Server Anwendungen liefern. Der client-seitige Zugriff erfolgt dabei i.d.R. durch einen Webbrowser. Besondere Bedeutung hat dabei das Model-View-Controller Entwurfsmuster erlangt.

ADB-Info[2], ein Browser basiertes Werkzeug, welches Informationen über aktuell laufende Prozesse im Data Warehouse der ZKB liefert, dient in dieser Diplomarbeit als Anschauungsbeispiel. Diese Applikation lag als ASP-Lösung vor und wird im Rahmen der Diplomarbeit zuerst in eine reine JSP-Applikation (Model 1) und danach in eine Struts-Applikation (Model 2) portiert. Anhand eines Vergleichs dieser beiden Versionen können später Argumente für und gegen den Einsatz von Struts gefunden werden.

1.2 Ziel

Das Ziel dieser Diplomarbeit ist es, einen Überblick über Struts zu geben und eine Entscheidungsgrundlage für den Einsatz von Struts zu liefern. Die beiden Versionen von ADB-Info werden bzgl. Entwicklungsaufwand, Funktionsumfang, Performance und Wartbarkeit verglichen. Hat sich der Entwickler für Struts entschieden, steht ihm die in Struts implementierte Applikation ADB-Info als Beispiel zur Verfügung, welches ihm das Erlernen des Frameworks erleichtert.

[1] [Gamm01] S. 31

[2] Die Abkürzung ADB steht für "Abfrageorientierte Datenbasis" und verkörpert das Data Warehouse der ZKB

2 Einführung und Grundbegriffe

2.1 J2EE – Java 2 Plattform Enterprise Edition

Dieses Kapitel gibt einen kurzen Überblick über J2EE und die dazugehörigen APIs. Detailliertere Informationen gibt es u.a. in [@J2EE] und in unzähliger weiterer Literatur zu diesem Thema.

Die Java 2 Plattform Enterprise Edition definiert einen Standard zur Implementierung, Konfiguration, Verteilung und zum Einsatz von unternehmensweiten Anwendungen. J2EE ist kein Produkt im eigentlichen Sinne, sondern definiert einen allgemeinen Rahmen zur Erstellung unternehmensweiter Anwendungen basierend auf einem Komponentenmodell und der Programmiersprache Java.

J2EE ermöglicht die Realisierung unternehmensweiter Anwendungen durch eine Drei-Schichten-Architektur. Folgende Abbildung veranschaulicht die Aufteilung in die drei Schichten.

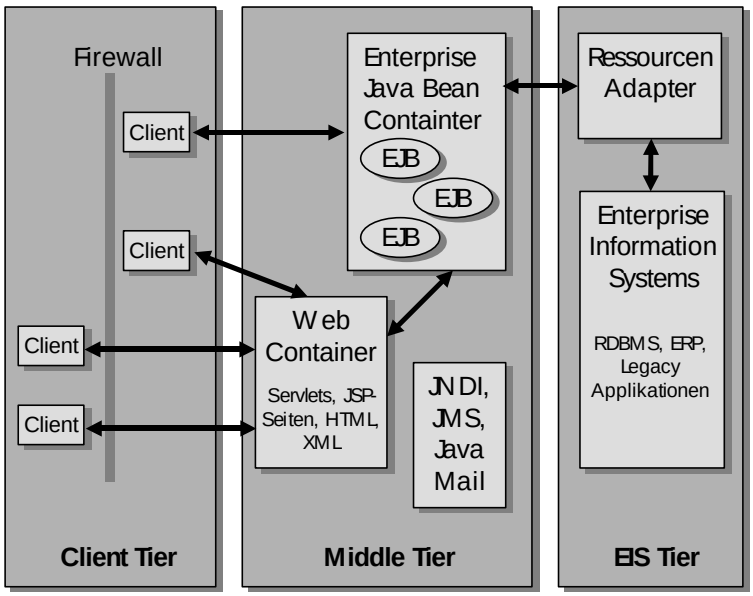

Abbildung 2-1: Die J2EE Plattform

Die mittlere Schicht (Middle Tier) wird oft auch als sogenannte Middleware bezeichnet. Im Vergleich zu anderen Middleware-Techniken wie Transaktionsmonitoren, Message-Queuing-Systemen und Object Request Brokern (ORB) erhebt J2EE den Anspruch, konsequent auf einem Komponentenmodell zu basieren und portabel zu sein. Der Kern des J2EE-Modells besteht darin, einfach anpassbare und leicht zu administrierende Komponenten zu entwickeln,

welche plattformübergreifend eingesetzt und an existierende Informationssysteme angekoppelt werden können.[1]

Die Laufzeitumgebung für eine J2EE-Anwendung wird durch sogenannte Container umgesetzt. Diese befinden sich auch in der mittleren Schicht. Die J2EE-Plattform definiert Dienste, welche durch J2EE-Container angeboten werden und allen Komponenten zur Verfügung stehen:

- Namensdienst
- Transaktionsdienst
- Sicherheitsdienst
- Konfigurationsdienst

Des weiteren müssen die Hersteller der J2EE-Container folgende Implementierungen der J2EE-APIs zur Verfügung stellen:

- **JDBC Extension**
  Diese API ist eine Erweiterung der Standard JDBC API und vereinbart Klassen und Interfaces für Connection-Pools, verteilte Transaktionen und änderbare Ergebnismengen von SQL-Anfragen.
- **JTA**
  Die Java Transaction API definiert eine Schnittstelle zur Interaktion mit einem Transaktionsmanager. Dieser regelt den Zugriff von Anwendungen auf gemeinsam genutzte Ressourcen wie Datenbanken oder Message-Systeme.
- **JNDI**
  Das Java Naming und Directory Interface ermöglicht den Zugriff auf Namens- und Verzeichnisdienste durch Java-Programme. Hierzu zählen neben dem J2EE-Namensdienst auch Verzeichnisdienste wie LDAP.
- **Servlet**
  Die Servlet API stellt Klassen und Interfaces zur Erstellung serverseitiger Web-Anwendungen zur Verfügung. Es werden sitzungsorientierte Anwendungen und eine gepufferte Ausgabe unterstützt.
- **JSP**
  Java Server Pages sind eine serverseitige Skriptsprache zur Erstellung von Web-Anwendungen auf der Basis von Servlets. Neben benutzerdefinierten Tag-Bibliotheken ist die enge Intergration von Java Beans ein wichtiges Merkmal. Auf die JSP Technologie wird später noch genauer eingegangen. JSPs spielen beim Einsatz von Jakarta Struts eine zentrale Rolle.

[1] vgl. [Tura01]

- **EJB**
  Enterprise Java Beans sind Softwarekomponenten, die den Zugriff auf transaktionsbasierte Dienste erlauben. Die Kommunikation mit CORBA-Objekten wird unterstützt.
- **RMI-IIOP**
  Remote Method Invocation ermöglicht die Kommunikation zwischen Java-Objekten in verschiedenen virtuellen Maschinen. RMI-IIOP ist eine Implementierung des RMI-Protokolls auf Basis des Corba Internet Inter ORB Protocol (IIOP) und ermöglicht die Integration von CORBA Objekten in J2EE-Anwendungen.
- **JMS**
  Java Message Service ist eine API zum Zugriff auf Message-Queuing-Systeme wie IBM MQ Series.
- **JCA**
  J2EE Connector Architecture ermöglicht die Integration von non-Java Anwendungen.
- **JavaMail**
  Die *Mail API* definiert Schnittstellen, mit denen E-Mails abgerufen oder verschickt werden können.
- **JAXP**
  Die *Java API for XML Parsing* definiert den Umgang mit XML-Parsern. Dadurch wird die J2EE-Platform unabhängig von bestimmten Parsern.

Container stellen eine einheitliche Sicht auf die von tiefer liegenden Schichten angebotenen Dienste zur Verfügung. Eine Komponente kann unabhängig von der verwendeten J2EE-Plattform die gleichen Dienste erwarten. Hierzu zählt auch die Verwaltung des Lebenszyklus von Komponenten. Alle Container sind dafür verantwortlich, die in der Konfigurationsdatei im XML-Format enthaltenen Einstellungen umzusetzen, beispielsweise die Authentifizierung von Benutzern.

Die Ankopplung existierender Informationssysteme (Enterprise Information Systems bzw. EIS) wie betriebliche Standardsoftware, Datenbanken oder Host basierte Systeme geschieht mit der Connector-Architektur. Diese basiert darauf, dass Hersteller von EIS Adapter zur Verfügung stellen. Diese implementieren die Java-Schnittstelle Common Client Interface (CCI) über die Komponenten auf die angebotenen Dienste zugreifen können.

Ausserdem bietet J2EE ein Rollenmodell, welches die einzelnen Rollen bei der Entwicklung von J2EE-Anwendungen definiert und die Entwicklung in grossen Teams oder das Zusammenspiel mehrerer Teams erleichtert.[1]

[1] vgl. [Tura01]

2.2 Java Server Pages und Servlets

Die Bedeutung von Web-Applikationen nimmt stetig zu. Web-Browser sind auf dem Weg das universelle Front-End für viele Anwendungen zu werden. Die J2EE-Plattform unterstützt zwei alternative Techniken zur dynamischen Generierung von HTML-Dokumenten[1]: Servlets und Java Server Pages (JSP). Beide Techniken stützen sich auf Java-APIs, wobei Servlets einen prozeduralen und JSPs einen deklarativen Ansatz verfolgen. Beide Techniken werden den Web-Komponenten zugeordnet. Sie haben Zugriff auf den J2EE-Namensdienst und können mit Enterprise-Java-Beans (EJB) kommunizieren. Die Konfiguration erfolgt mit Deployment-Deskriptoren.[2]

Der deklarative Stil von JSP ermöglicht eine bessere Trennung von Design und Anwendungslogik bei der Generierung eines Dokumentes. Dies wird durch die enge Integration von Java Beans mit Tags und durch benutzerdefinierte Tag-Bibliotheken erreicht. Java Server Pages entstanden erst, als Servlets schon lange im Einsatz waren. Grundsätzlich kann man aber sagen, dass der Funktionsumfang von JSPs und Servlets nahezu gleich ist. Alles was mit Servlets implementiert werden kann, kann auch mit Einsatz von JSPs implementiert werden. Dies wird umso klarer, wenn man die Tatsache betrachtet, dass JSPs vor dem ersten Ausführen oder nach Änderungen zuerst in Servlets übersetzt und dann ausgeführt werden.

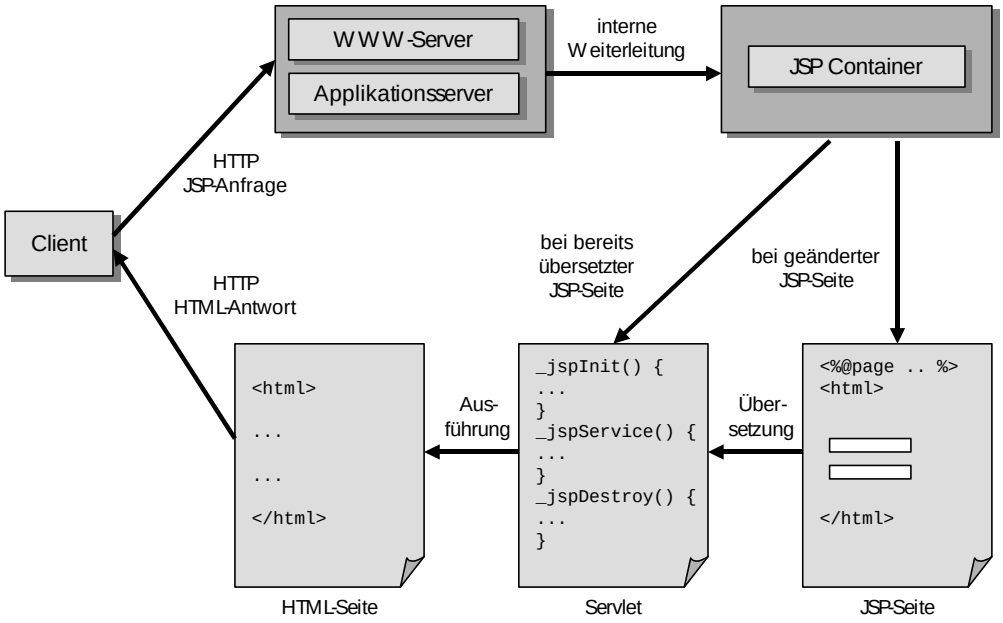

Abbildung 2-2: Anfrage einer JSP-Seite

[1] Grundsätzlich kann jegliche Art von Information (z.B. XML- oder WML-Dokumente) erzeugt werden. Da bei Web-Applikationen der Client i.d.R. ein Browser ist, spreche ich hier von HTML-Dokumenten.

[2] vgl. [Tura01]

Der grosse Nachteil von Servlets besteht in der Art der Generierung von Dokumenten. So muss beispielsweise jede einzelne Zeile eines HTML-Dokuments per *println()*-Befehl aus dem Java-Quellcode heraus generiert werden. Das macht es für Web-Designer extrem schwer die Benutzeroberfläche zu entwickeln. In Java Server Pages können HTML-Tags und Java-Code gemischt werden – JSPs sind HTML-Seiten, in die Java Code eingebettet ist. So können Web-Designer und Java-Entwickler leichter am gleichen Dokument arbeiten. JSP-Quellcode kann in jedem beliebigen HTML-Editor bearbeitet werden, weil er nur aus Tags besteht. [1]

Java Server Pages sind also ideal dazu geeignet textuelle Inhalte zu erzeugen und werden hauptsächlich dazu benutzt die Benutzerschnittstelle zu präsentieren und Benutzereingaben zu überprüfen. Sie bilden die Schnittstelle zwischen Client und dem Rest der Web-Applikation.

Servlets sind entweder Geschäftsdaten- oder Web-bezogen. Allgemein formuliert ist ein Servlet immer dann Web-bezogen, wenn ein Client es direkt anfragen kann. Einsatzgebiet Web-bezogener Servlets kann wie bei JSPs die dynamische Erzeugung von Inhalten[2] oder die Schaffung eines zentralen Zugangs sein. Geschäftsdatenbezogene Servlets implementieren Geschäftsprozesse und die Geschäftslogik.

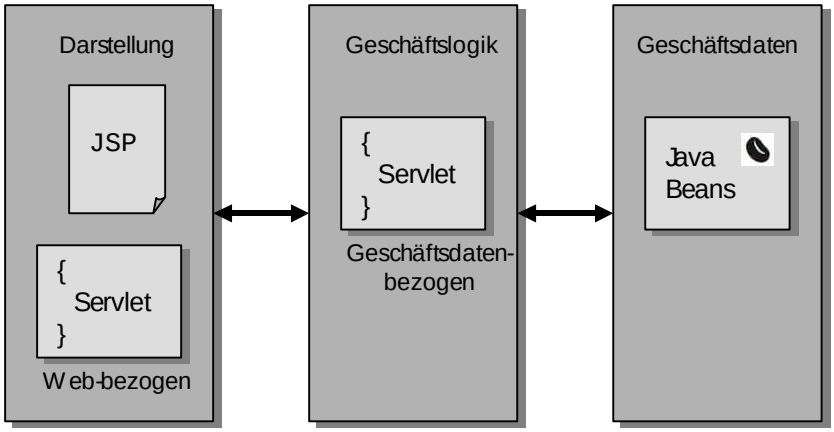

Abbildung 2-3: Das Zusammenspiel der Komponenten

An dieser Stelle sollten auch Java Beans erwähnt werden. Mit Java Beans hat der Entwickler die Möglichkeit Darstellungs- und Verarbeitungslogik voneinander zu trennen. Java Beans sind Softwarekomponenten, die als Java Klassen realisiert sind. Sie repräsentieren die Geschäftsdaten und können zusätzlich die Verarbeitungslogik beinhalten. Beans definieren eine Menge von Properties. Ein Property repräsentiert eine Objekt-Eigenschaft. Es lässt sich über Accessor-Methoden (Getter/Setter) abfragen und ändern. Die drei Komponenten JSP, Servlets und Java Beans ermöglichen die Erstellung von umfangreichen Web-Applikationen

[1] vgl. [Will01]

[2] Im Gegensatz zu JSPs können Servlets auch binäre Informationen erzeugen. Selbst das Erzeugen eines eigenentwickelten MIME-Typs ist vorstellbar, falls der Client in der Lage ist, ihn zu verarbeiten.

auf Grundlage der J2EE-Plattform. Mit Enterprise Java Beans können diese Möglichkeiten noch erweitert werden. Auf EJBs wird im Rahmen dieser Arbeit nicht eingegangen.

2.3 Architekturen

Ein wichtiger Aspekt beim Entwickeln von Web-Applikationen mit der J2EE-Plattform ist die Wahl der vorhandenen Komponenten-Technologien und die Aufteilung der Applikation auf die verschiedenen Komponenten. Im Laufe der Zeit haben sich zwei Architekturen herauskristallisiert, die in diesem Kapitel genauer vorgestellt werden. Die Model 1 Architektur ist vergleichbar mit unzähligen anderen serverseitigen Skriptsprachen wie Microsoft ASP, PHP oder auch CGI. Die Model 2 Architektur ist eine Weiterentwicklung der Model 1 Architektur und versucht das MVC-Paradigma umzusetzen.

2.3.1 Das MVC-Paradigma

Das Model-View-Controller-Paradigma wird nicht erst seit J2EE bei der Konstruktion von Applikationen mit grafischen Benutzerschnittstellen verwendet. Schon bei der Programmierung mit Smalltalk wurde mit dem MVC-Paradigma gearbeitet. In [Gamm01] wird die Aufteilung einer Applikation mit Benutzeroberfläche in die Model- (Datenbereitstellung), View- (Präsentation) und Controller- (Ablaufsteuerung) Objekte beschrieben. Ein View-Objekt muss sicherstellen, dass seine Darstellung den Zustand des Model-Objekts wiedergibt. Das Model benachrichtigt die von ihm abhängigen Views, wenn sich seine Daten ändern und der Controller ändert abhängig vom Benutzerverhalten das Model oder ruft andere Views auf. Wichtig dabei ist, dass ein Model keine Kenntnis von den View- und Controller-Komponenten haben muss und die Bestandteile somit separat entwickelt werden können.

Die folgende Abbildung zeigt die Kommunikation und den Datenaustausch zwischen den Teilnehmern einer MVC-Anwendung

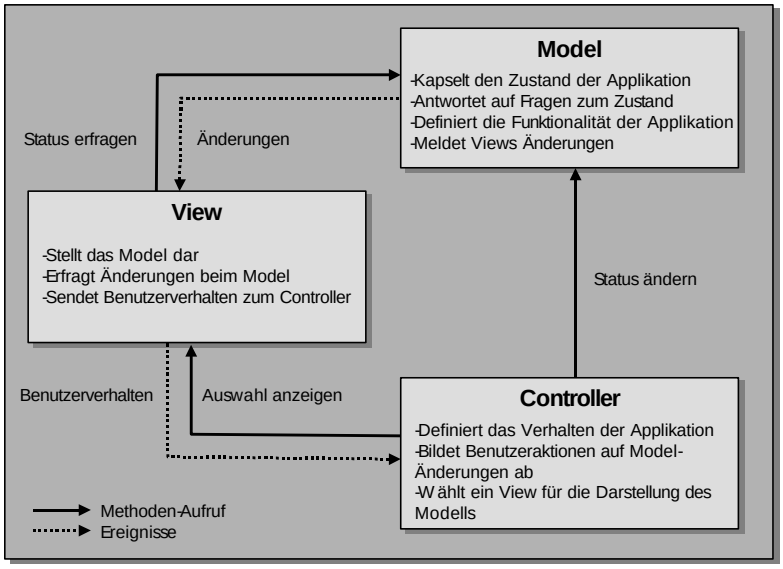

Abbildung 2-4: Das MVC-Paradigma

Bei einer J2EE-Applikation mit einem Web-Front-End werden die Views durch JSP-Seiten realisiert. Zur Visualisierung des Zustandes stellt ein View-Objekt Anfragen an das Model-Objekt. Alternativ kann auch das Model-Objekt bei Änderungen des Zustandes das View-Objekt benachrichtigen. Eine zusätzliche Aufgabe des View-Objektes besteht in der Entgegennahme von Benutzeranweisungen. Diese werden an das Controller-Objekt weitergeleitet, welches bei J2EE oft ein Servlet ist. JSPs verwenden hierzu HTML-Formulare. Die zugehörigen HTTP-Anfragen (GET oder POST) richten sich an das Controller-Objekt. Dieses verwendet die Methoden des Model-Objektes, um den internen Zustand zu ändern. Anschliessend wird das View-Objekt benachrichtigt. Das Model wird in einfachen Applikationen mittels Java Beans umgesetzt. Bei komplexen Applikationen können auch Enterprise Java Beans verwendet werden.[1]

2.3.2 Model 1 Architektur

Die Model 1 Architektur gibt es in zwei verschiedenen Varianten. Bei der ersten Variante werden die Beschaffung der Daten, die Anwendung der Geschäftslogik und die Darstellung des erzeugten Inhalts von nur einer JSP-Seite vorgenommen. Vorteil dieses Ansatzes ist die einfache Implementierung. Dieser Ansatz eignet sich für kleine und überschaubare Applikationen. Der Nachteil ist die starke Vermischung von HTML- und Java-Code und der insgesamt unübersichtliche JSP-Quellcode. Diese Variante ist ausserdem schlecht erweiterbar und skalierbar.

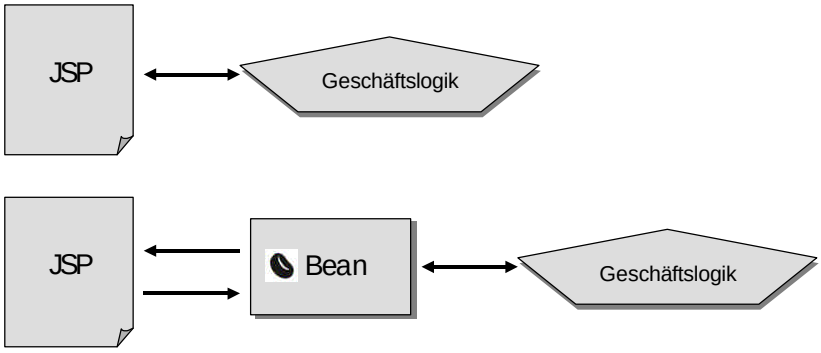

Abbildung 2-5: Die beiden Varianten des Model 1

Die erste Variante kann durch Benutzen von Java Beans erweitert werden. Das Prinzip der 2-Schichten-Architektur bleibt bestehen, allerdings kann ein Grossteil der Geschäftslogik sowie der Datenzugriffslogik und damit des Java-Codes in Beans ausgelagert werden. Vorteil dieses Ansatzes ist die Trennung von HTML- und Java-Code. Jetzt können Java-Entwickler und Web-Designer an verschiedenen Komponenten arbeiten. Durch die Auslagerung des Java-Codes wird die Wiederverwertbarkeit der Komponenten ermöglicht. Allerdings wird die Skalierbarkeit und Erweiterbarkeit dadurch nicht wesentlich verbessert.

[1] vgl. [Tura01] und [Gamm01]

2.3.3 Model 2 Architektur

Mit dem Einsatz der Model 1 Architektur ist die Entwicklung von MVC-Applikationen nicht möglich. Die Model 2 Architektur erweitert deshalb die Architektur um eine Controller-Komponente. Diese ist für die Vermittlung von Anfragen an die entsprechenden JSP-Seiten und die Bereitstellung des Models verantwortlich. Während in Model 1 eine 2-Schichten-Architektur vorliegt, ist Model 2 eine 3-Schichten-Architektur, die leicht um weitere Schichten (z.B. EJBs) zu einer N-Schichten-Architektur erweitert werden kann.

Auch die Model 2 Architektur kann in zwei Varianten unterteilt werden. In der ersten Variante verteilt das Vermittler-Servlet die Anfragen an die JSP-Seiten, welche sich das Datenmodell selbst beschaffen. Dazu kann die Geschäftslogik direkt in die JSP-Seite eingefügt oder in die Worker Beans ausgelagert werden. Vorteil ist die Schaffung eines zentralen Zugangs. Dieser kann z.B. für die Authentifizierung und Autorisierung oder eine zentrale Protokollierung verwendet werden. Hierbei wird eine klare Trennung zwischen Darstellung und Geschäftslogik erreicht. Die Applikation ist besser skalierbar und erweiterbar als Model 1 Applikationen. Die JSP/Worker-Bean-Kombinationen sind wiederverwendbar.

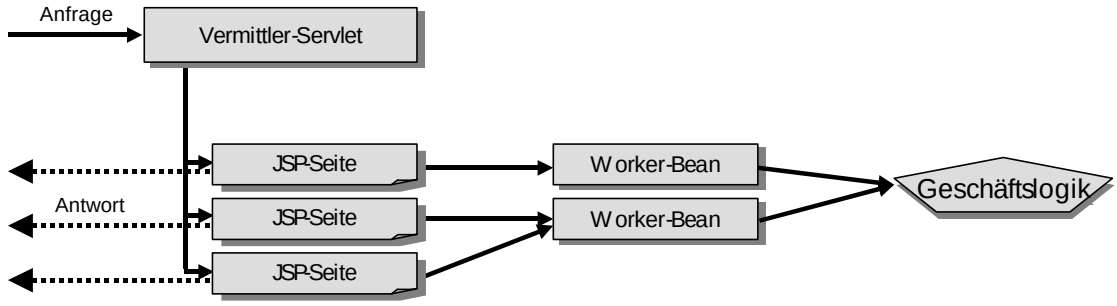

Abbildung 2-6: Ablauf einer Anfrage beim Model 2 (erste Variante)

In der zweiten Variante delegiert das Vermittler-Servlet die Anfrage nicht mehr an eine JSP-Seite, sondern direkt an eine Worker-Bean – also das Model. Diese ist dafür verantwortlich das Datenmodell zu erzeugen. Nachdem das Datenmodell erzeugt wurde, kann das Vermittler-Servlet die Anfrage an eine für die Präsentation geeignete JSP-Seite weiterleiten, die dann die Darstellung übernimmt. Hier ist das Vermittler-Servlet also wirklich eine Art Vermittlungsstelle und nicht nur ein Verteiler. Dadurch wird jegliche Kontrollfunktion aus dem JSPs in das Vermittler-Servlet ausgelagert und somit eine saubere Umsetzung des MVC-Paradigma erreicht.

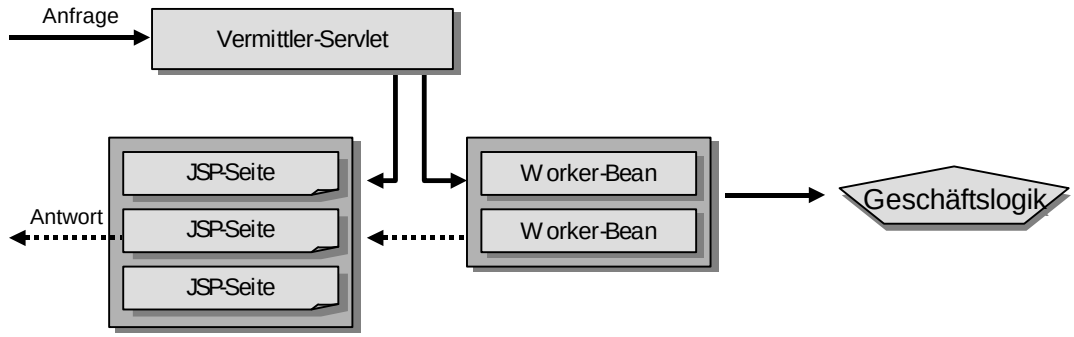

Abbildung 2-7: Ablauf einer Anfrage beim Model 2 (zweite Variante)

Da das Vermittler-Servlet bzw. der Controller bei umfangreichen Applikationen sehr komplex wird, können zusätzlich sogenannte Vermittler-Beans zwischen Vermittler-Servlet und Worker-Beans eingefügt werden. Mit ihnen kann eine Aufteilung der Kontrollkomponente realisiert werden. Erst mit dieser Architektur wird die Umsetzung grosser E-Commerce und E-Business Anwendungen möglich. Das MVC-Konzept ist konsequent umgesetzt. Die Applikation ist durch die starke Modularität gut skalierbar und erweiterbar. Die einzelnen Komponenten sind gut wiederverwendbar. Bei der Entwicklung ist eine klare Rollenverteilung möglich. Nachteil ist der hohe Arbeitsaufwand bei der Implementierung und in der Planungsphase. Die Anzahl der möglichen Fehlerquellen wird im Vergleich zur Model 1 Architektur stark erhöht.[1]

2.3.4 Kommunikation der Komponenten

Wenn die ganze Zeit von einer Aufteilung der Applikation gesprochen wird, stellt sich zwangsläufig die Frage wie die einzelnen Komponenten miteinander kommunizieren. Im Folgenden wird die Kommunikation zwischen JSPs, Java Beans und Servlets betrachtet. Das Wissen über die Kommunikation der Komponenten ist auch wichtig für das Verständnis des Frameworks Jakarta Struts.

Prinzipiell gibt es zwei Möglichkeiten der Kommunikation von Web-Komponenten: der direkte Datentransfer zwischen JSP und Servlet und der indirekte Datentransfer über eine Bean. Während die Web-Komponenten JSP und Servlet durch Weiterleiten (`include` und `forward`) in der Lage sind Daten aktiv zu transferieren, sind Beans eher ein Datencontainer. Sie sind darauf angewiesen, dass JSPs und Servlets die Daten abholen.

In diesem Zusammenhang muss der Begriff des Gültigkeitsbereiches (Scope) eines Datencontainers erklärt werden. Eine wichtige Rolle spielt dabei das `pageContext`-Objekt, das in jeder JSP Seite enthalten ist. Dieses Objekt bietet Methoden an, mit denen sich Daten in einen bestimmten Gültigkeitsbereich einpflegen und wieder auslesen lassen. Da JSP-Seiten nach der Übersetzung vom Container zu Servlets werden, ist es hilfreich zu wissen, welche

[1] vgl. [Tura01] und [Will01]

Möglichkeiten der Datenspeicherung in einem Servlet existieren. Damit lässt sich leichter nachvollziehen, welcher Gültigkeitsbereich in einer JSP-Seite durch welchen Datencontainer in einem Servlet repräsentiert wird. Nachfolgend sind die verschiedenen Objekte aufgelistet, die sich zur Datenaufnahme eignen. In Klammern ist der entsprechende Gültigkeitsbereich in einer JSP-Seite angegeben.[1]

- **Lokale Variablen der Methode `service()` (page)**
  Eignet sich nicht zum Datentransfer zwischen Web-Komponenten.
- **Das Request-Objekt (`HttpServletRequest`) (request)**
  Die Sichtbarkeit der Daten erstreckt sich über eine ganze Anfrage.
- **Das Session-Objekt (`HttpSession`) (session)**
  Die Sichtbarkeit der Daten erstreckt sich über einzelne Anfragen hinaus auf die ganze Session. Der Zugriff geschieht mit der Methode `getSession()` des `HttpServletRequest` -Objekts.
- **Das Context-Objekt (`ServletContext`) (application)**
  Die Sichtbarkeit der Daten erstreckt sich über die gesamte Web-Applikation und alle ihre Sessions. Der Zugriff geschieht mit der Methode `getServletContext()` des `ServletContext` -Objekts.

Nachfolgend werden die Möglichkeiten der Kommunikation zwischen Web-Komponenten aufgezeigt. Die folgenden Abschnitte erklären die Kommunikation kurz anhand von Code-Beispielen. Es wird in allen vier Beispielen davon ausgegangen, dass die Daten innerhalb einer Session transferiert werden (scope = session).

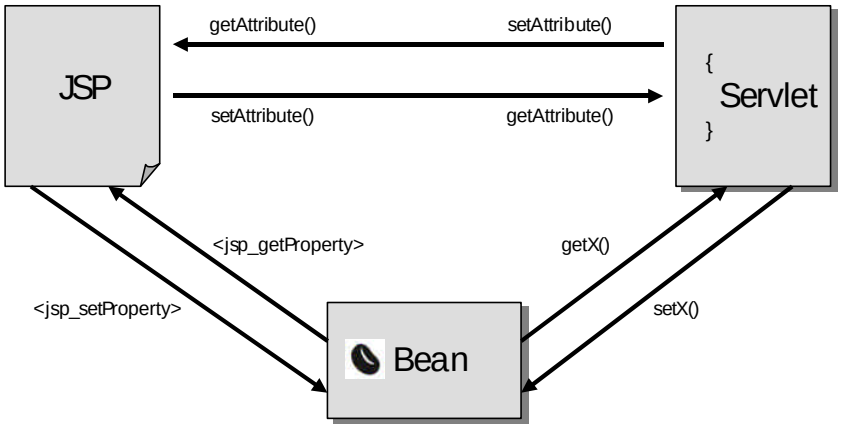

Abbildung 2-8: Die Kommunikation der Komponenten

[1] vgl. [Tura01]

JSP an Servlet

Die JSP-Seite muss die Daten mit `pageContext.setAttribute(name, wert, scope)` zuerst speichern und kann dann mit `<jsp:include .../>` oder `<jsp:forward .../>` die Verantwortung an ein Servlet weitergeben. Das Servlet kann nun mit `HttpServletRequest.getAttribute(name)`, `HttpSession.getAttribute(name)` oder `ServletContext.getAttribute(name)` auf die Daten zugreifen.

Servlet an JSP

Das Servlet kann mit `HttpServletRequest.setAttribute(name, wert)`, `HttpSession.setAttribute(name, wert)` oder `ServletContext.setAttribute(name, wert)` Daten speichern und mit `RequestDispatcher.forward()` oder `RequestDispatcher.include()` die Verantwortung an eine JSP weiterleiten. Die JSP-Seite kann nun mit `pageContext.getAttribute(name, scope)` auf die Daten zugreifen.

JSP über Bean an Servlet

Beans sind Komponenten, die in einer JSP-Seite genutzt werden können, indem sie mit der Aktion `<jsp:useBean id='mB' class='myBean' scope='session'>` bekannt gemacht werden. Der Datentransfer von der JSP-Seite in die Bean und umgekehrt ist danach über die Aktionen `<jsp:setProperty name='mB' property='name'>` bzw. `<jsp:getProperty name='mB' property='name'>` möglich. Die JSP-Seite kann nun mit `<jsp:include .../>` oder `<jsp:forward .../>` die Verantwortung an ein Servlet weitergeben, welches die Bean als Grundlage für weitere Aktionen nutzen kann. Die Bean kann somit als eine Art Transportcontainer genutzt werden. Das Servlet muss nun nur noch mit `HttpServletRequest.getAttribute('mB')`, `HttpSession.getAttribute('mB')` oder `ServletContext.getAttribute('mB')` eine Referenz auf die Bean erhalten und kann dann mit `referenzAufBean.getName()` auf die Daten zugreifen.

Servlet über Bean an JSP

Das Servlet kann, wenn es zuvor eine Referenz auf eine Bean erhalten hat, mit `referenzAufBean.setX()` Daten in der Bean speichern. Die Bean muss nun mit `HttpServletRequest.setAttribute(mB, referenzAufBean)`, `HttpSession.setAttribute(mB, referenzAufBean)` oder `ServletContext.setAttribute(mB, referenzAufBean)` in den entsprechenden Container eingefügt werden. Das Servlet kann dann die Verantwortung mit `RequestDispatcher.forward()` oder `RequestDispatcher.include()` an eine JSP weiterleiten. Diese muss die Bean mit `<jsp:useBean id='mB' class='myBean' scope='session'>` bekannt machen und kann dann die Daten mit `<jsp:getProperty name='mB' property='name'>` auslesen.

2.4 JSP im Vergleich mit anderen Technologien

2.4.1 Vergleich mit CGI

CGI (Common Gateway Interface) ist eine der ältesten Internettechnologien zur Generierung von dynamischen Webseiten. Bei CGI startet der Webserver für jeden Request ein Programm, welches die Ausgabe über die Standardausgabe an den Webserver liefert. Das bringt die gleichen Nachteile wie bei Servlets mit sich. Jede einzelne Zeile eines HTML-Dokuments muss per *println()*-Befehl aus dem CGI-Quellcode heraus generiert werden. Grundsätzlich könnte jede Programmiersprache für die Implementierung von CGIs verwendet werden. Am weitesten verbreitet bei der Implementierung von CGI sind die Sprachen Perl und C/C++. Weil CGI für jeden Request einen neuen Prozess erzeugt, ist es deutlich langsamer als JSP. Aus diesem Grund eignet sich Java nicht gut für CGI, es müsste bei jedem Request eine eigene JVM gestartet werden. Beim Programmieren mit CGI muss der Entwickler die Umsetzung von Requests, Cookies, Sessions und Templates selbst übernehmen.

2.4.2 Vergleich mit ASP

JSP und ASP sind zwei sehr ähnliche Technologien. Beide betten den Programmcode in HTML ein, haben teilweise eine ähnliche Syntax und benutzen die impliziten Objekte *session, request, response* und *application*. Die Ähnlichkeiten kommen daher, dass ASP als Vorbild für die ersten Versionen von JSP diente.

Das ASP zu Grunde liegende Komponentenmodell COM bzw. .NET ist plattformspezifisch. Deshalb sind die damit erzeugten Komponenten nicht portabel. Im Gegensatz dazu verwendet JSP Java Beans bzw. Enterprise Java Beans, die sich durch besonders gute Portabilität auszeichnen. ASP-Seiten werden mit der Skriptsprache VBScript geschrieben. JSP hingegen basiert auf Java, einer ausgewachsenen Programmiersprache, der unzählige APIs zur Verfügung stehen. Ein weiterer wichtiger Vorteil von JSP ist die Geschwindigkeit. Da JSP-Seiten kompiliert werden, laufen sie wesentlich schneller als ASP-Seiten, die bei jedem Request erneut vom Webserver geparst werden müssen.

2.4.3 Vergleich mit PHP

PHP ist ein Open-Source-Projekt, welches schon seit vielen Jahren weiterentwickelt wird. Die Sprache lehnt sich an C und Perl an. In den neusten Versionen ist sogar eine gepufferte Ausgabe und eine Sitzungsverwaltung implementiert. Sitzungsdaten werden allerdings nicht im Hauptspeicher, sondern in Dateien oder einer Datenbank verwaltet. PHP-Skripte sehen im Vergleich zu JSP kryptisch aus, eine Trennung von Präsentations- und Programmlogik ist kaum möglich. Es sind keine einheitlichen DB-Schnittstellen vorhanden. So sind einmal geschriebene PHP-Skripte an eine bestimmte Datenbank gebunden und schlecht portierbar. PHP bietet keine Möglichkeit der Ausgliederung in Komponenten und ist somit schwer skalierbar und erweiterbar. Trotzdem ist PHP weit verbreitet und wird intensiv weiterentwickelt. Es gibt heute sogar objektorientierte Versionen von PHP.

2.5 Laufzeitumgebungen

Im folgenden Kapitel werden Laufzeitumgebungen für JSPs bzw. Servlets vorgestellt. Um JSPs ausführen zu können, braucht man eine sogenannte JSP-Engine, welche die aufgerufene JSP-Seite in ein Servlet übersetzt, im Servlet-Container ausführt und die erzeugte Ergebnisseite an den Browser zurückliefert. Die Laufzeitumgebungen arbeiten in der Regel mit einem Webserver zusammen, der für die Darstellung der statischen Inhalte verantwortlich ist und die Verschlüsselung o.ä. bereitstellt. Was die Verbindung von Webserver und Servlet-Container anbelangt gibt es drei verschiedene Modelle. Beim Standalone-Modell ist der Webserver in Java geschrieben und läuft in derselben JVM wie der Servlet Container. Jakarta Tomcat ist ein Standalone Server.[1] Beim Out-of-Process Modell kommunizieren Webserver und Servlet-Container über eine TCP-Verbindung miteinander. Beim In-Process-Modell wird der Servlet-Container als Plugin in einen in nativem Code geschriebenen Webserver eingeklinkt. Diese Variante bietet die beste Performance. Der IBM WebSphere Application Server kann in dem Bereich als Beispiel genannt werden. Die ZKB SAR hat den Einsatz des IBM WebSphere Application Server als Richtlinie definiert. Die weiter unten beschriebene Beispielapplikation "ADB-Info" wurde für IBM WebSphere entwickelt.

2.5.1 Jakarta Tomcat Server

Auch wenn Jakarta Tomcat bei der ZKB nicht eingesetzt wird, soll der Server hier kurz vorgestellt werden, da er in der Welt der Applikationsserver eine bedeutende Rolle spielt. Tomcat ist die offizielle Server-Referenzimplementierung der Servlet- bzw. JSP-Spezifikation von Sun. Der Server hat eine lange Entwicklungsgeschichte hinter sich. Seit der aktuellen Version 4 besteht der Server aus den zwei Hauptkomponenten Jasper (JSP-Compiler und JSP Laufzeitumgebung) und Catalina (Servlet Container). Jasper ist für die Ausführung der JSP-Seiten zuständig. JSP-Seiten werden kompiliert und an den Servlet Container weitergereicht. Jasper ist eine eigenständige Komponente, die als Servlet realisiert und somit in jedem Servlet Container einsetzbar ist. Der Catalina Servlet Container ist der im Jakarta-Projekt für Tomcat entwickelte Nachfolger von Apache JServ. Zusammen bilden Jasper und Catalina einen Server, der den Web-bezogenen Teil der J2EE-Spezifikation umsetzt.

Jakarta Tomcat legt die Massstäbe für die Portabilität von J2EE-Applikationen unter verschiedenen Applikations-Servern fest. Zahlreiche Standard-Komponenten (auch Struts) werden auf Tomcat entwickelt und von dort auf kommerzielle Applikations-Server portiert.

Tomcat ist Open-Source-Software und für Windows- sowie für Unix- und Linux-Plattformen erhältlich. Der Server wird unter dem Dach der Apache Organisation im Umfeld des Jakarta Projekts entwickelt. Die Apache Group ist eine Gemeinschaft von Entwicklern. Sie gründeten die Apache Software Foundation (ASF), die als Non-Profit-Organisation die Apache Projekte unterstützt und garantiert, dass diese von freiwilligen Mitarbeitern weitergeführt werden. Die

[1] Es gibt allerdings auch Tomcat-Module, die sich in eine vorhandene Apache-Installation einbinden lassen. Diese Variante würde dann dem In-Process-Modell entsprechen.

Beteiligung von Firmen wie Sun, IBM, und Oracle soll das Vertrauen der Anwender in die Fortführung und Qualität der Projekte stärken. Unter dem Dach der Apache Group entstehen - neben dem sehr stark verbreiteten Apache Webserver - Projekte im Perl[1]-, PHP-, XML[2]- und Java-Umfeld (Jakarta Projekt). Jakarta ist ein Open-Source-Projekt und verantwortlich für die Entwicklung von Java-basierten Server-Lösungen wie Tomcat und Frameworks wie Turbine[3] oder Struts. Weitere Informationen gibt es unter [@Apac] bzw. [@Tomc] und in [Tura01]. Das Struts Framework wird in Kapitel 3 noch ausführlich behandelt.

2.5.2 IBM WebSphere Application Server

Der WebSphere Application Server 4.0 ist im Vergleich zu Tomcat nicht nur ein JSP- und Servlet-Container, sondern ein vollständiger J2EE Server. WebSphere stellt Entwicklern ein Portfolio an Programmierhilfen für Anwendungen sowie für dazugehöriges Transaktions-Management für Sicherheitsroutinen und Clustering-Optionen zur Optimierung der Leistungsfähigkeit der Programme zur Verfügung. WebSphere ist kompatibel zu vielen Software-Plattformen und Business-Applikationen, darunter SAP, PeopleSoft, IBM CICS, IMS und Lösungen zur Host-Integration. Darüber hinaus ist WebSphere problemlos mit Datenbanken wie IBM DB2, Oracle oder MS SQL-Server sowie mit führender Middleware wie IBM MQSeries, Tivoli und Lotus Domino einsetzbar. Zu den erweiterten Connectivity-Features des Servers gehören der Support für J2EE JCA, CORBA, Active-X und eine verbesserte Datenbank-Unterstützung mit Connection-Pooling.

Zu den unterstützten Standards und Technologien zählen UDDI (Universal Description Discovery and Integration), SOAP (Simple Object Access Protocol), WSDL (Web Services Description Language), EJB 2.0 Message Beans und eine erweiterte Integration führender XML-Technologien. Die Entwicklungsumgebung ist vollständig J2EE 1.2 zertifiziert.[4]

Eine Kernkomponente des WebSphere Application Server ist der HTTP-Server von IBM, der auf dem Quellcode des Apache Webservers basiert. Der Server arbeitet eng mit dem WebSphere Studio Application Developer (WSAD) zusammen. WSAD integriert Java- und Web-Entwicklungstools in dieselbe Umgebung. Es richtet sich an Entwickler von Java- und J2EE-Applikationen, die integrierte JSP, XML und Web-Services-Unterstützung benötigen. WSAD erleichtert die Anbindung von Datenbanken und ermöglicht das Deployment von erstellten Applikationen auf dem Server.

Theoretisch würden sich fast alle Features der WebSphere-Familie auch irgendwie durch Einsatz von Open-Source-Software umsetzen lassen. Der Vorteil des WebSphere Servers liegt laut IBM allerdings in der All-In-One Lösung der WebSphere-Produktfamilie und des

[1] z.B. mod_perl, ein Perl Plugin für den Apache Server

[2] z.B. Cocoon, ein XML Framework oder Xerces, ein Java XML Parser

[3] Servlet-basiertes Framework zum Erstellen sicherer Web-Applikationen

[4] [@IBM]

kompetenten Supports von IBM. Die Hersteller von Applikationsservern versuchen den Entwickler so gut wie möglich zu unterstützen, indem sie ihn von den Details der Systemarchitektur und dem komplexen Zusammenwirken der Komponenten im Hintergrund abschirmen. Ausserdem bieten grosse und finanzstarke Unternehmen eine gewisse Sicherheit, was den Fortbestand der Produkte anbelangt. Diese Sicherheit sehen viele Kunden bei den Open-Source-Projekten nicht gegeben.

2.5.3 Andere Applikationsserver

IBM teilt sich den Markt für Applikationsserver mit zahlreichen anderen Firmen wie Bea[1], Macromedia, Hewlett-Packard, Sun und Oracle. An dieser Stelle sollte zudem Apple's WebObjects Application Server erwähnt werden. WebObjects zeichnet sich durch sehr komfortable Entwicklungswerkzeuge aus, die dem Entwickler die Arbeit erleichtern. Der Server ist nicht sehr stark am Markt vertreten. Die Deutsche Bank AG betreibt ihre Online Banking Plattform auf Apple WebObjects. Einen umfangreichen Vergleichstest aktueller kommerzieller Applikationsserver erstellte die Computerwoche in [@Comp].
Daneben gibt es im Bereich Open-Source Software bzw. Public Software vielversprechende Projekte wie Enhydra[2], Orion[3], JBoss[4] oder Resin[5]. Diese J2EE Server stellen für den Mittelstand sowie für kleinere Firmen ernstzunehmende Alternativen dar.

[1] Bea (Weblogic Server [@Bea]) gehört zusammen mit IBM und Macromedia (Coldfusion Server [@Macr]) zu den marktführenden Herstellern von Applikationsservern.
[2] vgl. [@Enhy]
[3] vgl. [@Orio]
[4] vgl. [@JBos]
[5] vgl. [@Resi]

3 Jakarta Struts

Die Model 2 Architektur vermittelt einen Eindruck davon, wie schwer es ist, eine MVC-Applikation selbst zu entwickeln. Dieser Eindruck wird noch verstärkt, wenn man die vielfältigen Möglichkeiten der Kommunikation von Komponenten untereinander betrachtet. Jakarta Struts will den Entwickler beim Erstellen von MVC-Applikationen entlasten.

Das Struts-Projekt wurde im Mai 2000 von Craig R. McClanahan initiiert, um der Java-Gemeinschaft ein standardisiertes MVC-Framework zur Verfügung zu stellen. Im Januar 2001 wurde Struts 1.0 fertig gestellt. 2001 war auch das Jahr, in dem Struts einen sehr grossen Bekanntheitsgrad erlangt hat. Dass das Struts-Projekt unter dem Dach der Apache Organisation entwickelt wurde, hat sicher sehr stark dazu beigetragen. Struts ist Open-Source Software. Das Framework wird unter der "Apache Software Foundation License" (ASF) vertrieben. Der Code ist durch Copyright geschützt, darf aber frei in jeder beliebigen Anwendung verwendet werden.[1]

In diesem Kapitel wird zuerst die Architektur von Struts erläutert und der Programmablauf dargestellt. Danach wird ein Überblick über das Framework gegeben und später werden die einzelnen Komponenten im Detail beschrieben. Das UML-Klassendiagramm von Struts und ein Sequenzdiagramm ermöglichen einen Einblick in die interne Arbeitsweise des Frameworks. Am Ende dieses Kapitels befindet sich eine Art "Schritt für Schritt"-Anleitung, die anhand eines sehr einfachen Beispiels das Erstellen einer Struts-Applikation erklärt. Dies ist sozusagen die praktische Anleitung zum Erlernen von Struts, nachdem der Leser die eher theoretischen Kapitel über die Struts Komponenten gelesen hat. Abgeschlossen wird das Kapitel mit einem Überblick über einige weitere Frameworks, die ähnliche Ansätze wie Struts verfolgen. Die Implementierung der Beispielprogramme und die Ausführungen zum Framework beziehen sich ausschliesslich auf die Struts Version 1.0.2. Eine detaillierte Dokumentation des Frameworks findet sich in [@Stru3].

3.1 Architektur

Getreu dem Model-View-Controller Paradigma verwendet Struts drei Hauptkomponenten: Ein als Controller fungierendes Servlet, Java Server Pages als View-Komponenten sowie die Model-Komponenten (Geschäftslogik) in Form von Java Beans.

[1] Mehr zur Apache Software Foundation License auf [@Apac]

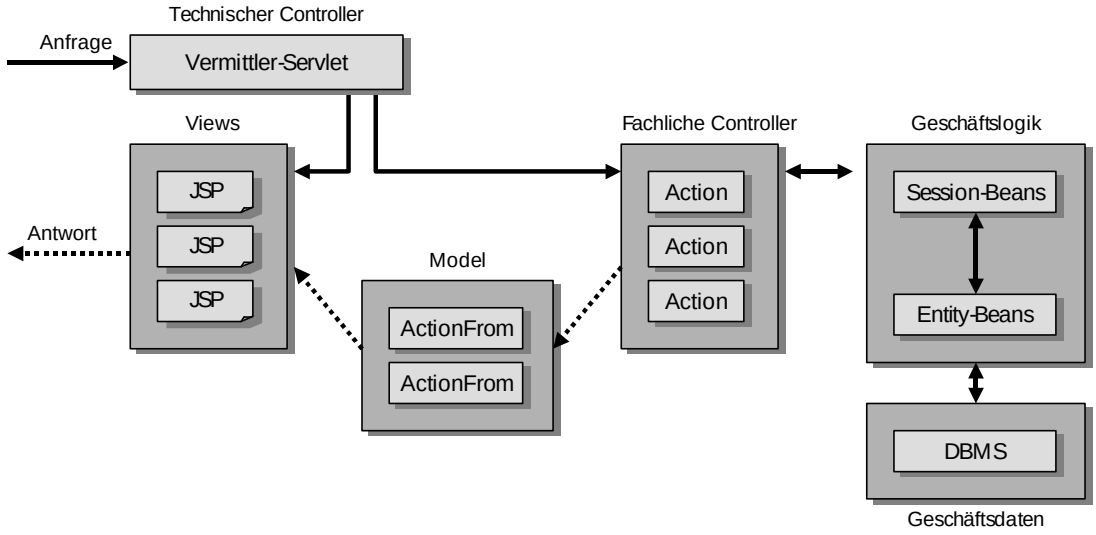

Abbildung 3-1: Architektur des Struts Frameworks

Es ist nicht immer ganz einfach die einzelnen Struts-Komponenten zuzuordnen. Laut Struts Spezifikation gehören die *Action*-Klassen zusammen mit dem *ActionServlet* (Vermittler Servlet) zum Controller, wobei man eine Aufteilung in technische und fachliche Controller vornehmen kann. Die *ActionForm*-Klassen gehören zu den Model-Komponenten, obwohl sie dem View sehr nahe stehen. Die eigentlichen Model-Klassen – also die Geschäftslogik - werden von Struts nicht implementiert, sondern müssen vom Entwickler selbst erstellt werden. Sie werden von den *Action*-Klassen aufgerufen. Die Views werden mit JSP implementiert.[1]

3.2 Programmablauf

Die Abbildung auf der folgenden Seite veranschaulicht das Zusammenspiel der einzelnen Komponenten aus denen Struts besteht. Es werden die Vorgänge von einer Anfrage eines Clients (HTTP-Request) bis zur fertigen Antwort (HTTP-Response) beschrieben. Obwohl der Leser an dieser Stelle die Struts-Komponenten noch nicht kennt, wird der Ablauf einer Anfrage erläutert, bevor im Detail auf die Komponenten eingegangen wird. Dies erleichtert dem Leser die Zusammenhänge zwischen den später beschriebenen Komponenten besser zu verstehen.

[1] vgl. [Kamm02]

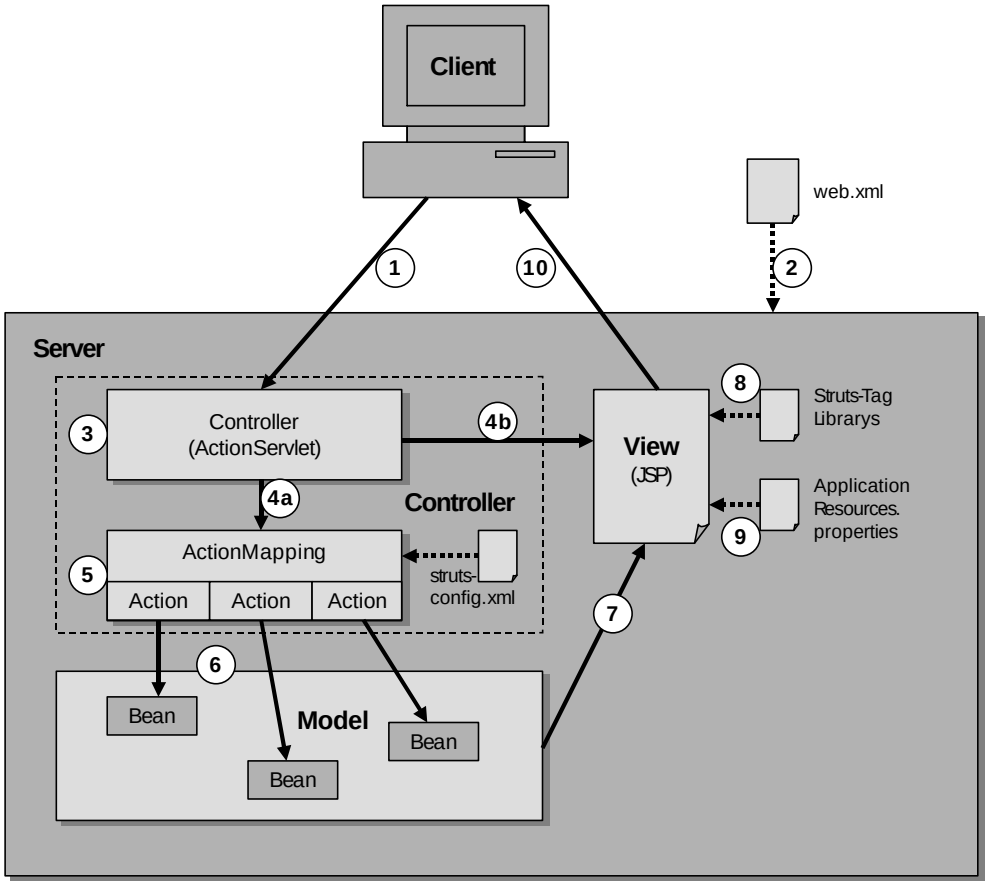

Abbildung 3-2: Programmablauf in Struts

Am Anfang steht die Anfrage eines Clients (i.d.R. Abschicken eines Formulars im Browser) an den Server (1). Die XML-Konfigurationsdatei *web.xml* (2) konfiguriert den Server dahingehend, dass das *StrutsAction*-Servlet die Anfrage entgegennimmt (3). Das Servlet entscheidet nun, ob die Anfrage an die Klasse *ActionMapping* übergeben wird (4a) oder ob die Anfrage gleich an eine View-Komponente weitergeleitet werden kann (4b). Dies entscheidet der Controller in Abhängigkeit davon, ob zur Darstellung der View Geschäftslogik (z.B. DB-Zugriffe) angesprochen werden muss oder ob direkt eine Ausgabe erfolgen kann. Wird eine bestimmte *Action*-Klasse anhand des Mappings der Datei *struts-config.xml* zur Verarbeitung ausgewählt (5), leitet diese Klasse die Anfrage an die entsprechende Java Bean, welche die Geschäftslogik enthält, weiter (6). Danach wird eine View-Komponente aufgerufen (7), die mit Hilfe der Struts Tag-Libraries (8) die Ergebnisse der Anfrage darstellt und die Antwort an den Client schickt (10). Die Properties-Datei (9) enthält die in den Views verwendeten Nachrichten in der jeweiligen Sprache. Für die Ablaufreihenfolge unrelevant, und aus diesem Grund in der Abbildung oben nicht eingezeichnet, sind die *ActionForm*-Klassen. Sie dienen dem Zwischenspeichern und Validieren der Formulardaten, werden vom Controller (3) erzeugt und können in den *Action*-Klassen (5) und in den View-Komponenten angesprochen werden. Die *ActionForm*-Klassen können also als eine Art Transportcontainer zwischen den Komponenten angesehen werden.

3.3 Überblick über das Framework

Der Controller bündelt die HTTP-Requests und leitet sie an andere Objekte (z. B. an Java Server Pages) des Frameworks weiter. Sobald der Controller initialisiert wird, parst er eine Datei, in der die Konfigurationsressourcen enthalten sind (*struts-config.xml*). Die Konfigurationsressourcen definieren u. a. die *Action*-Mappings der Anwendung. Der Controller verwendet diese Mappings, um HTTP-Requests in Anwendungsaktionen umzuwandeln. Ein Mapping muss mindestens zwei Dinge spezifizieren: Erstens einen Request-Pfad und zweitens den Objekt-Typ, mit dem der Request weiter verfahren soll. Das *Action*-Objekt kann entweder den Request verarbeiten und dem Client (normalerweise dem Browser) eine Antwort liefern, oder es kann veranlassen, dass der Request an ein anderes *Action*-Objekt weitergeleitet wird. Wenn beispielsweise ein Login erfolgt, ist es denkbar, dass das für den Login zuständige *Action*-Objekt die Kontrolle an ein anderes *Action*-Objekt weitergibt, welches für die Darstellung des Hauptmenüs zuständig ist.

Action-Objekte werden mit dem Controller der Anwendung verbunden und haben auf diese Weise Zugriff auf die Methoden des Servlets. Wenn die Kontrolle weitergegeben wird, kann ein Objekt indirekt ein oder mehrere gemeinsam genutzte Objekte (z. B. Java Beans) weiterleiten, indem es sie in einer der Standard-Collections platziert, die von Java-Servlets verwendet werden. Ein *Action*-Objekt kann beispielsweise ein Warenkorb-Bean erzeugen, dem Warenkorb einen Artikel hinzufügen, das Bean in einer Session-Collection platzieren und dann die Kontrolle einem anderen *Action*-Objekt übergeben, welches beispielsweise JSPs benutzt, um den Inhalt des Warenkorbes anzuzeigen. Da jeder Client seine eigene Session hat, hat jeder Client auch seinen eigenen Warenkorb. In einer Struts-Anwendung kann ein Grossteil der Geschäftslogik durch Java Beans repräsentiert werden.[1]

Java Beans können darüber hinaus verwendet werden, um HTML-Formulare zu verwalten. Ein zentrales Problem beim Design von Webanwendungen besteht darin, die Angaben aufzubewahren und zu validieren, die ein Benutzer zwischen zwei Requests in das Formular eingegeben hat. Mit Struts können die Daten eines Formulars sehr einfach in einem *ActionForm*-Bean gespeichert werden. Das Bean wird in einer der normalen Collections innerhalb des gemeinsamen Kontextes gespeichert, so dass es von anderen Objekten, speziell von den *Action*-Objekten, verwendet werden kann. Das *ActionForm*-Bean kann von einer JSP benutzt werden, um die Benutzereingaben zu speichern, es kann von einem *Action*-Objekt verwendet werden, um die Daten zu validieren und es kann dann wieder von einer JSP benutzt werden, um die Formularfelder mit diesen Eingaben zu füllen. Im Fall von Validierungsfehlern verwendet Struts einen von verschiedenen Stellen zugänglichen Mechanismus, um passende Fehlermeldungen anzuzeigen.

[1] vgl. [@Stru3]

In Struts wird ein *ActionForm*-Bean in der Konfigurations-Ressource definiert und mit einem *Action*-Mapping verbunden, indem ein gemeinsamer Eigenschaftsname verwendet wird. Wenn ein Request eine Aktion aufruft, die ein *ActionForm*-Bean verwendet, erhält das Controller-Servlet entweder ein *ActionForm*-Bean oder es erzeugt ein solches und reicht es anschließend an das *Action*-Objekt weiter. Das *Action*-Objekt kann dann die Inhalte des *ActionForm*-Beans prüfen, bevor das zugehörige HTML-Formular angezeigt wird. Außerdem kann es Nachrichten speichern, welche von dem Formular verarbeitet werden sollen. Wenn das erledigt ist, kann das *Action*-Objekt die Steuerung durch eine Weiterleitung an das HTML-Formular (üblicherweise eine JSP) zurückgeben. Der Controller kann dann auf den HTTP-Request antworten und den Client an die Java Server Page weiterleiten.

Das Struts-Framework beinhaltet benutzungsdefinierte Tags, die automatisch Formularfelder aus einem *ActionForm*-Bean heraus befüllen können. Das einzige, was die meisten Java Server Pages über den Rest des Frameworks wissen müssen, sind die richtigen Feldnamen und das Ziel, an welches das Formular geschickt werden soll. Nachrichten - z.B. die Nachrichten, die von einem *Action*-Objekt gesetzt werden - können ausgegeben werden, indem ein einziges benutzungsdefiniertes Tag verwendet wird. Weitere applikationsabhängige Tags können definiert werden, um Details der Implementierung für die Entwickler der Views unsichtbar zu machen.

Die Tags im Struts-Framework wurden so entworfen, dass sie die Fähigkeiten der Java-Plattform bezüglich der Internationalisierung nutzen. Alle Feldbezeichnungen und Nachrichten können aus einer Nachrichten-Ressource bezogen werden, wobei Java automatisch die richtige Ressource für Land und Sprache des Clients verwendet. Um Nachrichten für eine andere Sprache zur Verfügung zu stellen, ist einfach eine weitere Ressourcen-Datei zu erstellen. Neben der Internationalisierung gibt es weitere Vorzüge dieses Verfahrens wie z. B. konsistente Namensgebung unter den Formularen sowie die Fähigkeit, alle Bezeichnungen und Nachrichten von einem zentralen Ort aus zu kontrollieren.

In einfachen Anwendungen kann ein *Action*-Objekt die Geschäftslogik implementieren, die mit einem Request verbunden ist. In den meisten Fällen sollte jedoch ein *Action*-Objekt den Request an ein anderes Objekt weiterleiten, normalerweise an eine Java Bean. Um die Wiederverwendbarkeit auf anderen Plattformen zu gewährleisten, sollten die Java Beans der Geschäftslogik keine Referenzen zu anderen Objekten der Web-Anwendung haben. Das *Action*-Objekt sollte die benötigten Details aus dem HTTP-Request extrahieren und diese an die Geschäftslogik-Beans als reguläre Java-Variablen weiterleiten.

In einer Datenbankanwendung können die Geschäftslogik-Beans eine Datenbank ansprechen und das Ergebnis zurück an das *Action*-Servlet schicken, welches die Daten dann in einer

ActionForm-Bean speichert und anschliessend eine JSP-View aufruft, welche die Daten darstellt. Weder *Action*-Servlet noch JSP müssen wissen, woher das Ergebnis kommt.[1]

Die folgenden drei Kapitel (3.4 bis 3.6) beschreiben die Model-, View- und Controller-Komponenten im Detail.

3.4 Model Komponenten

Der Model-Teil eines MVC-basierten Systems kann in zwei Bereiche unterteilt werden: Der eine umfasst den internen Zustand des Systems, der andere die Aktionen um diesen Zustand zu ändern.

Allgemein gesagt repräsentiert eine Applikation den internen Zustand des Systems als eine Sammlung von Java Beans mit Eigenschaften, welche die Details des Zustandes definieren. Abhängig von der Komplexität der Applikation können diese Beans selbständig sein (und daher wissen, wie sie ihre Zustandsinformationen persistent speichern können) oder sie können Fassaden sein, die wissen, wie sie bei Bedarf Informationen aus externen Quellen erhalten können (wie z. B. aus einer Datenbank). Auch Enterprise Java Beans (EJBs) werden üblicherweise verwendet, um interne Zustandsinformationen zu repräsentieren.

In weniger umfangreichen Applikationen können die möglichen Aktionen in die *Action*-Klassen eingebettet werden, die einen Teil der Rolle des Controllers übernehmen. Dies ist angemessen, solange die Logik sehr einfach ist oder wenn die Wiederverwendbarkeit der Geschäftslogik in anderen Umgebungen nicht in Erwägung gezogen wird. Das Struts Framework unterstützt diese Vorgehensweise, jedoch ist die Trennung der Geschäftslogik von den *Action*-Klassen zu empfehlen. Struts überlässt dem Entwickler die Entwicklung der Model-Klassen jedoch vollständig. Der Hauptbestandteil des Frameworks ist der Controller und die Tag Libraries zum Erstellen der Views.[2]

3.4.1 *ActionForm*-Beans

ActionForm-Beans dienen dem Zwischenspeichern und Validieren von Formulardaten in einer View. Das Struts Framework nimmt generell an, dass für jedes Formular in der Anwendung ein *ActionForm*-Bean (d. h. eine Klasse, die von der *ActionForm*-Klasse erbt) erzeugt wurde. Falls solche Beans in der Konfigurationsdatei *struts-config.xml* definiert sind, wird das Struts-Controller-Servlet automatisch folgende Schritte ausführen, bevor die passende *Action*-Methode gestartet wird:

- In der Session des Benutzers wird unter dem passenden Schlüssel nach einer Instanz eines Beans der passenden Klasse gesucht.

[1] vgl. [@Stru3]
[2] vgl. [@Stru3]

- Falls kein solches Bean im Session-Gültigkeitsbereich verfügbar ist, wird automatisch ein neues erzeugt und der Session des Benutzers hinzugefügt.
- Für jeden Request-Parameter, dessen Name mit dem Namen einer Eigenschaft im Bean korrespondiert, wird die passende Setter-Methode aufgerufen. Dies entspricht dem JSP-Befehl *<jsp:setProperty>*.
- Das aktualisierte *ActionForm*-Bean wird an die *perform()*-Methode der *Action*-Klasse übergeben.

Die *ActionForm*-Klasse selbst hat keine spezielle Methode die implementiert werden müsste. Typischerweise implementiert eine *ActionForm*-Klasse nur Getter- und Setter-Methoden für Eigenschaften (getXxx()- und setXxx()-Methoden für jedes Feld, das im Formular vorhanden ist), jedoch keine Geschäftslogik. Das *ActionForm*-Objekt bietet einen standardisierten Validierungsmechanismus an. Wenn die Methode *validate()* überschrieben wird, validiert Struts den Input aus dem Formular automatisch. Siehe 3.5.2 für weitere Details. Anstatt der *ActionForm*-Validierung kann auch eine eigene Validierung im *Action*-Objekt implementiert werden. Dies sollte aber nur dann geschehen, wenn für die Validierung Geschäftslogik-Beans angesprochen werden müssen.

Struts unterstützt zudem Formulare, die sich über mehrere Seiten hinweg erstrecken (z.B. Wizards) und trotzdem nur eine *ActionForm*-Klasse verwenden. Die Formulare müssen in diesem Fall alle an die gleiche *Action* gesendet werden.

3.4.2 Beans für den Systemzustand

Der aktuelle Zustand eines Systems wird normalerweise durch eine Sammlung von einer oder mehreren Java Beans repräsentiert. Deren Eigenschaften definieren den gegenwärtigen Zustand. Beispielsweise besteht ein Warenkorbsystem aus einer Bean, die den für jeden Einkäufer verwalteten Warenkorb darstellt und die eine Sammlung von Artikeln beinhaltet, welche der Einkäufer zum Kauf ausgewählt hat. Daneben besteht das System außerdem aus verschiedenen Beans mit Informationen über das Benutzerprofil (z.B. Informationen über Kreditkarten und Lieferadressen) sowie über einen Katalog verfügbarer Artikel und Bestandsinformationen.

In kleineren Applikationen und für das Speichern von Zustandsinformationen, die nicht für längere Zeit gespeichert werden müssen, kann eine Sammlung von System-State Beans benutzt werden. Meist jedoch enthalten die System-State Beans Daten, die permanent in externen Datenbanken gespeichert werden (z.B. ein CustomerBean-Objekt, das mit einer Zeile in der CUSTOMERS Tabelle korrespondiert). Diese Beans werden bei Bedarf erzeugt und wieder aus dem Speicher des Servers entfernt. In grösseren Applikationen werden dafür auch Entity Enterprise Java Beans eingesetzt.

3.4.3 Beans für die Geschäftslogik

Die funktionale Logik einer Applikation sollte durch Methodenaufrufe auf Java Beans implementiert werden. Die Geschäftslogik kann entweder in den Beans für den Systemzustand oder in separaten Geschäftslogik-Beans enthalten sein. Im letzten Fall müssten die System State Beans den Geschäftslogik-Beans als Parameter beim Methodenaufruf mitgegeben werden.

Für die maximale Wiederverwendbarkeit des Codes sollten die Beans für die Geschäftslogik so entworfen und implementiert werden, dass sie nicht wissen, dass sie in der Umgebung einer Web-Anwendung ausgeführt werden. Es sollten also keine Import-Anweisungen für Java Web-Klassen (z.B. javax.servlet.*-Klassen) implementiert werden, um Informationen mit Hilfe des Request-Objekts zu übertragen. Anstatt dessen sollte die *Action*-Klasse alle benötigten Informationen aus dem HTTP-Request extrahieren und den Geschäftslogik-Beans als Methodenparameter oder mit Hilfe von Getter-/Setter-Methoden übergeben. Dadurch können die Geschäftslogik-Beans auch in anderen Applikationen wiederverwendet werden.

Abhängig von der Komplexität der Applikation können die Beans für die Geschäftslogik gewöhnliche Java Beans sein, die mit den System State Beans interagieren oder es können Java Beans sein, die eine Datenbank ansprechen. In grösseren Applikationen kommen häufig Enterprise Java Beans (EJB) zum Einsatz.

3.5 View Komponenten

Der View-Teil einer Struts-basierten Anwendung wird üblicherweise mit Java Server Pages konstruiert. JSP-Seiten können statisches HTML oder XML enthalten. Ausserdem können mit Hilfe spezieller *Action*-Tags erzeugte Inhalte zur Laufzeit eingefügt werden. Die JSP-Umgebung beinhaltet eine Sammlung von standardisierten Tags wie z. B. <*jsp:useBean*>, deren Zweck jeweils in der Java Server Pages Specification beschrieben ist. Zusätzlich gibt es eine standardisierte Möglichkeit eigene Tags zu definieren und sie in Custom Tag Libraries zu organisieren.

Struts beinhaltet eine umfassende Tag-Bibliothek, die es ermöglicht, GUIs zu erzeugen, die vollständig internationalisierbar sind und die gut mit den *ActionForm*-Beans interagieren, um die Formulardaten zu verwalten. Die Verwendung dieser Tags wird später im Detail betrachtet.

Zusätzlich zu den JSP-Seiten und den darin enthaltenen Aktionen und Tags ist es für Geschäftsobjekte oftmals notwendig, selbst in der Lage zu sein, basierend auf ihrem aktuellen Zustand zur Laufzeit in HTML (oder XML) überführt zu werden. Der erzeugte Output eines solchen Objektes kann einfach in eine resultierende JSP Seite eingebunden werden, indem das standardisierte <*jsp:include*>-Action-Tag verwendet wird.[1]

[1] vgl. [@Stru3]

3.5.1 Struts-Tags

Die Tag-Libraries von Struts sind ein wesentlicher Bestandteil des Frameworks. Sie werden in vier Gruppen unterteilt: Struts HTML Tags, Struts Bean Tags, Struts Logic Tags und Struts Template Tags. Die Struts HTML Tag Library wird zum Erzeugen von Formularen benutzt. Ausserdem enthält sie einige Tags, die generell in Struts JSPs eingesetzt werden. Die Struts Bean Tag Library enthält Tags für den Zugriff auf Beans und deren Properties sowie für das Erzeugen neuer Beans aus Cookies, Requests oder Request-Parametern. Die Struts Logic Tag Library ermöglicht die bedingte Ausgabe von Text bzw. Tags, die wiederholte Ausgabe von Text bzw. Tags sowie die Umsetzung von Steuerungslogik in der View. Die Struts Template Tag Library enthält Tags, die einen Template-Mechanismus definieren. Die einzelnen Tags werden nun kurz beschrieben. Eine detaillierte Dokumentation aller Funktionalität und Parameter findet sich in [@Stru3].

Struts HTML Tags

Tag Name	Beschreibung
base	Erzeugt ein HTML <base> Element mit dem Pfad zur Applikation.
button	Erzeugt ein Eingabefeld vom Typ Button für ein HTML-Formular.
cancel	Erzeugt einen Cancel-Button für ein HTML-Formular.
checkbox	Erzeugt eine Checkbox für ein HTML-Formular.
errors	Zeigt Fehlermeldungen an, falls die *Errors*-Collection *Errors* enthält.
file	Erzeugt einen "Datei öffnen" Dialog.
form	Erzeugt ein HTML-Formular-Tag.
hidden	Erzeugt verstecktes (hidden) Textfeld für ein HTML-Formular.
html	Erzeugt ein HTML <html> Element.
image	Erzeugt einen grafischen Submit-Button für ein HTML-Formular.
img	Erzeugt ein HTML <img> Element. Falls der Client keine Cookies unterstützt, wird automatisch URL-Rewriting eingesetzt um die Session-ID der URL als Parameter zu übergeben. Mit Hilfe dieses Tags kann beispielsweise eine Grafik in der View angezeigt werden, die dynamisch von einem Servlet generiert wird (z.B. Börsenkurse).
link	Erzeugt ein HTML <link> Element. Falls der Client keine Cookies unterstützt, wird automatisch URL-Rewriting eingesetzt um die Session-ID der URL als Parameter zu übergeben.
multibox	Erzeugt eine Menge von Checkbox Elementen für ein HTML-Formular.
option	Erzeugt ein Element einer Auswahlliste für ein HTML-Formular.
options	Erzeugt mehrere Elemente einer Auswahlliste für ein HTML-Formular.
password	Erzeugt ein Passwort-Textfeld (verdeckte Eingabe) für ein HTML-Formular.
radio	Erzeugt ein Radiobutton für ein HTML-Formular.
reset	Erzeugt einen Reset-Button für ein HTML-Formular.
rewrite	Erzeugt eine URI wie der link-Tag nur ohne das <a>.

select	Erzeugt eine Auswahlliste für ein HTML-Formular
submit	Erzeugt einen Submit-Button für ein HTML-Formular.
text	Erzeugt ein Eingabefeld für ein HTML-Formular.
textarea	Erzeugt ein Eingabebereich (mehrzeilig) für ein HTML-Formular.

Struts Bean Tags

Tag Name	Beschreibung
cookie	Liest einen Cookie aus und erzeugt den Wert (die Werte) als Variable im Gültigkeitsbereich page.
define	Liest die Property einer Bean und erzeugt den Wert (die Werte) als Variable im Gültigkeitsbereich page.
header	Liest einen Request-Header aus und erzeugt den Wert (die Werte) als Variable im Gültigkeitsbereich page.
include	Ähnliche Funktionalität wie der <*jsp:include*>-Tag. Falls der Client keine Cookies unterstützt, wird automatisch URL-Rewriting eingesetzt um die Session-ID der URL als Parameter zu übergeben. Mit Hilfe dieses Tags kann eine View aus mehreren Unterseiten zusammengesetzt werden. Der Unterschied zum <*jsp:include*>-Tag ist, dass *bean:include* zur Laufzeit ausgeführt wird, also eine bedingte Ausführung des Tags möglich ist.
message	Erzeugt eine internationalisierte Nachricht und gibt sie auf der Standardausgabe aus.
page	Erzeugt eine Bean aus einem Element im Gültigkeitsbereich *page*.
parameter	Liest einen Request-Parameter aus und erzeugt den Wert als Variable im Gültigkeitsbereich page.
resource	Lädt eine Ressource der Web-Applikation und schreibt sie in eine Bean.
size	Erzeugt eine Bean mit der Anzahl der Elemente einer Collection oder Map.
struts	Erzeugt eine Bean aus einem Objekt der internen Struts Konfiguration.
write	Gibt die Property einer Bean auf der Standardausgabe aus.

Struts Logic Tags

Tag Name	Beschreibung
equal	*If*-Tag. Führt den Body des Tags aus, wenn die angegebene Variable einem bestimmten Wert entspricht.
forward	Leitet an eine andere JSP weiter (entspricht *PageContext.forward()* oder *HttpServletResponse.sendRedirect()*). Falls der Client keine Cookies unterstützt, wird automatisch URL-Rewriting eingesetzt, um die Session-ID der URL als Parameter zu übergeben.
greaterEqual	*If*-Tag. Führt den Body des Tags aus, wenn die angegebene Variable grösser oder gleich einem bestimmten Wert ist.
greaterThan	*If*-Tag. Führt den Body des Tags aus, wenn die angegebene Variable grösser einem bestimmten Wert ist.

iterate	Schleife. Wiederholt den Body des Tags für jedes Element einer übergebenen Collection.
lessEqual	*If*-Tag. Führt den Body des Tags aus, wenn die angegebene Variable kleiner oder gleich einem bestimmten Wert ist.
lessThan	*If*-Tag. Führt den Body des Tags aus, wenn die angegebene Variable kleiner einem bestimmten Wert ist.
match	*If*-Tag. Führt den Body des Tags aus, wenn ein bestimmter Wert ein Substring der angegebenen Variable ist.
notEqual	*If*-Tag. Führt den Body des Tags aus, wenn die angegebene Variable ungleich einem bestimmten Wert ist.
notMatch	*If*-Tag. Führt den Body des Tags aus, wenn ein bestimmter Wert kein Substring der angegebenen Variable ist.
notPresent	*If*-Tag. Führt den Body des Tags aus, wenn ein bestimmter Wert im Request nicht vorhanden ist.
present	*If*-Tag. Führt den Body des Tags aus, wenn ein bestimmter Wert im Request vorhanden ist.
redirect	Erzeugt einen HTTP Redirect (entspricht *HttpServletResponse.sendRedirect()*). Falls der Client keine Cookies unterstützt, wird automatisch URL-Rewriting eingesetzt, um die Session-ID der URL als Parameter zu übergeben.

Struts Template Tags

Put-Tags legen Inhalte in den Request-Gültigkeitsbereich. Diese Inhalte werden durch ein *get*-Tag in eine andere JSP (dem Template) eingelesen. Dieses Template kann durch den *insert*-Tag in eine weitere JSP eingefügt werden.

Tag Name	Beschreibung
insert	Fügt ein Template ein. Templates sind JSP Seiten, die parametrisierte Inhalte enthalten. Diese Inhalte werden durch put-Tags erzeugt.
put	Legt Inhalte in den Request-Gültigkeitsbereich.
get	Liest Inhalte aus dem Request-Gültigkeitsbereich.

Die Struts Template Tags bieten nicht viel Komfort und Funktionalität. Aus diesem Grund wird bei grösseren Applikationen oft eine separate Template Engine benötigt. Struts unterstützt in der aktuellen Version 1.0.2 offiziell keine externe Template Engine, soll aber in zukünftigen Versionen mit Velocity zusammenarbeiten können. Velocity ist eine leistungsfähige Java Template Engine, die ebenfalls im Jakarta Projekt entwickelt wird.[1]

3.5.2 Internationalisierung

Um die Internationalisierung von Struts nutzen zu können, müssen lediglich sogenannte Properties-Dateien im Klassenpfad der Web-Applikation erstellt werden. Ausserdem müssen

[1] Siehe auch [@Veloc]

in den Views die Struts Tags zur Ausgabe von Text benutzt werden. Der Tag *<html:html locale="true">* am Anfang einer JSP weist Struts an die Internationalisierung zu aktivieren. Dazu ermittelt Struts über den Browser die eingestellte Sprache des Benutzers.

Der Name der Properties-Dateien kann in der Konfigurationsdatei *web.xml* definiert werden. Im folgenden Beispiel ist der Name *ApplicationResources*. Die Endung ist *.properties*.

```
01    <servlet>
02
03      <servlet-name>action</servlet-name>
04      <servlet-class>org.apache.struts.action.ActionServlet</servlet-class>
05
06      <init-param>
07        <param-name>application</param-name>
08        <param-value>ApplicationResources</param-value>
09      </init-param>
10
11    </servlet>
```

Listing 3-1: web.xml - Definition der Property-Datei einer Applikation

Der Name der Datei mit den Nachrichten in der Standardsprache wäre dann *ApplicationResources.properties*. Ein Eintrag in diese Textdatei würde beispielsweise lauten: *prompt.saghallo=Hallo*. Um eine weitere Sprache zu unterstützen muss eine weitere Datei mit dem Namen *ApplicationResources_xx.properties* angelegt werden. xx steht dabei für den ISO-Sprachcode einer Sprache[1]. *ApplicationResources_fr.properties* wäre beispielsweise der Name für eine französische Property-Datei. Ein Eintrag in diese Textdatei müsste dann lauten: *prompt.saghallo=Bonjour*. Die Ausgabe dieser Nachrichten in einer JSP View würde mit dem Tag *<bean:message key="prompt.saghallo"/>* implementiert werden. Struts kann ausserdem Zahlen und Datumswerte in der jeweiligen Landeseinstellung formatieren.

3.5.3 Formulare und Formular-Validierung

Innerhalb einer Web-Applikation ist es sehr häufig notwendig die Benutzereingaben in einem Formular zu validieren und im Fehlerfall geeignete Fehlermeldungen auszugeben. Die Ausgabe der Fehlermeldungen geschieht in der Regel in der gleichen View wie die Formulareingabe. Dabei ist es wichtig, dass der Benutzer bereits richtige Eingaben nicht noch einmal eingeben muss. Die richtigen Eingaben müssen also erneut in das Formular geschrieben werden, während die Felder mit Fehlern leer sind und eine Fehlermeldung den Benutzer über seine Fehler informiert.

Wenn der Entwickler einer Web-Applikation diese Funktionalität ohne Struts implementieren möchte, ist der Aufwand wesentlich höher. Er müsste die Benutzereingaben in das abgesendete Formular zuerst selbst in eine Bean schreiben, die Eingaben danach validieren und im Fehlerfall die JSP erneut aufrufen, um eine Fehlermeldung anzuzeigen. Das Tag im JSP müsste in diesem Fall wie folgt aussehen:

[1] Eine Liste aller Sprachcodes findet sich im Javadoc der Klasse *ResourceBundle* in [@J2SE]

```
<input type="text" name="username" value="<%= loginBean.getUsername() %>"/>
```

Dieser umständliche Weg kann mit Struts wesentlich vereinfacht werden. Der Struts Controller schreibt die Formulardaten automatisch in eine Bean und validiert diese. Die Informationen über die Zusammenhänge zwischen einer JSP-View und der Formular-Bean erhält der Struts-Controller aus der Konfigurationsdatei *struts-config.xml*. Ein einfacher Struts Tag der Form

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

ermöglicht diese Funktionalität innerhalb der JSP. Das Java Bean (*ActionForm*-Bean), aus dem die Werte bezogen werden, muss im Tag nicht explizit definiert werden. Struts wählt automatisch die richtige Bean. Ein einfaches Tag am Anfang einer JSP gibt im Fehlerfall Fehlermeldungen aus, ohne dass der Entwickler sich explizit darum kümmern muss:

```
<html:errors/>
```

Die Fehlermeldungen können aus den internationalisierten Nachrichten-Ressourcen der Applikation bezogen werden. Die Anleitung zum Entwickeln einer kleinen Beispielapplikation in Kapitel 3.8 beschreibt dies im Detail.

Struts unterstützt, wie schon weiter oben beschrieben, folgende HTML-Tags für das Erzeugen von Formularfeldern:

- Checkbox
- Radiobutton
- Auswahlliste
- Texteingabefeld
- Mehrzeiliges Texteingabefeld
- Texteingabefeld für Passwörter
- Versteckte Felder
- Reset Button
- Submit Button

Um die automatische Validierung der Formulardaten zu aktivieren, muss in der zum Formular gehörenden *ActionForm*-Klasse die Methode *validate()* überschrieben werden. Die Methode wird vom Struts-Controller aufgerufen nachdem die Properties gesetzt wurden, aber noch bevor die *perform()*-Methode der *Action*-Klasse aufgerufen wird.

Die Methode liefert entweder null bzw. eine Instanz der Klasse *ActionErrors* mit der Länge 0 zurück wenn keine Fehler gefunden wurden, oder der Rückgabewert ist eine Instanz der

Klasse *ActionErrors*, die *ActionError*-Objekte enthält wenn Fehler gefunden wurden. Die *ActionError*-Objekte enthalten die Schlüssel mit deren Hilfe die Fehlermeldungen aus der internationalisierten Nachrichten-Ressource gelesen werden.

Die Validierung in der *ActionForm*-Klasse ist optional. Die Standardimplementierung der *validate()*-Methode gibt null zurück. Neben einer Validierung in der Formular-Bean kann zudem innerhalb der *Action*-Klasse validiert werden. Es ist üblich eine einfache Validierung mit der *validate()*-Methode durchzuführen und darüber hinaus eine Validierung gegenüber der Geschäftslogik innerhalb des *Action*-Objektes zu implementieren.

3.6 Controller Komponenten

Der Controller ist der zentrale Teil innerhalb des Frameworks und zusammen mit den Struts Tag Libraries Hauptbestandteil des Frameworks. Der Struts-Controller hat die Aufgabe Requests vom Client entgegenzunehmen, zu entscheiden welche Funktion der Geschäftslogik auszuführen ist und dann an eine passende View-Komponente zu delegieren, welche für das Erzeugen der nächsten Sicht der GUI zuständig ist. In Struts ist die primäre Komponente des Controllers ein Servlet der Klasse *ActionServlet*. Dieses Servlet wird konfiguriert, indem eine Sammlung von Mappings (beschrieben durch eine Klasse vom Typ *ActionMapping*) definiert wird. Jedes Mapping definiert einen Pfad, auf den hin der eingehende Request-URI geprüft wird, sowie den voll qualifizierten Klassennamen einer *Action*-Klasse. Diese ist dafür verantwortlich die gewünschte Aktion der Geschäftslogik auszuführen, die Kontrolle an die passende View-Komponente weiterzugeben, um schließlich die Antwort für den Client zu erzeugen.

Struts erlaubt es logische Namen für Aktionen zu verwenden. So kann eine *Action*-Methode beispielsweise die "Login"-Seite aufrufen, ohne zu wissen, was der tatsächliche Name der JSP-Seite ist. Diese Features ermöglichen dem Entwickler die Steuerungslogik von der Darstellung zu entkoppeln.[1]

3.6.1 *Action* Klassen

Die *Action*-Klasse verarbeitet einen Request und gibt ein *ActionForward*-Objekt zurück, das diejenige View definiert, welche den Response generiert. Die Verarbeitung des Request geschieht in der vom Entwickler überschriebenen *perform()*-Methode der Klasse *Action*. Die *perform()*-Methode einer *Action*-Klasse kann folgende Funktionalität implementieren:

- Sicherheit: Überprüfen der Benutzer-Session. Wenn keine gültige Session-ID für den Benutzer existiert, kann an die Login-View weitergeleitet werden. Dies könnte der Fall sein, wenn der Benutzer versucht ohne Login und durch direkte Eingabe einer URL in die Applikation einzusteigen oder wenn die Session nach einer Zeitüberschreitung

[1] vgl. [@Stru3]

ungültig wird. Falls eine gültige Session-ID existiert, wird der Programmablauf normal weitergeführt.

- Validierung: Neben der Validierung der Formulardaten in der *ActionForm*-Bean kann auch in der *perform()*-Methode eine Validierung stattfinden. Dies sollte aber nur der Fall sein, wenn zur Validierung Geschäftslogik angesprochen werden muss.
- Geschäftslogik: Die *perform()*-Methode kann Geschäftslogik (z.B. Zugriffe auf eine Datenbank) enthalten. Geschäftslogik sollte aber nur bei kleineren Applikationen in der *Action*-Klasse enthalten sein. Besser ist es die Geschäftslogik in externe Klassen (auch EJBs) auszugliedern und diese Klassen durch Methodenaufrufe in der *Action*-Klasse anzusprechen.
- Kommunikation der Komponenten: Die *perform()*-Methode erhält die zu verarbeitenden Daten i.d.R. durch die *ActionForm*-Klasse und muss nach dem Ausführen der Geschäftslogik die Daten auch wieder in serverseitigen Objekten ablegen, so dass sie bei der Darstellung der nächsten View benutzt werden können. Dies geschieht entweder wieder mit Hilfe der *ActionForm*-Klasse oder in sogenannten HelperBeans, die im Request- oder Session-Gültigkeitsbereich abgelegt werden.
- Weiterleiten an eine View: Durch Zurückliefern eines *ActionForward*-Objektes wird diejenige JSP-Seite definiert, die den Response auf den Request generiert. Die Referenz auf ein solches Objekt erhält man, indem die Methode *findForward()* auf das *ActionMapping*-Objekt angewendet wird.

Bei der Implementierung der *perform()*-Methode sollten einige Design-Richtlinien beachtet werden:

- Das Controller-Servlet erzeugt nur eine Instanz der *Action*-Klasse und benutzt diese für alle Requests. Aus diesem Grund muss der Code in der *perform()*-Methode so implementiert werden, dass er in Multithread-Umgebungen korrekt arbeitet.[1]
- Exceptions, die z.B. durch Zugriffe auf Datenbanken oder andere Geschäftslogik-Klassen innerhalb der *perform()*-Methode entstehen, müssen korrekt abgefangen werden.
- Benutzte Ressourcen (z.B. Datenbankverbindungen) sollten sofort freigegeben werden. Dies sollte auch im Fehlerfall beachtet werden.
- Die *Action*-Klasse sollte nicht zu gross werden. Das Ausgliedern von Geschäftslogik in separate Klassen dient nicht nur der Wiederverwendbarkeit, sondern auch der Übersichtlichkeit.

[1] Wichtig ist das Benutzen von lokalen Variablen im Gegensatz zu Klassenvariablen. Lokale Variablen werden auf für jeden Request-Thread unterschiedlichen Stacks abgelegt.

3.6.2 Konfiguration der *ActionMapping* Klasse

Die Konfiguration des Struts-Controllers wird in der XML-Datei *struts-config.xml* definiert. Struts enthält ein Digester-Modul, welches die XML-basierte Beschreibung eines Mappings einliest und daraus ein *ActionMapping*-Objekt erzeugt.

Die Datei *struts-config.xml* muss vom Entwickler erzeugt und im WEB-INF Verzeichnis der Applikation abgelegt werden. Das Format dieses Dokuments wird definiert durch die Document Type Definition *http://jakarta.apache.org/struts/dtds/struts-config_1_0.dtd*. Das äusserste XML-Element muss <*struts-config*> lauten. Innerhalb dieses Bereichs gibt es folgende Elemente:

```
<form-beans>
...
</form-beans>
```

Dieser Bereich enthält die Form-Bean Definitionen. Die Attribute sind:

- name: Name des Attributes, mit dem dieses *ActionForm*-Bean im Request oder in der Session gespeichert wird.
- type: Der voll qualifizierte Java Klassenname des *ActionForm*-Bean.

```
<global-forwards>
...
</global-forwards>
```

Dieser Bereich enthält <*forward*>-Elemente, die JSP-Seiten logische Namen zuweisen. Diese Forwards sind global gültig. Sie können beispielsweise in der *perform()*-Methode jeder *Action*-Klasse benutzt werden. Die Attribute sind u.a.:

- name: Logischer Name.
- path: Dateiname der View.

```
<action-mappings>
...
</action-mappings>
```

Dieser Bereich enthält die Definitionen der Aktionen. Für jede Aktion wird ein separates <*action*>-Element verwendet. Die Attribute sind u.a.:

- path: Der auf den Kontext der Anwendung bezogene relative Pfad für die Aktion.
- type: Der voll qualifizierte Java Klassenname der *Action*-Klasse.
- name: Der Name des <*form-bean*>-Elementes, das mit dieser Aktion verwendet werden soll.
- scope: Der Gültigkeitsbereich, in dem das *ActionForm*-Bean existiert.

- input: Das JSP, von dem aus der Request abgesendet wurde und an das im Fehlerfall zurückgeleitet wird.

Die <*action*>-Elemente können zudem lokale Forwards enthalten, die nur innerhalb einer Aktion gültig sind. Dazu muss innerhalb des <*action*>-Bereichs ein <*forward*>-Element eingefügt werden, welches die gleichen Attribute hat wie die <*forward*>-Elemente innerhalb des <*global-forwards*>-Bereichs.

Es gibt ausserdem die Möglichkeit in einem <*data-sources*>-Bereich Datenquellen zu definieren. Struts bietet einen einfachen JDBC-Connection-Pool. Die Datenbankanbindung mit Struts-Komponenten ist nach Aussage der Entwickler aber nicht sonderlich ausgereift. Es wird empfohlen die Datenbankanbindung selbst zu entwickeln, bzw. die Funktionalität des Applikation-Servers zu nutzen. Bei der Entwicklung von ADB-Info kam die zweite Variante zum Einsatz.

3.6.3 Der Deployment-Descriptor der Webanwendung

Die Konfiguration des Servlet Containers, in dem die Webapplikation läuft, wird in der XML-Datei *web.xml* definiert. Die Datei *web.xml* muss vom Entwickler erzeugt und im WEB-INF Verzeichnis der Applikation abgelegt werden. Das Format dieses Dokuments wird definiert durch die Document Type Definition *http://java.sun.com/j2ee/dtds/web-app_2_2.dtd*. Das äusserste XML-Element muss <*web-app*> lauten. Innerhalb dieses Bereichs gibt es folgende Elemente:

- <*servlet*>: Definition des Struts-Controllers.
- <*servlet-mapping*>: Konfiguration des *Action*-Servlet Mappings.
- <*taglib*>: Definition der Struts Tag-Libraries.

Definition des Struts-Controllers

```
<servlet>
...
</servlet>
```

Dieser Bereich definiert das *Action*-Servlet, welches die Requests entgegennimmt und enthält folgende Elemente:

```
<servlet-name>action</servlet-name>
```

action ist der Name des Servlets.

```
<servlet-class>
  org.apache.struts.action.ActionServlet
</servlet-class>
```

org.apache.struts.action.ActionServlet ist der qualifizierende Java Klassenname des Struts Controllers.

```
<load-on-startup>2</load-on-startup>
```

<load-on-startup> gibt an, dass das Controller-Servlet bereits beim Start des Containers geladen werden soll.

```
<init-param>
  <param-name>config</param-name>
  <param-value>/WEB-INF/struts-config.xml</param-value>
</init-param>
```

Das Controller-Servlet kann mit diversen Initialisierungsparametern konfiguriert werden: (in eckigen Klammern ist der Standardwert angegeben)

- *application*: Java Klassenname der Anwendungsressource. [NONE]
- *bufferSize*: Die Größe des Eingabebuffers, der bei Datei-Uploads benutzt wird. [4096]
- *config*: Kontextrelativer Pfad auf die XML-Ressourcen, welche die Konfigurations-informationen enthalten. [/WEB-INF/struts-config.xml]
- *content*: Standard Content-Typ und Zeichenkodierung für alle Responses. Kann überschrieben werden durch ein Forwarded-To-Servlet oder eine JSP-Seite. [text/html]
- *debug*: Debug-Level. Definiert, welche Informationen protokolliert werden. [0]
- *detail*: Debug-Level für den Digester. [0]
- *factory*: Java Klassenname der *MessageResourcesFactory*, die dazu verwendet wird, das *MessageResources*-Objekt der Anwendung zu erzeugen.
- *formBean*: Java Klassenname der zu verwendenden *ActionForm*-Bean Implementation [org.apache.struts.action.ActionFormBean].
- *forward*: Java Klassenname der zu verwendenden *ActionForward* Implementation [org.apache.struts.action.ActionForward].
- *locale*: Wenn der Wert true ist und eine Session existiert, wird ein *java.util.Locale*-Objekt in der Session gespeichert.
- *mapping*: Java Klassenname der zu verwendenden *ActionMapping* Implementation [org.apache.struts.action.ActionMapping]
- *maxFileSize*: die maximale Größe (in Bytes) einer Datei, die beim Datei-Upload akzeptiert wird. K, M, oder G werden als Kilobytes, Megabytes, oder Gigabytes intepretiert. [250M]
- *multipartClass*: Der voll qualifizierte Name der zu verwendenden *MultiplartRequestHandler* Implementation, die Datei-Uploads verarbeitet. [org.apache.struts.upload.DiskMultipartRequestHandler]
- *nocache*: Wenn der Wert true ist, werden HTTP-Header für jeden Response gesetzt, damit das Zwischenspeichern von generierten oder weitergeleiteten Responses im Browsers vermieden wird. [false]

- *null*: Wenn der Wert true ist, geben die Applikations-Ressourcen null zurück, falls ein unbekannter Nachrichtenschlüssel verwendet wird. Andernfalls wird eine Fehlermeldung zurückgegeben, die den Nachrichtenschlüssel enthält. [true]
- *tempDir*: Das zu verwendende temporäre Arbeitsverzeichnis beim Verarbeiten von Datei-Uploads.
- *validate*: true, wenn das neue Format für die Konfigurationsdatei benutzt wird. [true]

Konfiguration des *Action*-Servlet Mappings

Die Konfiguration des *Action*-Servlet Mappings beschreibt in welchen Fällen der Struts-Controller eine Anfrage entgegennimmt und einen Response generiert. Es gibt zwei verschiedene Möglichkeiten zu definieren welche URLs vom Controller verarbeitet werden: Präfix-Matching und Extension-Matching:

Präfix-Matching leitet alle URLs, die nach dem Kontextpfad der Applikation mit einem bestimmten Wert anfangen, an den Controller weiter. Folgendes Beispiel veranschaulicht dies:

```
<servlet-mapping>
   <servlet-name>action</servlet-name>
   <url-pattern>/execute/*</url-pattern>
</servlet-mapping>
```

Dieser Eintrag in der Datei web.xml würde, wenn der Kontextpfad der Web-Applikation /myapplication ist, bei folgendem Aufruf den Request an den Struts-Controller weiterleiten.

```
http://www.mycompany.com/myapplication/execute/logon
```

Extension-Matching leitet alle URLs, die mit einem Punkt und einer frei wählbaren Endung enden, an den Controller weiter. Folgendes Beispiel veranschaulicht dies.

```
<servlet-mapping>
   <servlet-name>action</servlet-name>
   <url-pattern>*.do</url-pattern>
</servlet-mapping>
```

Dieser Eintrag in der Datei web.xml würde, wenn der Kontextpfad der Web-Applikation /myapplication ist, bei folgendem Aufruf den Request an den Struts-Controller weiterleiten.

```
http://www.mycompany.com/myapplication/logon.do
```

Definition der Struts Tag-Libraries

Die Struts Tag-Libraries werden folgendermassen definiert:

```
taglib>
   <taglib-uri>/WEB-INF/struts-bean.tld</taglib-uri>
```

```
   <taglib-location>/WEB-INF/struts-bean.tld</taglib-location>
</taglib>

<taglib>
   <taglib-uri>/WEB-INF/struts-html.tld</taglib-uri>
   <taglib-location>/WEB-INF/struts-html.tld</taglib-location>
</taglib>

<taglib>
   <taglib-uri>/WEB-INF/struts-logic.tld</taglib-uri>
   <taglib-location>/WEB-INF/struts-logic.tld</taglib-location>
</taglib>

<taglib>
   <taglib-uri>/WEB-INF/struts-template.tld</taglib-uri>
   <taglib-location>/WEB-INF/struts-template.tld</taglib-location>
</taglib>
```

Definiert werden müssen nur diejenigen Tag-Libraries, die in den JSPs benutzt werden.

3.7 Struts Klassen- und Sequenzdiagramm (UML)

Dieses Kapitel beschreibt das Zusammenspiel der Klassen des Frameworks. Zum Entwickeln einer einfachen Struts Applikation ist das Wissen darüber nicht unbedingt erforderlich. Allerdings vervollständigen die Diagramme das nach der Lektüre der ersten Kapitel entstandene Bild über Struts. Der Leser erhält einen Eindruck über die interne Arbeitsweise des Frameworks.

Folgende Abbildung zeigt das UML Klassendiagramm der Struts-Klassen im Package org.apache.struts.action (weiss dargestellt). In der Abbildung blau dargestellt sind diejenigen Klassen, die der User selbst erstellen muss (User Klassen).

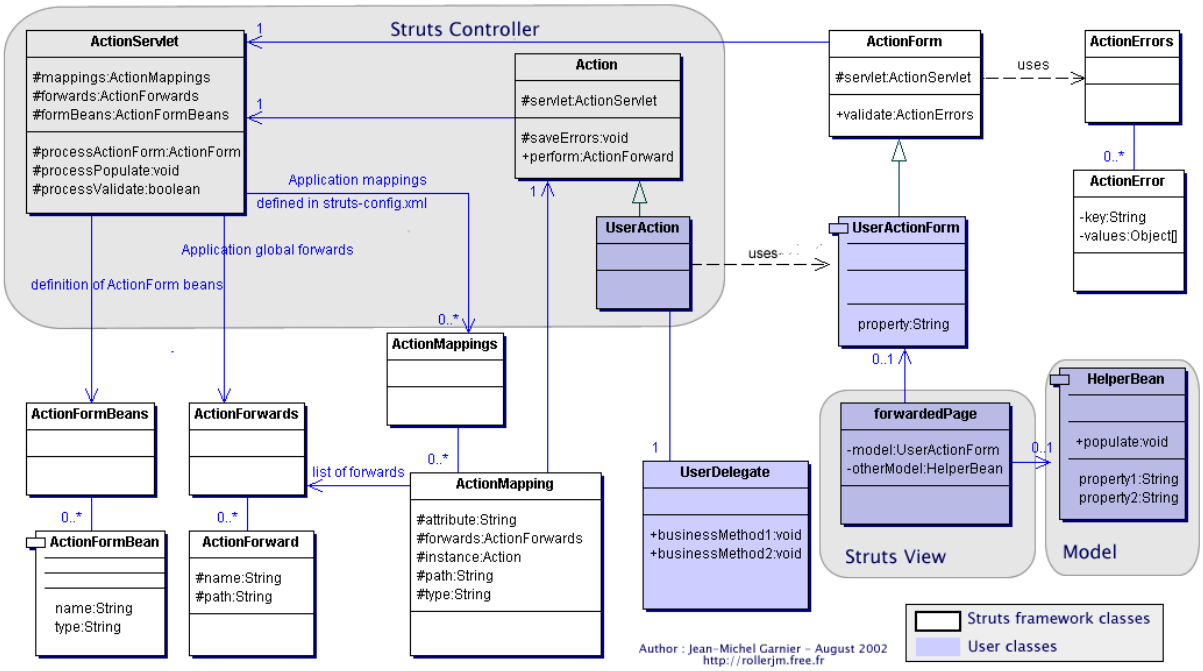

Abbildung 3-3: UML Klassendiagramm der Struts-Klassen im Package org.apache.struts.action[1]

[1] [@Stru4]

Die oben abgebildeten Klassen haben folgende Funktion: (die internen Methoden von Struts werden ebenfalls beschrieben)

Struts Klassen

- Struts Conroller:
  Die *ActionServlet*-Klasse ist Kern des Frameworks. Die Klasse ist ein Servlet, welches Requests entgegennimmt und diese in Abhängigkeit von der Konfigurationsdatei *struts-config.xml* weiterleitet.
  Bei der Initialisierung der Applikation wird die Konfigurationsdatei *struts-config.xml* geparst und die darin enthaltenen Informationen in Instanzen der Klassen *ActionMapping*, *ActionForward* und *ActionFormBean* abgelegt.
    - Die Methode *processActionForm()* initialisiert die mit einem Mapping verbundene *ActionForm*-Klasse bzw. erzeugt eine, falls sie noch nicht existiert.
    - Die Methode *processPopulate()* füllt die Properties in der *ActionForm*-Klasse mit den Formulardaten aus dem Request-Parameter des aktuellen Request.
    - Die Methode *processValidate()* ruft die *validate()* Methode der *ActionForm*-Klasse auf und ruft im Fehlerfall die letzte View wieder auf.

  Die *Action*-Klasse wird von *ActionServlet* erzeugt und benutzt. Die *Action*-Klasse muss abgeleitet und die *perform()* Methode überschrieben werden. Von dieser Methode aus kann dann die Geschäftslogik angesprochen werden.
    - Die Methode *perform()* verarbeitet den HTTP-Request und erzeugt den HTTP-Response. Sie liefert eine Instanz von *ActionForward* zurück, welche beschreibt wohin und wie die Kontrolle weitergeleitet werden soll.
    - Die Methode *saveErrors()* sichert evtl. vorhandene Fehlermeldungen in den zugehörigen Request.
- *ActionForm*:
  Die *ActionForm*-Klasse enthält die Formulardaten eines HTML-Formulars. Struts extrahiert die Werte automatisch aus dem HttpServletRequest und schreibt sie mit Hilfe der Setter-Methoden in die *ActionForm*-Klasse.
- Mapping Management:
    - Die Klasse *ActionMappings* enthält eine Collection von *ActionMapping*-Objekten.
    - Die Klasse *ActionMapping* enthält Informationen über die Aktionen. Die Informationen kommen aus der Konfigurationsdatei *struts-config.xml (<action path="/user" type=.xxx.UserAction" name="userForm" scope="session">*).
        - *path* definiert die Request-URI, mit der eine Aktion aufgerufen wird.
        - *type* ist der Name der *Action*-Klasse, welche die Aktion ausführt.
        - *name* ist der Name der *ActionForm*-Bean, welche mit der Aktion in Zusammenhang steht.

        - *scope* definiert den Gültigkeitsbereich, in dem die *ActionForm*-Bean erstellt wird.
    - Die Klasse *ActionForwards* enthält eine Collection von *ActionForward*-Objekten.
    - Die Klasse *ActionForward* enthält Informationen darüber, wohin der Benutzer nach der Aktion weitergeleitet wird. Die Informationen kommen aus der Konfigurationsdatei *struts-config.xml*
      (<forward name="next" path="/forwardedPage.jsp" />)
- *ActionForm* Bean Management:
    - Die Klasse *ActionFormBeans* enthält eine Collection von *ActionFormBean*-Objekten.
    - Die Klasse *ActionFormBean* ordnet einem logischen Namen eine *ActionForm*-Klasse zu. Die Informationen darüber kommen aus der Konfigurationsdatei *struts-config.xml*
      (<form-bean name="userForm" type="xxx.UserActionForm" />)
- Error Management:
    - Die Klasse *ActionErrors* enthält eine Collection von *ActionError*-Objekten.
    - Die Klasse *ActionError* enthält Informationen über Fehlermeldungen.

User Klassen

- *UserAction*: Erbt von *Action*. Fachlicher Controller. Die überschriebene *perform()*-Methode enthält die Geschäftslogik bzw. ruft diese auf.
- *UserActionForm*: Erbt von *ActionForm*. Speichert die Formulardaten. Die überschriebene *validate()*-Methode validiert die Formulardaten.
- *HelperBean*: Model-Klasse zum Speichern von Daten. (optional)
- *UserDelegate*: Aufruf von Geschäftslogik-Klassen. Oft wird das J2EE-Entwurfsmuster "Business Delegate" eingesetzt. (optional)
- *forwardedPage* ist keine Klasse, sondern eine JSP zum Darstellen der View.[1]

Das UML Sequenzdiagramm auf der folgenden Seite zeigt die Abläufe von einem Request an den Struts-Controller bis zur fertigen JSP-Antwortseite.

[1] vgl. [@Stru4]

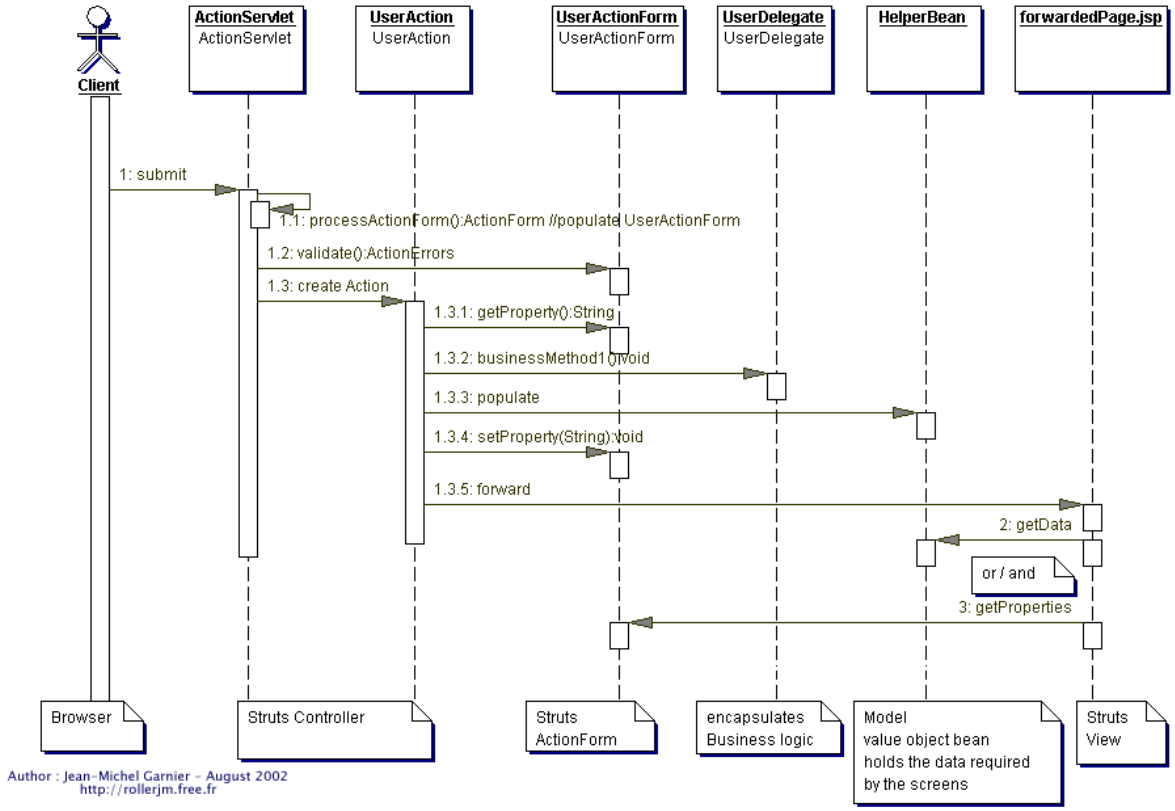

Abbildung 3-4: Struts UML Sequenzdiagramm[1]

Am Anfang steht ein Request (i.d.R. Abschicken eines Formulars im Browser) an den Struts-Controller *ActionServlet* (1). Der Struts-Controller legt zuerst die Formulardaten in der *UserActionForm* Bean ab (1.1), validiert danach die eingegebenen Daten und generiert evtl. Fehlermeldungen (1.2) und erzeugt als Drittes eine Instanz der *UserAction*-Klasse (1.3). Die *UserAction*-Klasse definiert Aktionen in der *perform()*-Methode. Die *UserAction*-Klasse kann nun Daten aus der *UserActionForm* Bean lesen (1.3.1) und weitere Geschäftslogik-Klassen ansprechen (1.3.2). Der Datenaustausch zwischen *UserAction*-Klasse und der am Ende aufgerufenen View kann auf zwei verschiedene Arten erfolgen. Entweder mit Hilfe einer *HelperBean* oder über die *UserActionForm* Bean. Eine Kombination beider Möglichkeiten ist auch möglich. Das Schreiben von Daten in die *HelperBean* geschieht in Schritt (1.3.3) während in (1.3.4) die *UserActionForm* Bean aktualisiert wird. Am Ende leitet die *UserAction*-Klasse an eine in der Konfigurationsdatei *struts-config.xml* definierte View (*forwardedPage.jsp*) weiter (1.3.5). Diese View bezieht nun die anzuzeigenden Daten aus der *HelperBean* (2) und/oder aus der *UserActionForm* Bean (3).[2]

[1] [@Stru4]
[2] vgl. [@Stru4]

3.8 Struts an einem Beispiel

In diesem Kapitel wird eine einfache Struts Applikation Schritt für Schritt entwickelt und dem Leser das Verstehen des Frameworks erleichtert. Die Anwendung besteht aus einem kleinen Formular, in das der Name und die Anrede eingetragen werden kann. Beim Klicken auf den Absenden-Button ruft sich die Seite selbst auf, zeigt die eingegebenen Daten im Formular an und begrüsst den Anwender mit der Anrede und seinem Namen. Für das Ausführen der Applikation benutze ich den Servlet Container Apache Tomcat 4.0.[1] Der Quellcode des Beispiels befindet sich im Anhang A.1. auf der CD. Sämtliche benötigte Software befindet sich im Anhang B. auf der CD.

Beim Durcharbeiten der Anleitung sollte der Leser immer die offizielle Struts Dokumentation zur Hand haben.[2] Um die Übersichtlichkeit zu bewahren werden hier nicht alle Details beschrieben. Java-, JSP- und HTML-Kenntnisse sowie die Lektüre der vorangegangenen Kapitel werden vorausgesetzt.

Abbildung 3-5: Die Beispiel-Applikation

3.8.1 Schritt 1: Downloaden und Installieren von Struts

Struts kann auf der Homepage des Jakarta Projekts heruntergeladen werden[3]. Ich benutze für die Entwicklung dieses Beispiels die Version 1.0.2. Voraussetzung für das Kompilieren der selbstgeschriebenen Klassen ist ein installiertes Java Development Kit (JDK) der Version 1.3 oder höher. Ausserdem sollte das J2EE Software Development Kit (SDK) der Version 1.3.1 oder höher installiert sein. Dies enthält den von Struts benötigten Java API for XML Parsing (JAXP) kompatiblen XML-Parser und die *javax.servlet*-Klassen.

Wenn man das Struts-Archiv entpackt und in das Verzeichnis *webapps* wechselt, findet man eine Datei mit dem Namen *struts-blank.war*. Dies ist eine "leere" Struts Applikation mit allen Verzeichnissen und Dateien, die zum Entwickeln einer Struts-Anwendung benötigt werden. Das ist ein idealer Startpunkt für das erste Struts-Projekt. Mit WinZip lässt sich die .war Datei

[1] Download von Tomcat und Installationsanleitung auf [@Tomc] oder im Anhang B.3.
[2] [@Stru3]
[3] [@Stru] oder Anhang B.1. (die Binärversion reicht zum Entwickeln dieses Beispiels aus)

in das *webapps*-Verzeichnis der Tomcat Installation entpacken. Den Namen des Verzeichnisses kann man selbstverständlich beliebig ändern. Ich habe anstatt *struts-blank* *struts-beispiel* gewählt. Nach dem Neustart von Tomcat kann man die Applikation mit dem Browser starten, indem man folgendes eingibt: *http://localhost:8080/struts-beispiel/*. Wenn alles geklappt hat, sollte Hello World! im Browser angezeigt werden.

Ein zweiter Weg zum Installieren von Web-Applikationen ist das einfache Kopieren einer .war Datei in das *webapps*-Verzeichnis der Tomcat Installation. Tomcat entpackt bei einem Neustart automatisch die .war Datei in einen gleichnamigen Ordner. Die Dateien *struts-documentation.war* und *struts-example.war* im *webapps*-Verzeichnis des Struts-Archiv enthalten die komplette Struts Dokumentation bzw. eine weitere Beispielanwendung. Es lohnt sich diese zu installieren.

Die einfache Installation zeigt, dass Struts keine Modifikation am Servlet Container vornimmt, sondern als eigenständige, normale Servlet/JSP-Applikation angesehen werden kann.

3.8.2 Schritt 2: Die Datei- und Verzeichnisstruktur von Struts

Folgende Abbildung zeigt die Verzeichnisse und Dateien der Standard Struts-Applikation *struts-blank.war*:

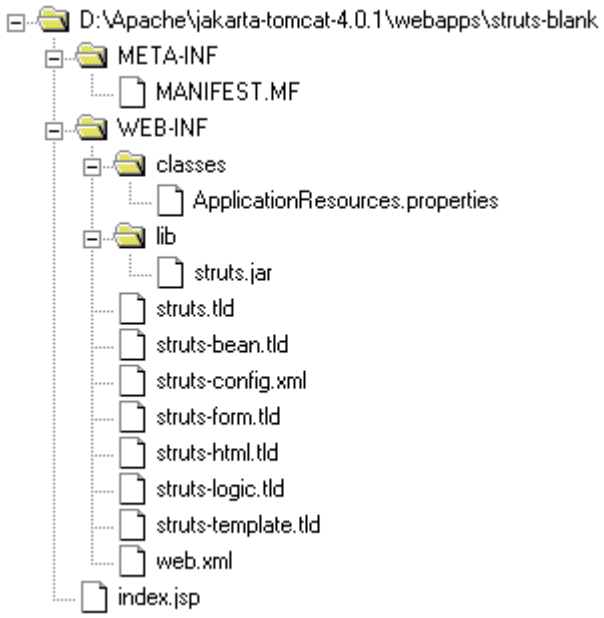

Abbildung 3-6: Die Struts Verzeichnisstruktur

Die Verzeichnisse und Dateien haben folgende Funktionen:

Verzeichnis- oder Dateiname	**Zweck**
META-INF	Meta-Informationen. Wird u.a. von

	Entwicklungswerkzeugen benötigt.
WEB-INF/classes	In diesem Verzeichnis werden die selbst entwickelten Java-Klassen abgelegt.
WEB-INF/classes/ ApplicationResources.properties	In dieser Datei stehen alle Nachrichten und Fehlermeldungen, die von den Views ausgegeben werden können.
WEB-INF/lib/struts.jar	Dieses Java-Archiv beinhaltet u.a. das Struts Controller-Servlet, alle weiteren Struts Klassen und die Klassen der Struts Tag Libraries.
WEB-INF/*.tld	Definitionen der Struts Tag Libraries.
WEB-INF/struts-config.xml	Konfigurationsdatei von Struts.
WEB-INF/web.xml	Konfigurationsdatei des Servlet Containers.
index.jsp	Im Root-Verzeichnis werden die JSP- und HTML-Dateien abgelegt.

3.8.3 Schritt 3: Erstellen von web.xml

Die Datei *web.xml* ist die Konfigurationsdatei für den Servlet-Container.

```
12  <?xml version="1.0" encoding="ISO-8859-1"?>
13
14  <!DOCTYPE web-app
15    PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.2//EN"
16    "http://java.sun.com/j2ee/dtds/web-app_2_2.dtd">
17
18  <web-app>
19
20    <!-- Standard Action Servlet Configuration (with debugging) -->
21    <servlet>
22      <servlet-name>action</servlet-name>
23      <servlet-class>
24        org.apache.struts.action.ActionServlet
25      </servlet-class>
26      <init-param>
27        <param-name>application</param-name>
28        <param-value>ApplicationResources</param-value>
29      </init-param>
30      <init-param>
31        <param-name>config</param-name>
32        <param-value>/WEB-INF/struts-config.xml</param-value>
33      </init-param>
34      <init-param>
35        <param-name>debug</param-name>
36        <param-value>2</param-value>
37      </init-param>
38      <init-param>
39        <param-name>detail</param-name>
40        <param-value>2</param-value>
41      </init-param>
42      <init-param>
43        <param-name>validate</param-name>
44        <param-value>true</param-value>
45      </init-param>
46      <load-on-startup>2</load-on-startup>
47    </servlet>
48
49    <!-- Standard Action Servlet Mapping -->
50    <servlet-mapping>
51      <servlet-name>action</servlet-name>
52      <url-pattern>*.do</url-pattern>
53    </servlet-mapping>
54
```

```
55    <!-- The Usual Welcome File List -->
56    <welcome-file-list>
57      <welcome-file>index.jsp</welcome-file>
58    </welcome-file-list>
59
60    <!-- Struts Tag Library Descriptors -->
61    <taglib>
62      <taglib-uri>/WEB-INF/struts-bean.tld</taglib-uri>
63      <taglib-location>/WEB-INF/struts-bean.tld</taglib-location>
64    </taglib>
65
66    <taglib>
67      <taglib-uri>/WEB-INF/struts-html.tld</taglib-uri>
68      <taglib-location>/WEB-INF/struts-html.tld</taglib-location>
69    </taglib>
70
71    <taglib>
72      <taglib-uri>/WEB-INF/struts-logic.tld</taglib-uri>
73      <taglib-location>/WEB-INF/struts-logic.tld</taglib-location>
74    </taglib>
75
76  </web-app>
```

Listing 3-2: web.xml

Die XML-Datei besteht aus vier Teilen:

- Zeilen 10 bis 36: Die Definition des Struts Controller-Servlets *ActionServlet.*
- Zeilen 38 bis 42: Das URL Mapping für die Aufrufe des Servlets.
- Zeilen 44 bis 47: Die Welcome-File Definition. Sie definiert welche Datei standardmässig aufgerufen werden soll.
- Zeilen 49 bis 63: Die Definition der Struts Tag Librarys.

Wie man in der Datei oben erkennt, wird bei einem Browser-Aufruf von *irgendwas.do* das Struts Controller Servlet aufgerufen. Diese Aufrufe werden i.d.R. in einem Formular-Tag gemacht. Woher Struts weiss was es nach dem Aufruf tun muss, wird weiter unten beschrieben.

3.8.4 Schritt 4: Erstellen von struts-config.xml

Die Datei *struts-config.xml* ist die Konfigurationsdatei des Struts-Controllers. Hier werden die .jsp-Dateien (also die Views) den *Action*- und *ActionForm*-Klassen zugeordnet. Diese beiden Klassen haben folgende Aufgabe:

ActionForm-Klasse: Nach dem Initiieren einer Struts Aktion (z.B. Betätigen eines Submit-Button in einem Formular mit dem Parameter *action="irgendwas.do*") sichert der Struts-Controller als erstes die in das Formular eingegebenen Werte in einer Java Bean, die von der Struts *ActionForm*-Klasse abgeleitet wird. Dieses *ActionForm*-Bean dient gewissermassen als Puffer zwischen Browser und Datenbank und kann zudem die Benutzereingaben validieren. Der Entwickler muss diese Klasse selbst erstellen.

Action-Klasse: Nach dem Sichern der Formulardaten wird vom Struts-Controller eine weitere Bean aufgerufen, nämlich die *Action*-Bean. Diese Klasse wird von der Struts *Action*-Klasse abgeleitet und muss ebenfalls vom Entwickler erstellt werden. Sie definiert die Funktionalität der Anwendung. Innerhalb dieser Bean kann auf die Formulardaten (*ActionForm*-Klassen) zugegriffen werden. Hier wäre auch der Ort, an dem weitere Java Beans bzw. auch EJBs (also

die Geschäftslogik) aufgerufen werden könnten, bzw. Datenbankverbindungen aufgebaut würden. (siehe Schritt 7)

Die *Action*- bzw. *ActionForm*-Klassen sollten einer Namenskonvention entsprechen:

Klassentyp	**Name**
ActionForm	<aktion>Form, wobei <aktion> der Aktion-Pfadname (path) in der Datei *struts-config.xml* ist
Action	<aktion>Action

Folgendes Listing zeigt eine Struts XML-Konfigurationsdatei. In Zeile 10 bis 13 wird dem Formular-Bean ein logischer Name zugeordnet. In den Zeilen 16 bis 25 wird eine Struts-Aktion definiert. Der Name der Aktion ist *submit* (*path="/submit"*). *type="beispiel.SubmitAction"* gibt an, welche Action-Klasse aufgerufen werden soll, *name="submitForm"* definiert das zu benutzende *ActionForm*-Bean und *input="/index.jsp"* definiert die View, welche evtl. Fehlermeldungen anzeigt. Durch scope="request" wird festgelegt, wie lange die *ActionForm*-Bean existiert. Die Forward-Tags definieren an welche Views der Controller nach dem Abarbeiten der *Action*-Klasse weiterleiten soll (siehe auch Schritt 7). In unserem Fall besteht die Anwendung aus nur einer View und kehrt in jedem Fall dahin zurück.

```
01  <?xml version="1.0" encoding="ISO-8859-1" ?>
02
03  <!DOCTYPE struts-config PUBLIC
04   "-//Apache Software Foundation//DTD Struts Configuration 1.0//EN"
05   "http://jakarta.apache.org/struts/dtds/struts-config_1_0.dtd">
06
07  <struts-config>
08
09    <!-- ========== Form Bean Definitions ================= -->
10    <form-beans>
11      <form-bean      name="submitForm"
12                      type="beispiel.SubmitForm"/>
13    </form-beans>
14
15    <!-- ========== Action Mapping Definitions ============ -->
16    <action-mappings>
17      <action   path="/submit"
18                type="beispiel.SubmitAction"
19                name="submitForm"
20                input="/index.jsp"
21                scope="request">
22      <forward name="success" path="/index.jsp"/>
23      <forward name="failure" path="/index.jsp"/>
24      </action>
25    </action-mappings>
26
27  </struts-config>
```

Listing 3-3: struts-config.xml

Es ist wichtig zu erkennen, dass bei der Entwicklung mit Struts keine Datei- oder Klassennamen in die JSPs oder die *Action*-/*ActionForm*-Klassen fest einprogrammiert werden. Alle Zusammenhänge zwischen den Klassen und JSPs werden in der Datei *struts-config.xml* definiert. Dies wird auch "Lose Kopplung" genannt und ist eine wesentliche Eigenschaft von Struts.

3.8.5 Schritt 5: Erstellen der JSP-Seite

Am Anfang der JSP werden die benötigten Struts Tag-Libraries definiert. Mit diesen Tag-Libraries kann der View-Designer sämtliche Struts-Funktionalität nutzen, ohne eine einzige Zeile Java-Code schreiben zu müssen. Die in diesem Beispiel eingesetzten Tags werden nun kurz beschrieben. Eine ausführliche Beschreibung aller Struts-Tags befindet sich in [@Stru3].

```
01  <%@ page language="java" %>
02  <%@ taglib uri="/WEB-INF/struts-bean.tld" prefix="bean" %>
03  <%@ taglib uri="/WEB-INF/struts-html.tld" prefix="html" %>
04  <%@ taglib uri="/WEB-INF/struts-logic.tld" prefix="logic" %>
05
06  <html:html locale="true">
07
08  <head>
09  <title><bean:message key="index.title"/></title>
10  <html:base/>
11  </head>
12
13  <body bgcolor="white">
14
15  <h3><bean:message key="index.heading"/></h3>
16
17  <html:errors/>
18
19  <html:form action="submit.do">
20    <p><bean:message key="index.name"/>:
21    <html:text property="name"/>
22    </p>
23    <p><bean:message key="index.anrede"/>:
24    <html:radio property="anrede" value="f"/> <bean:message key="index.frau"/>
25    <html:radio property="anrede" value="h"/> <bean:message key="index.herr"/>
26    <br>
27    </p>
28    <html:submit>
29      <bean:message key="button.submit"/>
30    </html:submit>
31  </html:form>
32
33  <logic:present name="name" scope="request">
34    <bean:message key="index.hallo"/>
35    <logic:equal name="submitForm" property="anrede" value="f">
36      <bean:message key="index.frau"/>
37    </logic:equal>
38    <logic:equal name="submitForm" property="anrede" value="h">
39      <bean:message key="index.herr"/>
40    </logic:equal>
41    <bean:write name="name" scope="request"/>
42  </logic:present>
43
44  </body>
45  </html:html>
```

Listing 3-4: index.jsp

In Zeile 6 wird die Internationalisierung von Struts aktiviert. Der Tag *<bean:message key="index.title"/>* in Zeile 9 erzeugt den Satz "Struts Beispiel", den Struts aus der Message-Ressource *ApplicationResources.properties* bezieht.

```
01  index.title=Struts Beispiel
02
03  index.heading=Struts Beispiel
04
05  index.name=Name
06
07  index.anrede=Anrede
08  index.frau=Frau
09  index.herr=Herr
10  index.hallo=Hallo
11
12  button.submit=Absenden
```

```
13
14  errors.header=<h4>Fehler</h4><ul>
15  errors.footer=</ul><hr>
16
17  error.name=<li>Bitte Name eingeben
18  error.anrede=<li>Bitte Anrede eingeben
```

Listing 3-5: ApplicationResources.properties

Das Tag in Zeile 17 würde Fehlermeldungen ausgeben, sobald in der Struts Java-Collection für Fehlermeldungen Elemente vorhanden wären. Dies ist der Fall, wenn die Validierung der Formulardaten in der *ActionForm*-Klasse fehlschlagen würde.

Damit das Speichern der Formularwerte in der *ActionForm*-Bean funktioniert, müssen für den Aufbau des Formulars in den Zeilen 19 bis 31 die *Form*-Tags von Struts benutzt werden. Auch innerhalb des *Form*-Tag werden die auszugebenden Texte aus der Message-Ressource gelesen. Sobald das Formular durch Betätigen des *Absenden*-Button an den Server geschickt wird, schreibt der Struts-Controller die Properties *name* und *anrede* in die *ActionForm*-Klasse und validiert diese.

In den Zeilen 33 bis 42 wird die Begrüssung mit Name und Anrede generiert. Da die Ausgabe nur erfolgen soll, nachdem korrekte Daten in das Formular eingegeben wurden, werden hier einige Struts *Logic*-Tags benutzt. Diese entsprechen einer If-Anweisung in Java. *<logic:present name="name" scope="request">* schaut, ob die Property *name* in der zur JSP gehörenden *ActionForm*-Klasse vorhanden ist. Ist dies der Fall, wird der Body des Tags ausgeführt, also die Begrüssung ausgegeben. Mit *<logic:equal name="submitForm" property="anrede" value="f">* wird überprüft, ob die Property *anrede* gleich "f" ist. Wenn das der Fall ist, wird "Frau" ausgegeben, ansonsten "Herr".

Obwohl diese eine JSP-Seite sehr viel Funktionalität beinhaltet, wird keinerlei Java-Code benötigt. Sämtliche Funktionen sind durch Benutzen der Struts-Tags implementiert. Es fällt auf, dass keine einzige Ausgabe auf der Seite direkt im JSP-Code steht. Alles wird aus der Message-Ressource von Struts gelesen.

3.8.6 Schritt 6: Entwickeln der ActionForm-Klasse

Diese Klasse wird von der Struts Klasse *ActionForm* abgeleitet und speichert die Formulardaten des zugehörigen JSPs. Damit dies funktioniert, muss die Klasse zwei Properties mit den Namen *name* und *anrede* und die dazugehörenden Setter- und Getter-Methoden implementieren. Würde beispielsweise *name* mit dem Wert "Mustermann" initialisiert werden, würde Struts dies zur Laufzeit automatisch in das Formular eintragen. Es können in dieser Klasse also auch Formularfelder initialisiert werden.

```
01  package beispiel;
02
03  import javax.servlet.http.HttpServletRequest;
04  import org.apache.struts.action.*; // Struts
05
```

```
06  public final class SubmitForm extends ActionForm {
07
08    // Formularfelder
09    private String name = "";
10    private String anrede = "";
11
12    // Getter und Setter der Formularfelder
13    public String getName() {
14      return (this.name);
15    }
16    public void setName(String name) {
17      this.name = name;
18    }
19
20    public String getAnrede() {
21      return (this.anrede);
22    }
23    public void setAnrede(String anrede) {
24      this.anrede = anrede;
25    }
26
27    // Formularfelder zuruecksetzen
28    public void reset(ActionMapping mapping, HttpServletRequest request) {
29
30      name = "";
31      anrede = "";
32    }
33
34    // Auf Fehler ueberpruefen
35    public ActionErrors validate(ActionMapping mapping, HttpServletRequest request) {
36
37      ActionErrors errors = new ActionErrors();
38
39      if (name == null || name.equals("")) { // Name eingegeben?
40        errors.add("Name", new ActionError("error.name"));
41      }
42      if (anrede == null || anrede.equals("")) { // Anrede eingegeben?
43        errors.add("Anrede", new ActionError("error.anrede"));
44      }
45      return errors;
46      }
47  }
```

Listing 3-6: SubmitForm.java

Die Methode *reset()* initialisiert die Properties bei einem Neustart der Applikation und die Methode *validate()* überprüft die Eingaben des Benutzers. In diesem Beispiel wird lediglich überprüft, ob überhaupt etwas eingegeben wurde und im Fehlerfall der Struts *errors*-Collection ein Struts *ActionError* hinzugefügt. Die Fehlermeldung wird ebenfalls aus der Message-Ressource gelesen und kann somit ebenfalls internationalisiert werden.

Die kompilierte Datei *SubmitForm.class* muss in das Verzeichnis WEB-INF/classes kopiert werden.

3.8.7 Schritt 7: Entwickeln der Action-Klasse

Die *Action*-Klassen sind aus der Sicht des Entwicklers das Herz der Applikation. In diesen Klassen wird die eigentliche Geschäftslogik aufgerufen. Sie sollten so klein wie möglich gehalten werden und lediglich andere Java Beans oder gar EJBs aufrufen, die z.B. eine Datenbank ansprechen und die Geschäftslogik ausführen.

Die Klasse *SubmitAction* wird von der Struts *Action*-Klasse abgeleitet und muss eine Methode mit dem Namen *perform()* implementieren. Die Methode bekommt u.a. den Request und die *ActionForm* übergeben und hat somit Zugriff auf alle relevanten Daten.

```
01  package beispiel;
02
03  import javax.servlet.http.*;
04  import org.apache.struts.action.*; // Struts
05
06  public final class SubmitAction extends Action {
07
08    // Auszufuehrende Aktion definieren
09    public ActionForward perform(ActionMapping mapping,
10                                 ActionForm form,
11                                 HttpServletRequest request,
12                                 HttpServletResponse response) {
13
14      SubmitForm f = (SubmitForm) form; // Form Bean holen
15      String name = f.getName(); // Name holen ...
16      // ... und veraendert im request abspeichern
17      request.setAttribute("name", name.toUpperCase());
18
19      // zum success Ziel weiterleiten
20      return (mapping.findForward("success"));
21    }
22  }
```

Listing 3-7: SubmitAction.java

In diesem Beispiel wird lediglich das Property *name* aus dem *ActionForm*-Bean gelesen, in Grossbuchstaben umgewandelt und im Request abgespeichert. Danach wird zum *success*-Ziel weitergeleitet. Die Konfigurationsdatei *struts-config.xml* definiert als *success*-Ziel das View *index.jsp*.

Da Struts nur eine Instanz der Action-Klasse erstellt und alle Sessions die gleiche Instanz benutzen, sollten keine Klassenvariablen, sondern nur lokale Variablen benutzt werden.

Die kompilierte Datei *SubmitAction.class* muss in das Verzeichnis WEB-INF/classes kopiert werden.

3.8.8 Schritt 8: Testen der Applikation

Zuerst sollte Tomcat neu gestartet werden. Dies sollte immer geschehen, nachdem eine Java-Klasse, eine JSP oder eine Konfigurationsdatei geändert oder hinzugefügt wurde. Nach dem Neustart von Tomcat kann man die Applikation mit dem Browser starten, indem man folgendes eingibt: *http://localhost:8080/struts-beispiel/*. Wenn nun der Absenden-Button betätigt wird, ohne dass Daten in das Formular eingegeben wurden, sollte folgende Fehlermeldung angezeigt werden.

Abbildung 3-7: Fehlermeldungen

Bis zu diesem Zeitpunkt hat Struts im Hintergrund schon einiges getan. Nach dem Absenden der Formulardaten werden diese im *ActionForm*-Bean zwischengespeichert und validiert. In diesem Fall merkt Struts, dass nichts eingegeben wurde und schreibt Fehlermeldungen in eine Java-Collection, welche das JSP anzeigt, nachdem es erneut vom Controller aufgerufen wird.

Wird das Formular korrekt ausgefüllt und der Absenden-Button betätigt, ruft der Controller die JSP ebenfalls erneut auf. Da keine Validierungsfehler auftraten, wird nach dem Sichern der Formulardaten im *ActionForm*-Bean zuvor aber die *Action*-Bean aufgerufen, welche Daten verändert und den Controller aufruft, der die JSP anzeigt. Da die Java-Collection für Fehlermeldungen leer ist, werden nun keine Fehlermeldungen angezeigt. Allerdings findet Struts nun Daten in der *ActionForm*-Bean und trägt diese automatisch wieder in das Formular ein. Ausserdem wird der Benutzer mit einer Meldung begrüsst. Die Umwandlung des Namens in Grossbuchstaben geschieht in der *Action*-Bean.

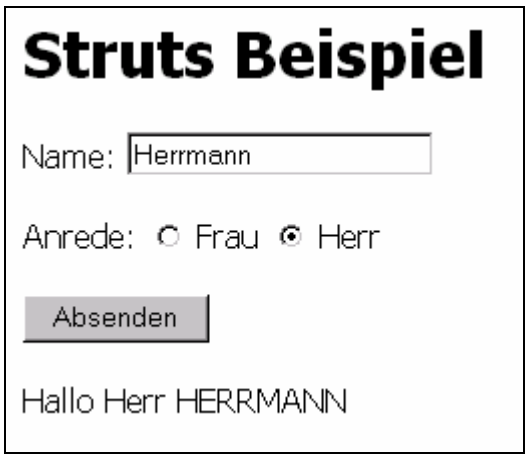

Abbildung 3-8: Die erste Struts MVC Anwendung funktioniert

Interessant ist, dass Struts auch eingegebene Zeichen wie < oder " korrekt behandeln würde. Diese Zeichen können in Web-Anwendungen Probleme verursachen, wenn sie nicht korrekt behandelt werden. Es gibt selbstverständlich noch viel mehr Möglichkeiten bei der Entwicklung mit Struts. Dieses Beispiel ist eine sehr einfache Struts-Anwendung.

3.9 Vergleichbare Frameworks

Neben Struts gibt es noch eine Reihe von weiteren MVC-Frameworks, die ähnliche Ziele verfolgen. Oft unterscheiden sie sich durch einen erweiterten Funktionsumfang von Struts. Dadurch sollen dem Entwickler mehr Werkzeuge in die Hand gegeben und die Entwicklungszeit verkürzt werden. Dies verlängert auf der anderen Seite aber auch die nötige Einarbeitungszeit. Dieses Kapitel gibt einen Überblick über einige Frameworks. Wenn eine Entscheidung für oder gegen Struts getroffen werden soll, ist es wichtig die Alternativen zu kennen.

	Barracuda	Expresso	Struts	Turbine	WebWork
Ausgabe Views	H	J	J	F,J,V,W,X	J,V,X
Trennung Design von Entwicklung	✓	✓	✓	✓	✓
Internationalisierung	✓	✓	✓	✓	✓
Sicherheit		✓		✓	
Formular Validierung	✓	✓	✓	✓	✓
Dokumentation	B,F,G,J,T	B,F,G	B,F,G,J,T	B,G,J	F,G
Error Handling	✓	✓	✓	✓	✓
Einführung	1997	1999	2000	1999	2001
Lizenz	EPL	Apache-ähnlich	Apache	Apache	LGPL
EJB Integration möglich	✓	✓	✓	✓	✓
O/R Mapping		✓		✓	
Connection Pool		✓	✓	✓	
XML-RPC				✓	
WAP/WML	✓			✓	✓
Logging		✓	✓	✓	✓
Cache Manager		✓		✓	
Content Management		✓			
Job Scheduling		✓		✓	

Legende:	
Ausgabe Views:	**Dokumentation:**
F = FreeMarker	D = Online Demo
H = HTML	B = Beispielcode
J = JSP	F = Frequently Asked Questions
T = Tea	G = User's Guide
V = Velocity	J = Online JavaDocs
W = WebMacro	T = Tutorial
X = XML/XSLT	

Abbildung 3-9: Vergleich bekannter MVC-Frameworks[1]

[1] vgl. [@Wafe] und [@Expr2]

Alle hier vorgestellten Frameworks sind wie Struts Open-Source[1] Software und somit kostenlos erhältlich. Beim Betrachten der Tabelle erkennt man, dass sich die Frameworks oft lediglich durch die angebotenen Zusatzfunktionen unterscheiden. Diese sind in der zweiten Hälfte der Tabelle dargestellt. Dass nur zwei der Frameworks im Bereich Sicherheit Funktionalität implementieren, bedeutet nicht, dass die anderen Frameworks unsicher sind. Bei allen Frameworks ist es möglich Sicherheitsmechanismen des Servlet Containers bzw. des Applikations-Servers einzusetzen. Der Einsatz der sogenannten "container-managed security" ist bei Web-Applikationen durchaus üblich. Auch Services wie O/R-Mapping, Connection Pooling, Logging oder Cache Management werden von vielen Applikations-Servern angeboten.

Unter den hier vorgestellten Frameworks ist Struts am einfachsten erlernbar. Struts bietet lediglich die Kernfunktionalität eines MVC-Frameworks und keine weiteren Funktionen. Somit ist Struts schneller erlernbar und besser erweiterbar bzw. anpassbar. Es scheint als wäre dies bei vielen Entwicklern ein Argument, schliesslich ist Struts viel weiter verbreitet als beispielsweise Turbine und Expresso, die wesentlich mehr Funktionalität bieten.

3.9.1 Turbine

Turbine ist wie Struts eine Umsetzung des MVC-Paradigma und wird ebenfalls im Jakarta Projekt der Apache Organisation entwickelt, ist jedoch deutlich komplexer als Struts. Das Framework 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 zur Modellerzeugung und ein Service zur Template-basierten Erzeugung von Views integriert. Ergänzend stehen ein ausgeklügeltes Benutzerkonzept, 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 sogenanntes Turbine Development Kit (TDK) zum Download bereit.

Die *Actions* stellen eines von insgesamt fünf Modulen dar, aus denen sich Turbine zusammensetzt. Sie enthalten die eigentliche Business-Logik, z.B. Verarbeitung von Formulardaten oder auch Aufruf von EJBs. *Actions* verarbeiten jedoch keine HTTP-Requests, sondern werden vom *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 generieren. Während

[1] Der Einsatz von Open-Source Software ist in der ZKB unter Einhaltung von Rahmenbedingungen erlaubt. Siehe dazu 5.3.

im *Screen*-Modul die anzuzeigenden Informationen aufbereitet werden, stellt das *Navigation*-Modul austauschbare Navigationselemente bereit. In beiden Modulen besteht grundsätzlich die Möglichkeit externe Komponenten wie z.B. EJBs anzusprechen. Im *Layout*-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 Turbine Entwickler den Einsatz von Template-Engines, speziell aber von Velocity. Templates sind vollständige HTML-Seiten, die mit einfachen Makros und Variablen angereichert werden, um zur Laufzeit dynamische Inhalte zu generieren. Zu jedem Template wird eine Java-Klasse benötigt, um die View mit konkreten Daten zu versorgen.

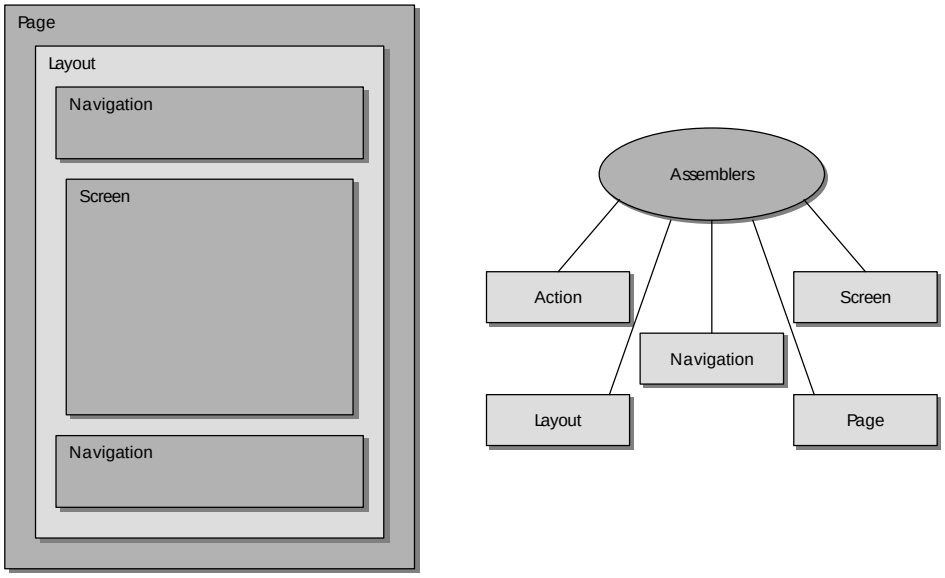

Abbildung 3-10: Die Komponenten des Turbine Frameworks

Auch bei Turbine übernimmt ein zentrales Servlet die Ablaufsteuerung. Abhängig vom Namen des angeforderten Templates wird eine äquivalente Java-Klasse gesucht. Wird eine solche gefunden, kommt diese zum Einsatz, anderenfalls wird auf Default-Klassen zurückgegriffen. In den Java-Klassen wird eine Kontext-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 sie selbst die Controller-Funktionalität. Aus den *Actions* heraus wird die Business-Logik ausgeführt und zudem die Session mit Variablen gefüllt.[1]

3.9.2 Barracuda

Das Barracuda-Framework aus dem Hause Enhydra basiert ebenfalls auf einer MVC-Architektur. Es distanziert sich jedoch entschieden von den JSP-basierten Pull-Ansätzen und setzt mit dem hauseigenen XMLC eine Push-Strategie ein. Enhydra sieht in der Mächtigkeit von JSP und Tag-Libraries einige Schwachpunkte: So muss der Web-Designer meist doch über geringe Java-Kenntnisse verfügen, zumindest aber die neuen Tags der Tag-Library erlernen. Zwar minimiert die Verwendung der Tag-Libraries den Java-Code in den JSP-Seiten, man wird jedoch nicht dazu gezwungen keinen Java-Code einzusetzen. So findet man in vielen Web-Applikationen Formatierungen und Konvertierungen von Zahlen und Strings zwischen den HTML-Tags wieder.

Auch Template-Engines vollziehen diese Trennung nicht vollständig. Die Tag-Libraries verschwinden zu Gunsten einer spezifischen Makrosprache. Für den View-Designer ändert dies relativ wenig. Auf elegante Weise löst XMLC das 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 *setElementSampleTag()* generieren. 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.

Wenn Barracuda einen HTTP-Request empfängt, wird zunächst der Typ des Clients festgestellt. So können unterschiedliche Clients entsprechend ihren Darstellungsfähigkeiten wie z.B. Scripting und Formatierung mit Informationen versorgt werden. Im nächsten Schritt wird aus dem Request ein *Event* generiert und entsprechend weiterverarbeitet. Ein weiteres Package bietet ein automatisches Mapping und die Validierung von HTML-Formularen, wodurch die Arbeit erheblich vereinfacht wird. Die Elemente eines Formulars werden auf Java-Objekte abgebildet – bei Validierungsfehlern werden entsprechende Exceptions geworfen. Nun kann die eigentliche XMLC-basierte Generierung eines HTTP-Response beginnen.

[1] vgl. [@Turb], [@Barr2] und [@Expr2]

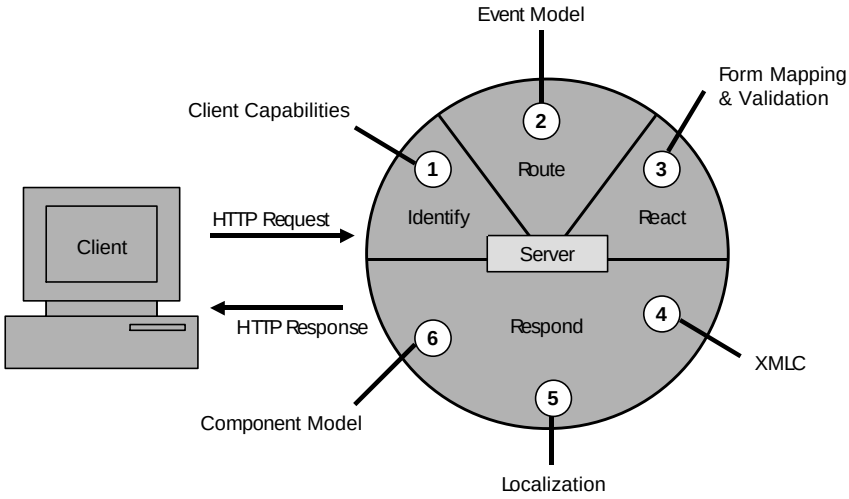

Abbildung 3-11: Die Barracuda Architektur

Die Barracuda-Architektur ist Komponenten-basiert und durch ein serverseitiges Event-Modell verbunden. 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.[1]

3.9.3 Expresso

Jcorporate Expresso ist ein ähnlich komplett ausgestattetes Framework wie Turbine. Es besteht aus drei Kernkomponenten – den Schemata, den Controllern und den *DatabaseObjects*. Abgerundet wird die Architektur durch ein Security-Konzept und eine Reihe zusätzlicher Funktionen wie z.B. Logging, Caching, Connection-Pooling, etc. Das Schema-Objekt ist als eine Art Anwendungskontext zu verstehen – es registriert in einer von *com.jcorporate.expresso.core.dbobj.Schema* abgeleiteten Klasse die anwendungsspezifischen Einstellungen insbesondere aber 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 bereitstellen. Ein *DatabaseObject* bezieht sich auf genau eine Tabelle und verfügt über *add(), delete()* und *update()* Methoden, sowie eine Reihe von Finder-Methoden.

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 Controllers, sondern architekturbedingt auch die des Models. Eine strikte Trennung der Rollen wird also in Expresso nicht verwirklicht. Zudem können Controller

[1] vgl. [@Barr], [@Barr2] und [@Expr2]

mehrere User Interfaces bedienen. Neben JSPs und Template-Lösungen kann der Output auch als XML bzw. via XSLT transformiert zurückgegeben werden. Ein integriertes Sicherheitskonzept ermöglicht die automatische Prüfung von Zugriffsrestriktionen. Dazu stehen die speziellen Klassen *SecuredDBObject* und *DBController* zur Verfügung.[1]

Mit der Version 4.0 wurde 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-Funktionalität, um damit bessere Synergieeffekte zu erzielen.

3.9.4 WebWork

OpenSymphony WebWork ist ein Community-Projekt, welches in mancher Hinsicht dem Framework Struts ähnelt, sich aber dennoch entschieden davon distanziert. WebWork will sich durch seine Einfachheit speziell im Feld der Präsentations-Frameworks etablieren. 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 mit 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 bietet WebWork verschiedene Möglichkeiten der View-Gestaltung. Neben JSPs können auch Template-Engines 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.[2]

[1] vgl. [@Expr], [@Barr2]
[2] vgl. [@Webw]

3.9.5 Sun's Java Server Faces

Die Java Server Faces Technologie ist kein MVC-Framework und aus diesem Grund auch nicht direkt mit Struts vergleichbar. Ich will die Technologie hier trotzdem kurz vorstellen, weil sie die Weiterentwicklung von Struts beeinflussen wird. Java Server Faces beschreibt Sun als Framework von GUI-Komponenten für die Erzeugung von Web-Frontends. Unter anderem soll es in der Lage sein, Elemente einer HTML-Seite mit bestimmten Java-Klassen auf dem Server zu assoziieren. Treten bestimmte Benutzerereignisse auf, beispielsweise das Anklicken einer Checkbox oder das Herausbewegen des Cursors aus einem Textfeld, sollen die zugeordneten Methoden auf dem JSP-Server aktiviert werden. Auf diese Weise könnten beispielsweise Benutzereingaben validiert werden. Laut Hersteller können Tools, die diese Spezifikation umsetzen, Entwickler von den Eigenheiten bestimmter Browser abschirmen und die Verbindung zwischen Client und Server mittels visueller Programmierung erlauben. Derzeit durchläuft Java Server Faces noch den Java Community Process und ist als Java Specification Request (JSR) 127 registriert.[1]

Java Server Faces werden in zukünftigen Struts Versionen Teile des Frameworks ergänzen, bzw. ersetzen. Der Chefentwickler des Struts Frameworks Craig R. McClanahan ist zugleich Mitarbeiter bei Sun Microsystems und Leiter der Expert Group, welche die Spezifikation der Java Server Faces Technologie entwickelt. Laut dem Entwicklerteam ist JSF keine direkte Konkurrenz zu Struts. Die Struts-Entwickler werden JSF sowie auch die relativ neuen JSP Standard Tag Library (JSTL) von Sun in Struts integrieren und Teile der Struts Tag-Libraries mit ähnlicher Funktionalität entfernen. Die Integration dieser neuen Standards wird von den Struts-Benutzern bis jetzt sehr positiv aufgenommen. Laut Craig R. McClanahan bietet sie den Struts-Entwicklern die Möglichkeit, Struts in anderen Richtungen zu erweitern und zu verbessern.

[1] vgl. [@JSF]

4 Entwicklung mit Struts

Neben der kleinen Beispiel-Applikation im dritten Kapitel wurde mit Struts auch eine grössere Applikation entwickelt. Der Name der Applikation ist ADB-Info. Meine Erfahrungen bei der Entwicklung mit Struts dokumentiert dieses Kapitel. Ziel war es mit der Struts-Entwicklung vertraut zu werden, um besser beurteilen zu können in welchen Fällen der Einsatz von Struts Sinn macht. Ausserdem dient der Quellcode der Applikation als zusätzliches Beispiel und soll den Einstieg in die Entwicklung mit dem Framework erleichtern. Der Quellcode der Applikation liegt als WebSphere-Projekt sowie als WAR Archiv im Anhang A.3. auf der CD. Dieses Kapitel dokumentiert die Applikation nicht im Detail. Es werden lediglich einige wesentliche Punkte herausgestellt, die für die Entwicklung mit Struts in der ZKB relevant sind. Die Vorgängerversion von ADB-Info war eine JSP-Applikation (Model 1). Der Quellcode dieser Applikation befindet sich im Anhang A.2.

4.1 Beispielapplikation "ADB-Info"

ADB-Info liefert Informationen über die aktuell laufenden Prozesse der ADB. Dabei sind Zustandsinformationen, wie auch entsprechende Kennzahlen, abrufbar. Ausgehend von einer idealtypischen Data Warehouse Architektur unterscheidet auch die ADB die Ladeseite von der Extraktionsseite.

Das Laden der ADB umfasst die Beschaffung der Daten aus den operativen Systemen. ADB-Info stellt Status und Kennzahlen, die beim Laden der Daten von Interesse sein können, auf Abruf zur Verfügung. Das Extrahieren der Daten umfasst das Zurverfügungstellen der im Ladeprozess gewonnenen und strukturiert abgelegten Daten. ADB-Info stellt auch hier Informationen über den Verlauf der Extraktionsprozesse und die anfallenden Kennzahlen zur Verfügung.

Genauere Angaben zu den während des Ladeprozesses verarbeiteten Datenmengen geben die Kennzahlen. Es steht einerseits ein Bericht bezüglich der Anzahl geladener und zurückgewiesener Datensätze zur Verfügung und andererseits ein Bericht, der nur die Anzahl der zurückgewiesenen Datensätze ausweist. Die Kennzahlen der Extraktion geben Auskunft über die während des Extraktionsprozesses neuen, geänderten und gelöschten Datensätze.

Es ist zu beachten, dass eine Verarbeitung der Daten nur an Werktagen stattfindet. Da die Verarbeitung der Daten nachts geschieht, wird beim Start von ADB-Info vom ermittelten aktuellen Datum automatisch ein Tag abgezogen, bzw. an einem Wochenendtag oder Montag zum letzten Freitag gewechselt.

ADB-Info lief bereits seit ca. einem Jahr als ASP-Applikation produktiv auf einem Microsoft IIS-Server. Diese Applikation wurde von mir im Rahmen dieser Diplomarbeit zuerst in eine

reine JSP-Applikation (Model 1 Variante) und dann in eine Struts-Applikation (Model 2 Variante) portiert.

Es liegt bereits ein Pflichtenheft zu ADB-Info vor, in dem die Applikation spezifiziert wird[1]. Da ADB-Info lediglich 1:1 portiert wurde, ohne Veränderungen an der Funktionalität und GUI vorzunehmen, haben die meisten Informationen in diesem Pflichtenheft nach wie vor Gültigkeit. Dieses Kapitel ist keine offizielle Dokumentation von ADB-Info und kein Benutzerhandbuch. Die folgenden Kapitel sollen lediglich meine Erfahrungen bei der Entwicklung unter Einsatz von Struts darlegen.

4.1.1 Infrastruktur und Architektur

ADB-Info lässt sich, wie weiter unten dargestellt, in das J2EE-Schichtenmodel einordnen. Dies gilt für beide Varianten gleichermassen. Die beiden Varianten unterscheiden sich durch den unterschiedlichen Einsatz von Komponenten, laufen aber auf der gleichen Infrastruktur.

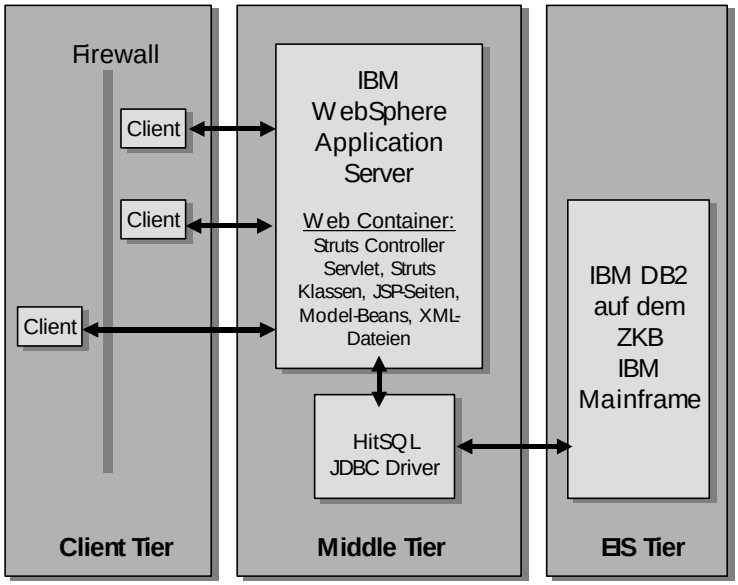

Abbildung 4-1: ADB-Info im J2EE-Schichtenmodel

Es war eine Anforderung der SAR der ZKB, dass ADB-Info auf dem IBM WebSphere Server der ZKB laufen muss. Aktuell wird die Version 4.0 eingesetzt. Daher machte es Sinn bei der Entwicklung mit dem WebSphere Entwicklungswerkzeug WSAD (WebSphere Studio Application Developer) zu arbeiten. Es wurde die Version 4.0.3 eingesetzt. Die von der Applikation angesprochene Datenbank liegt auf dem IBM Mainframe der ZKB. Es wird IBM DB2 for z/OS in der Version 7.1 eingesetzt. Der Einsatz von Struts beeinflusst diese Wahl nicht.

[1] vgl. [ADBI1], [ADBI2] und [ADBI3]

4.1.2 Datenzugriff

Der Zugriff auf die Daten erfolgt mit JDBC. Allerdings wird in den Datenbank-Klassen keine eigene JDBC Connection implementiert, sondern es werden die HitSQL[1] JDBC-Treiber eingesetzt und im WebSphere-Server eine Connection-Pooling-fähige Datenquelle definiert. Diese Datenquelle bietet bessere Performance und schont zudem die Ressourcen der Datenbank, weil nicht für jeden Request eine neue Verbindunge hergestellt werden muss. Die Datenquelle wird von der Applikation aus mittels JNDI aufgerufen. Dadurch müssen die Datenbank-Klassen keine Rechneradressen, Ports, Logins und Ähnliches enthalten. Es wird lediglich ein logischer JNDI-Name in der Form *java:comp/env/jdbc/SampleDB* benutzt, um die Datenquelle zu erhalten und danach mit ihr zu arbeiten. Somit könnten jederzeit ein anderer Treiber, eine andere Datenbank oder andere Benutzerdaten eingesetzt werden, ohne dass irgendwelche Klassen der Applikation verändert werden müssten.

Auch Struts bietet einen einfachen JDBC Connection Pool und die Möglichkeit Datenquellen für eine Anwendung in der Konfigurationsdatei zu definieren. Da die WebSphere Mechanismen mächtiger und komfortabler sind, habe ich mich gegen die Datenbankanbindung mit Struts entschieden.

4.1.3 Implementierung der Model 1 Variante

Die JSP-Version von ADB-Info besteht insgesamt aus 11 JSP-Dateien, die sämtliche Funktionalität beinhalten. Folgende Abbildung zeigt die Struktur der Applikation (Die Ladeseite zeigt Informationen über Liefersysteme, die Extraktionsseite Informationen über Datamarts an):

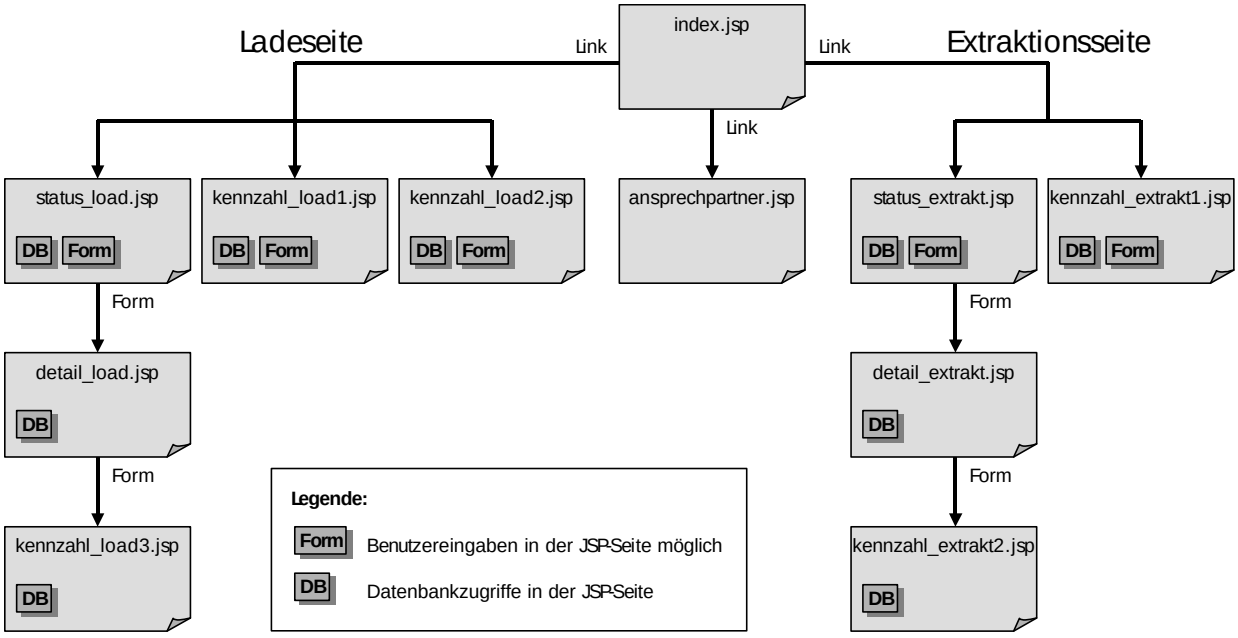

Abbildung 4-2: Struktur der JSP-Version von ADB-Info

[1] HitSoftware entwickelt Produkte für die performante Anbindung von Datenbanken an die Middleware (in diesem Fall der Applikation Server WebSphere). [@Hits]

Die JSP-Version ist eine direkte Übersetzung der ursprünglichen ASP-Dateien in JSP-Dateien. Die JSPs enthalten HTML-Tags zur Darstellung der View sowie die Steuerungslogik und den Code für die Datenbankzugriffe. Diese Variante entspricht somit der Model 1 Architektur. Die einzelnen JSPs werden dadurch extrem gross und unübersichtlich. Zudem gibt es Code-Teile (z.B. die Berechnung des Datums der letzten Verarbeitung), die in mehreren JSPs eingesetzt werden. Bei Veränderungen an diesen Code-Teilen müssten mehrere JSPs angepasst werden.

Der Einstieg in die Applikation geschieht mit einem Aufruf der Datei index.jsp. Von hier aus führen fünf Links auf JSP-Seiten mit DB-Zugriffen und HTML-Formularen und ein Link zu einer JSP-Seite ohne DB-Zugriffe und Formular (ansprechpartner.jsp). Die fünf Views implementieren allesamt ähnliche Funktionalität und unterscheiden sich nur durch die SQL-Statements und die Struktur der darzustellenden Tabellen. Alle fünf Views generieren ein Formular, das aus einem Text-Eingabefeld (Datum der letzten Verarbeitung in der ADB) und einer Dropdown-Liste (Liefersysteme bzw. Datamarts) besteht. Die Elemente der Dropdown-Liste werden aus der DB gelesen. Das Datum der letzten Verarbeitung wird in der jeweiligen JSP errechnet. Beim Absenden des Formulars rufen sich die Seiten selbst auf. Die übergebenen Daten des Formulars auf der jeweiligen Seite entsprechen den Parametern, die dem SQL-Statement übergeben werden. Mit den Eingaben in das Formular beeinflusst der Benutzer also die anzuzeigenden Daten auf der Seite. Von den Seiten status_load.jsp und status_extrakt.jsp aus gelangt man zudem zu Detailansichten der jeweils dargestellten Daten. Der Aufruf der Detailansichten geschieht über Formulare mit *Hidden*-Feldern, die in die Tabelle eingebunden werden und lediglich aus einem *Submit*-Button bestehen. Von den Detailansichten aus können ausserdem Kennzahlen angezeigt werden.

4.1.4 Implementierung der Model 2 Variante mit Struts

Die Struktur der JSP-Dateien der Struts-Version entspricht ebenfalls der in Abbildung 4-2 dargestellten. Die Struts Version von ADB-Info besteht auch aus 11 JSP-Dateien, allerdings entstanden durch den Einsatz von Struts 37 Java Klassen. In diese ist die gesamte Funktionalität ausgegliedert worden. In den JSPs ist nun nur noch der zur Darstellung relevante Code enthalten. Es gibt fünf *ActionForm*-Klassen, 14 *Action*-Klassen, zehn DB-Klassen und acht sonstige Klassen (Datumsverarbeitung und Definition der Datenstrukturen). Es werden also 19 Struts-Klassen benötigt. Die Klassen haben folgende Funktionen:[1]

ActionForm-Klassen

Klasse	Beschreibung
StatusLoadForm	Halten der Daten, die statusLoad.jsp darstellt.
KennzahlLoad1Form	Halten der Daten, die kennzahlLoad1.jsp darstellt.
KennzahlLoad2Form	Halten der Daten, die kennzahlLoad2.jsp darstellt.

[1] Eine detaillierte Beschreibung der Klassen findet sich im Anhang A auf der CD.

StatusExtractForm	Halten der Daten, die statusExtract.jsp darstellt.
KennzahlExtract1Form	Halten der Daten, die kennzahlExtract1.jsp darstellt.

Action-Klassen

Klasse	Beschreibung
StartStatusLoadAction	Vorbereiten der Daten, die statusLoad.jsp darstellt.
StatusLoadAction	Vorbereiten der Daten, die statusLoad.jsp darstellt.
DetailLoadAction	Vorbereiten der Daten, die detailLoad.jsp darstellt.
KennzahlLoad3Action	Vorbereiten der Daten, die kennzahlLoad3.jsp darstellt.
StartKennzahlLoad1Action	Vorbereiten der Daten, die kennzahlLoad1.jsp darstellt.
KennzahlLoad1Action	Vorbereiten der Daten, die kennzahlLoad1.jsp darstellt.
StartKennzahlLoad2Action	Vorbereiten der Daten, die kennzahlLoad2.jsp darstellt.
KennzahlLoad2Action	Vorbereiten der Daten, die kennzahlLoad2.jsp darstellt.
StartStatusExtractAction	Vorbereiten der Daten, die statusExtract.jsp darstellt.
StatusExtractAction	Vorbereiten der Daten, die statusExtract.jsp darstellt.
DetailExtractAction	Vorbereiten der Daten, die detailExtract.jsp darstellt.
KennzahlExtract2Action	Vorbereiten der Daten, die kennzahlExtract2.jsp darstellt.
StartKennzahlExtract1Action	Vorbereiten der Daten, die kennzahlExtract1.jsp darstellt.
KennzahlExtract1Action	Vorbereiten der Daten, die kennzahlExtract1.jsp darstellt.

Datenbank-Klassen

Klasse	Beschreibung
DBSelectSys	Liest die Daten für die Pull-Down Auswahlliste für Liefersysteme bzw. Datamarts.
DBStatusLoad	Liest die Daten für statusLoad.jsp aus der DB.
DBDetailLoad	Liest die Daten für detailLoad.jsp aus der DB.
DBKennzahlLoad3	Liest die Daten für kennzahlLoad3.jsp aus der DB.
DBKennzahlLoad1	Liest die Daten für kennzahlLoad1.jsp aus der DB.
DBKennzahlLoad2	Liest die Daten für kennzahlLoad2.jsp aus der DB.
DBStatusExtract	Liest die Daten für statusExtract.jsp aus der DB.
DBDetailExtract	Liest die Daten für detailExtract.jsp aus der DB.
DBKennzahlExtract2	Liest die Daten für kennzahlExtract2.jsp aus der DB.
DBKennzahlExtract1	Liest die Daten für kennzahlExtract1.jsp aus der DB.

Sonstige Klassen

Klasse	Beschreibung
ActualDate	Diese Klasse ermittelt den Tag, an dem die letzten Verarbeitungen stattfanden.
StructureStatus	Diese Klasse ist für die Datenhaltung einer Zeile der Tabellen in statusLoad.jsp und statusExtract.jsp zuständig.

StructureDetail	Diese Klasse ist für die Datenhaltung einer Zeile der Tabellen in detailLoad.jsp und detailExtract.jsp zuständig.
StructureKennzahlLoad3	Diese Klasse ist für die Datenhaltung einer Zeile der Tabelle in kennzahlLoad3.jsp zuständig.
StructureKennzahlLoad1	Diese Klasse ist für die Datenhaltung einer Zeile der Tabelle in kennzahlLoad1.jsp zuständig.
StructureKennzahlLoad2	Diese Klasse ist für die Datenhaltung einer Zeile der Tabelle in kennzahlLoad2.jsp zuständig.
StructureKennzahlExtract2	Diese Klasse ist für die Datenhaltung einer Zeile der Tabelle in kennzahlExtract2.jsp zuständig.
StructureKennzahlExtract1	Diese Klasse ist für die Datenhaltung einer Zeile der Tabelle in kennzahlExtract1.jsp zuständig.

Obwohl die grosse Anzahl von Klassen auf den ersten Blick verwirrend scheint, ist die Applikation so besser strukturiert. Eine detaillierte Dokumentation aller Klassen befindet sich als JavaDoc auf der CD im Anhang A.3. Der Java Quellcode ist sehr gut kommentiert, so dass sich leicht nachvollziehen lässt was gemacht wird.

4.2 Grundsätze und Richtlinien der ZKB SAR

Die Systemarchitektur der ZKB hat diverse Richtlinien zum Entwickeln von Web-Applikationen definiert. Vor der Entwicklung der Struts-Applikation musste überprüft werden, ob Struts mit der ZKB SAR vereinbar ist. Die folgende Tabelle zeigt relevante Grundsätze aus [SAR02] auf und beschreibt inwiefern sich der Einsatz von Struts mit den Grundsätzen verträgt. Eine detaillierte Darstellung aller Richtlinien und Grundsätze befindet sich in [SAR02].

Grundsatz	**Einsatz von Struts**
Alle Neuentwicklungen sind konsequent in **Client-/Server Komponenten** aufzuteilen.	☺ Der Einsatz von Struts erfüllt diesen Grundsatz.
MVC-Prinzip: Die gewünschte Isolation der Systemteile wird durch eine Anwendungsstruktur mit Trennung in Model (Verarbeitung, Datenhaltung), View (Benutzeroberfläche) und Controller (Organisation, Ereignisvermittlung) erreicht.	☺ Der Einsatz von Struts erfüllt diesen Grundsatz.
Für die Entwicklung neuer Applikationen wird **Java** (zusammen mit WebSphere) eingesetzt.	☺ Struts ist zu 100% mit Java Technologien entwickelt worden. Struts Applikationen bestehen zu 100% aus Java Komponenten.
Java soll nicht nur Client-seitig, sondern auch auf den Servern eingesetzt werden.	☺ Der Einsatz von Struts erfüllt diesen Grundsatz.

Java Anwendungen sollen nach Möglichkeit aus **Java Beans** Komponenten bestehen.	☺ Der Einsatz von Struts erfüllt diesen Grundsatz.
Der **Application-Server** wird als die integrierte technische Middleware für die Unterstützung von Java Server Pages (JSP), Servlets, Enterprise Java Beans (EJB) und sichere Kommunikation zwischen Client und Server eingesetzt. Der ZKB Standard-Application-Server ist **IBM WebSphere**.	☺ Struts läuft auf WebSphere.
Die **technischen Services** wie Session Management, Load Balancing, Transaction Management und Instance Pooling werden vom Application Server WebSphere zur Verfügung gestellt.	☺ Der Einsatz von Struts beeinflusst diesen Grundsatz nicht.
Einfache Internet-Applikationen mit dem Schwerpunkt Information können ausschliesslich Java Server Pages verwenden. **Definition**: Einfache Internet-Applikationen sind solche, welche vorwiegend einen Informations-Charakter haben.	☹ Kein Einsatz von Struts für einfache Model 1 Applikationen.
Komplexe Internet-Applikationen verwenden eine Kombination aus Java Server Pages und mindestens einem Servlet als Dispatcher-Mechanismus. **Definition**: Komplexe Internet-Applikationen sind solche, welche den Schwerpunkt auf die Interaktivität beziehungsweise die Eingaben legen oder gar vermögenswirksame Transaktionen zulassen.	☺ Einsatz von Struts für die Entwicklung von Model 2 Applikationen.
Einsatz vom Enterprise Java Beans: Das Muster JSP/Servlets **mit EJBs** ist anzuwenden, wenn die Applikation eine umfangreiche, komplexe Business-Logik aufweist, die Business-Logik wiederverwendbar sein soll, mehrere unterschiedliche Clients (Browser (Ultra-Thin-)(HTML), Thin-(Java) , Mobile, ...), bedient werden müssen, hohe Anforderungen an die Skalierbarkeit gestellt werden und ein langer Lebenszyklus der Applikation erwartet wird.	☺ Struts unterstützt den Einsatz von EJBs. Es können aber auch "nur" Java Beans eingesetzt werden.

Das Muster JSP/Servlets **ohne EJBs** ist anzuwenden, wenn die Applikation eine einfache Business-Logik aufweist, nur einen Client, zum Beispiel ein Browser (Ultra-Thin-)-Interface unterstützt, niedrige Anforderungen an die Skalierbarkeit stellt und nur ein vergleichsweise kurzer Lebenszyklus der Applikation erwartet wird.	
Die Internet-Applikationen der ZKB werden nicht mit **Browser-Abhängigkeiten** versehen und orientieren sich an den neutralen Standards der W3C-Komission.	☺ Struts beeinflusst den erzeugten HTML Code nicht.
Für Ultra-Thin-Client-Applikationen wird als **Session-Mechanismus** die Cookie- oder die URL-Rewriting-Methode angewendet. Die URL-Rewriting-Methode ist vorzuziehen.	☺ Keine Probleme beim Einsatz von Struts. Struts unterstützt den Entwickler bei der Implementierung der URL-Rewriting-Methode.

Der Einsatz von Struts lässt sich gut mit den Richtlinien der ZKB SAR vereinbaren.

4.3 Sessionhandling und Sicherheit

Struts bietet keine eigenen Sicherheits-Mechanismen. Der Entwickler von Struts Applikationen muss die Mechanismen des Applikations-Servers - also die sogenannte "container-managed security" - nutzen. Der J2EE-Container verwaltet die Session und bietet ausserdem Funktionen für die Authentifizierung und Autorisierung an[1].

Da das HTTP-Protokoll ein zustandloses Protokoll ist, ist es schwer Informationen einer Session über die gesamte Zeit einer Interaktion mit dem Benutzer zu speichern. Um einen HTTP-Request seiner zugehörigen Session zuzuordnen, verwendet der Container eine Session-ID, die per Cookie oder in der Request-URL weitergereicht wird. Da viele Benutzer Cookies in der Browserkonfiguration deaktiviert haben, wird in den meisten Web-Applikationen die Session-ID in der Request-URL weitergereicht. Dies nennt man URL-Rewriting. Der Entwickler muss URL-Rewriting normalerweise selbst implementieren. Hier bietet Struts eine praktische Funktion. Bei deaktivierten Cookies stellt Struts automatisch auf die URL-Rewriting Methode um, ohne dass der Entwickler sich darum kümmern muss. Wichtig dabei ist, dass die Applikation nicht durch das Betätigen des Back-Button im Browser in einen undefinierten Zustand kommt. Darum muss sich i.d.R. der Entwickler selbst kümmern. Auch Struts bietet keine Funktionalität, um dieses Problem zu umgehen.

[1] Umfangreiche Informationen über Sicherheit im J2EE Umfeld findet man u.a. in [Tura01]

Gerade im Bankenbereich ist es oft wichtig zu wissen, ob eine in den Geschäftslogik-Beans gestartete Transaktion[1] erfolgreich abgeschlossen werden konnte. Struts bietet keine Mechanismen an, die diese Aufgabe übernehmen. Transaktionssicherheit muss der Entwickler selbst implementieren. Allerdings bieten auch hier die Applikations-Server Mechanismen an, die den Entwickler entlasten. IBM WebSphere beispielsweise bietet zu allen hier angesprochenen Aspekten Lösungen.

Die Beispielapplikation ADB-Info stellt geringe Anforderungen an die Sicherheit und setzt URL-Rewriting ein, um die Session auch bei deaktivierten Cookies zu erhalten. Ein Anmelden vor dem Benutzen der Applikation ist nicht notwendig.

4.4 Portabilität WSAD – Tomcat – WebSphere Server (AIX)

Da Struts zu 100% J2EE-kompatibel ist, läuft es auf allen J2EE-zertifizierten Applikations-Servern. Seit dem Release des WebSphere Servers Version 4.0 unterstützt IBM Struts offiziell. Es gibt keinerlei Probleme beim Entwickeln einer Struts-Applikation mit dem Entwicklungswerkzeug WSAD. IBM widmet Struts zwei Kapitel im offiziellen "WebSphere Version 4 Application Development Handbook".[2] Vor dem Release der Version 4.0 musste man noch umfangreiche Veränderungen am Server und sogar am Framework vornehmen bevor eine Struts-Applikation lief.

Die Portierung einer mit WSAD entwickelten Struts-Applikation auf den Open-Source Server Tomcat und umgekehrt verläuft ohne Probleme. Eine mit WSAD entwickelte Struts-Applikation konnte von mir problemlos auf dem in der ZKB eingesetzten WebSphere Cluster (AIX) sowie auf einem Tomcat Server installiert werden.

[1] Beispielsweise das Schreiben in eine Datenbank oder das Durchführen einer Überweisung auf ein Konto mittels der Geschäftslogik.

[2] vgl. [@IBM2]

4.5 Struts-Tools

Es gibt Werkzeuge, die dem Entwickler die Arbeit mit Struts erleichtern sollen. Ich habe im Rahmen der Diplomarbeit zwei bekannte Werkzeuge getestet. Struts Console dient lediglich dem komfortablen Editieren von Struts Konfigurationsdateien während Scioworks Camino die gesamte Entwicklung von Struts Applikationen vereinfachen soll.

4.5.1 Struts Console

Struts Console ist ein kostenloses Java-Programm mit grafischer Oberfläche und dient dem Verwalten von Struts Applikationen. Mit Struts Console können die Konfigurationsdateien des Frameworks komfortabel editiert werden. Ausserdem können bestehende JSP/HTML Seiten in Stuts Views, welche Struts Tag-Libraries benutzen, konvertiert werden. Struts Console arbeitet mit diversen Entwicklungsumgebungen wie z.B. Borland JBuilder usw. zusammen. Mehr Informationen gibt es auf [@Cons].

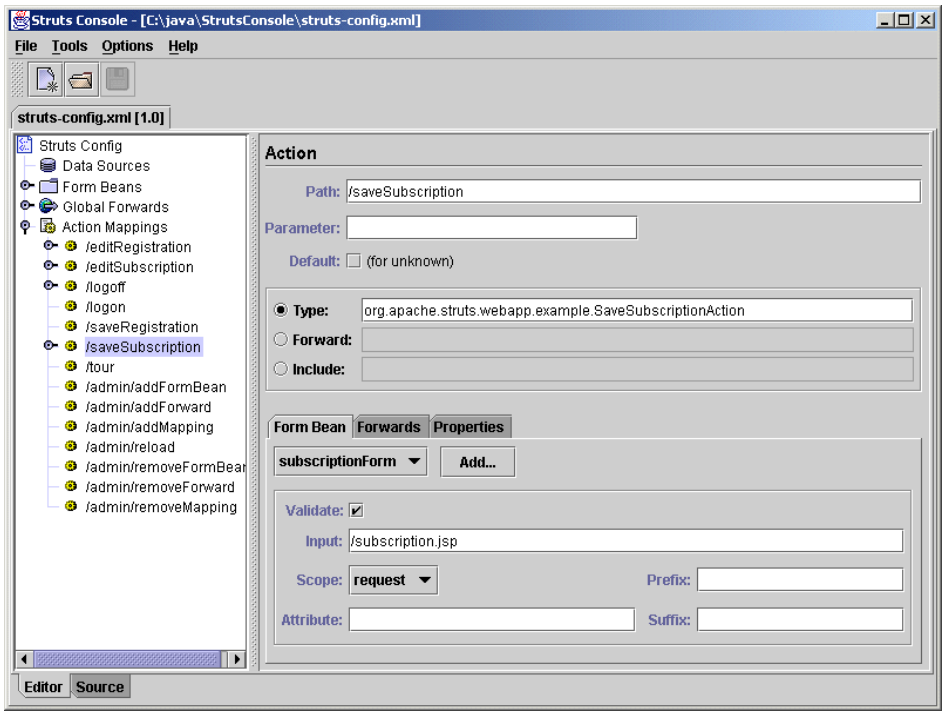

Abbildung 4-3: Screenshot Struts Console – Konfigurieren einer Struts *Action*

4.5.2 Scioworks Camino

Scioworks Camino ist ein wesentlich umfangreicheres Werkzeug als Struts Console. Mit Camino lassen sich ganze Struts Applikationen grafisch mit der Maus zusammenstellen. Die Software erstellt aus dem Design automatisch ein Klassengerüst, in das der Entwickler nur noch den anwendungsspezifischen Quellcode einfügen muss. So kann sich der Entwickler auf das Programmieren der Geschäftslogik konzentrieren. Der von Camino erzeugte Quellcode ist laut Hersteller 100% Struts- bzw. J2EE-kompatibel. Camino kann sogar fertige Struts-Applikationen einlesen, grafisch darstellen und editieren (round-trip engineering). Wizards helfen dem Entwickler beim Erstellen einer Struts-Applikation und konvertieren bestehende

JSP-/HTML-Dateien zu Struts-Views. Die Konfiguration einer Struts-Applikation mit Camino ist sehr komfortabel. Wer grosse und komplexe Struts-Applikationen entwickelt und den Preis von 250$ nicht scheut, sollte sich überlegen Camino einzusetzen. Mehr Informationen gibt es auf [@Scio].

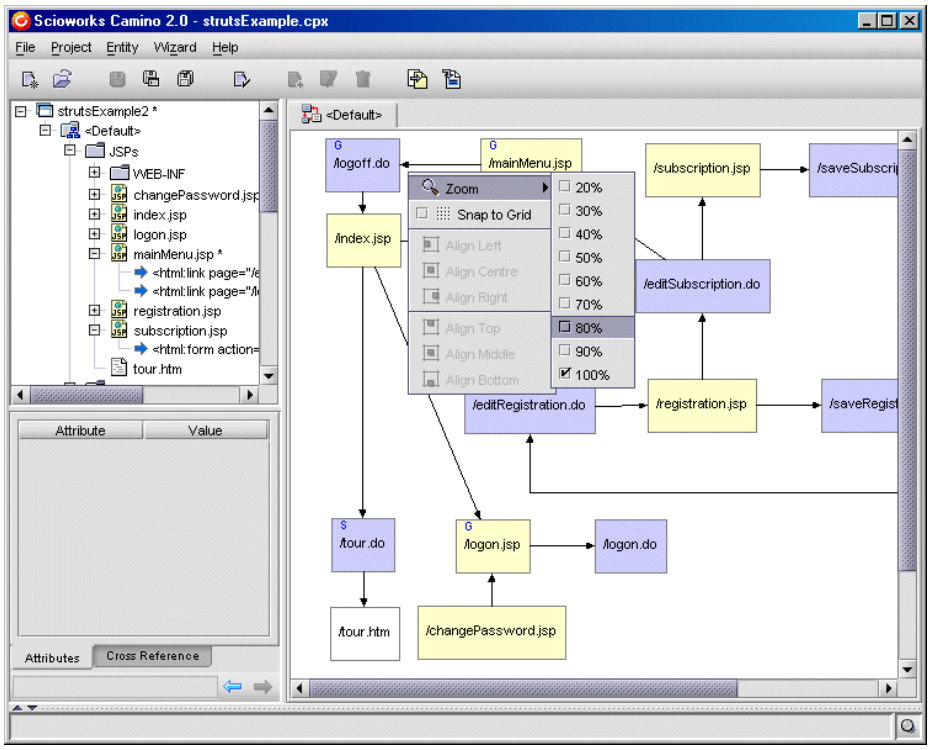

Abbildung 4-4: Screenshot Scioworks Camino - Grafische Entwicklung von Struts Applikationen

5 Erkenntnisse

5.1 Vergleich der Model 1- und Struts-Variante

5.1.1 Funktionsumfang

Der Funktionsumfang der beiden Varianten ist identisch. Aus diesem Grund lassen sich die Unterschiede in der Performance, der Wartbarkeit und im Entwicklungsaufwand gut erkennen.

5.1.2 Performance

Für die Messung der Performance wurde ein kleines Java-Programm aus [Will01] erweitert und angepasst. Das Programm entspricht einem HTTP-Client. Es sendet einen HTTP-Request an einen Server und misst die Zeit vom Aufruf bis zu dem Zeitpunkt, an dem die Antwort vollständig den Client erreicht hat. Der Vorgang entspricht einem Benutzer, der mit einem Browser eine bestimmte Seite betrachtet. Der Client benutzt die `URL`- bzw. `URLConnection`-Klassen für die HTTP-Verbindung. Es kann über Parameter beim Aufruf die URL, die Anzahl der zu erzeugenden parallel ablaufenden Treads (im übertragenen Sinn die Anzahl der parallelen Benutzer) und die Anzahl der Requests pro Thread übergeben werden. Beispielsweise generiert der Client beim Aufruf mit `HttpBench 30 20 http://www.yahoo.de` 30 Threads, die gleichzeitig jeweils 20 Requests auf www.yahoo.com ausführen. Insgesamt werden also 600 Requests ausgeführt. Der Client gibt die Antwortzeit und die Anzahl der Requests pro Sekunde aus. Die benötigten Java-Klassen befinden sich im Anhang A.4. auf der CD. Im Folgenden ist der Quellcode des Clients dargestellt.

```
01 /************************************************
02  *  HTTP-Benchmark                              *
03  *  08/2002 Markus Herrmann                     *
04  *                                              *
05  *  Aufruf: HttpBench <Threads> <Requests> <URL>*
06  *    Threads: Anz. der zu erzeugenden Threads  *
07  *    Requests: Anz. der Requests pro Thread    *
08  *    URL: Request URL                          *
09  ************************************************/
10
11 import java.io.*; // Streams
12 import java.net.*; // URL Request
13 import java.text.*; // Formatierung der Ausgabe von Dezimalzahlen
14
15 public class HttpBench extends Thread
16 {
17    static int THREADS = 1; // Anz. gleichzeitiger Threads
18    static int REQUESTS_PER_THREAD = 10; // Requests pro Thread
19    static URL url; // URL Objekt
20
21    public static void main (String[] arg) throws Exception
22    {
23       url = new URL (arg[2]); // URL auslesen
24       THREADS = Integer.parseInt(arg[0]); // Anz. Threads auslesen
25       REQUESTS_PER_THREAD = Integer.parseInt(arg[1]); // Anz. Requests auslesen
26
27       Thread[] thread = new Thread[THREADS]; // Threads erzeugen
28       for (int i = 0; i < THREADS; i++)
29       {
30          thread[i] = new HttpBench ();
31       }
```

```
32
33        int requests = THREADS * REQUESTS_PER_THREAD; // Gesamt-Anz. Requests
34
35        System.out.println ("---------------------------------------" +
36                            "---------------------------------------");
37        System.out.println ("Benchmark fuer URL " + url);
38
39        System.out.print ("Threads: " + THREADS);
40        System.out.print ("  Requests per Thread: " + REQUESTS_PER_THREAD);
41        System.out.println ("  Requests gesamt: " + requests);
42
43        System.out.println ("** Start Benchmark **");
44
45        long start = System.currentTimeMillis (); // Zeitmessung beginnen
46
47        for (int i = 0; i < THREADS; i++)
48        {
49           thread[i].start (); // Threads starten
50        }
51
52        for (int i = 0; i < THREADS; i++)
53        {
54           thread[i].join (); // Warten auf Ende der Threads bzw. Requests
55        }
56
57        long end = System.currentTimeMillis (); // Zeitmessung beenden
58
59        System.out.println ("** Ende Benchmark **");
60
61        long millis = end-start;
62        double seks = (double) millis / 1000;
63        double mins = seks / 60;
64        double rps = requests * 1000 / (double) millis;
65
66        DecimalFormat df1 = new DecimalFormat("#,###,##0.00"); // 123,456.78
67        DecimalFormat df2 = new DecimalFormat("#,###,##0"); // 123.456
68
69        System.out.print ("Benoetigte Zeit:  Millisekunden: " + df2.format(millis));
70        System.out.print ("  Sekunden: " + df1.format(seks));
71        System.out.println ("  Minuten: " + df1.format(mins));
72        System.out.println ("Requests/Sekunde: " + df1.format(rps));
73     }
74
75     public void run () // Diese Methode wird für jeden Thread einmal ausgeführt
76     {
77        try
78        {
79           // Request auf die uebergebene URL
80           for (int i = 0; i < REQUESTS_PER_THREAD; i++)
81           {
82              // Seite in einen Buffer einlesen
83              InputStream in = url.openStream ();
84              ByteArrayOutputStream out = new ByteArrayOutputStream ();
85              byte[] buffer = new byte[4096];
86              int len;
87              while ((len = in.read (buffer)) != -1)
88              {
89                 out.write (buffer, 0, len);
90              }
91              in.close ();
92           }
93        }
94        catch (Exception e)
95        {
96           e.printStackTrace ();
97        }
98     }
99  }
```

Listing 5-1: Java Client für die Messung der Performance

Die hier verglichenen Varianten laufen beide auf dem gleichen Server und benutzen beide die gleiche Datenbankanbindung. Da der WebSphere Server von mehreren Applikationen gleichzeitig benutzt wird und zudem die Datenbankinstanz nicht exklusiv genutzt werden

kann, können Schwankungen in den Messungen durch Einflüsse von aussen auftreten. Auch die Geschwindigkeit der Netzwerkverbindung des Clients zum WebSphere Server bzw. zwischen WebSphere Server und Datenbank spielt eine Rolle. Um diese Einflüsse möglichst gering zu halten und trotzdem aussagefähige Ergebnisse zu ermitteln, wurden die Messungen an verschiedenen Tageszeiten wiederholt und die Durchschnittswerte ermittelt.

Der Vergleich greift bei beiden Varianten auf die gleiche View mit dem Namen *status_load.jsp* zu. Bei der JSP-Variante (Model 1) entspricht dies einem Aufruf der URL *http://adbinfo1.ex.zkb.ch/ prod/status_load.jsp.* Bei der Struts Variante (Model 2) entspricht dies einem Aufruf einer Struts Aktion mit dem Namen *startStatusLoad*. Dies geschieht durch den Aufruf der URL *http://adbinfo1.ex.zkb.ch/prod1/startStatusLoad.do*.

Es werden pro Variante 9 Messungen vorgenommen. Dabei wird jeweils die gesamte Antwortzeit in Millisekunden, die Antwortzeit pro Request in Millisekunden und die Anzahl der Requests pro Sekunde notiert. Um möglichst reale Bedingungen zu simulieren, werden eine unterschiedliche Anzahl von Requests von einer unterschiedlichen Anzahl von Clients abgeschickt.

1 Client 1 Request, gesamt 1 Request

Variante	**Antwortzeit gesamt (ms)**	**Antwortzeit/Request (ms)**	**Requests/sek**
JSP	3.325	3.325	0,3
Struts	8.913	8.913	0,11

1 Client, 10 Requests, gesamt 10 Requests

Variante	**Antwortzeit gesamt (ms)**	**Antwortzeit/Request (ms)**	**Requests/sek**
JSP	27.299	2.729,9	0,37
Struts	26.989	2.698,9	0,37

1 Client, 20 Requests, gesamt 20 Requests

Variante	**Antwortzeit gesamt (ms)**	**Antwortzeit/Request (ms)**	**Requests/sek**
JSP	59.425	2.971,25	0,34
Struts	48.209	2.410,45	0,41

10 Clients, 1 Request, gesamt 10 Requests

Variante	**Antwortzeit gesamt (ms))**	**Antwortzeit/Request (ms)**	**Requests/sek**
JSP	5.989	598,9	1,67
Struts	4.767	476,7	2,1

10 Clients, 10 Requests, gesamt 100 Requests

Variante	**Antwortzeit gesamt (ms)**	**Antwortzeit/Request (ms)**	**Requests/sek**
JSP	48.079	480,79	2,08

Struts	39.697	396,97	2,52

10 Clients, 20 Requests, gesamt 200 Requests

Variante	Antwortzeit gesamt (ms)	Antwortzeit/Request (ms)	Requests/sek
JSP	99.483	497,42	2,01
Struts	96.368	481,84	2,08

20 Clients, 1 Request, gesamt 20 Requests

Variante	Antwortzeit gesamt (ms)	Antwortzeit/Request (ms)	Requests/sek
JSP	10.956	547,8	1,83
Struts	11.276	563,8	1,77

20 Clients, 10 Requests, gesamt 200 Requests

Variante	Antwortzeit gesamt (ms)	Antwortzeit/Request (ms)	Requests/sek
JSP	89.909	449,54	2,22
Struts	98.562	492,81	2,03

20 Clients, 20 Requests, gesamt 400 Requests

Variante	Antwortzeit gesamt (ms)	Antwortzeit/Request (ms)	Requests/sek
JSP	152.189	380,47	2,63
Struts	231.833	579,58	1,73

Beim Betrachten der Messergebnisse fällt auf, dass Requests vom Server wesentlich schneller abgearbeitet werden, wenn sie parallel erzeugt werden. Beispielsweise haben 10 hintereinander gestartete Requests auf die JSP-Variante insgesamt eine Antwortzeit von 27.299 ms während 10 parallel gestartete Requests auf die JSP-Variante lediglich eine Antwortzeit von 5.989 ms haben. Dies ist durch die Funktionsweise des WebSphere Servers begründet. Da der WebSphere Server einen Cache-Speicher benutzt um die Antwortzeit zu verkürzen, gibt es in allen Messreihen immer einen Ausreisser am Anfang. Die Antwortzeit des ersten Requests ist immer höher als die der nachfolgenden Requests. Dies macht sich vor allem im der ersten Tabelle bemerkbar. Diese zwei Eigenschaften treten bei beiden Varianten auf. Somit lassen sich die Ergebnisse gut vergleichen.

Durchschnittswerte aller 9 Messungen

Variante	Antwortzeit/Request (ms)	Requests/sek
JSP	1.330	1,49
Struts	1.889	1,45

Fazit:

Ein Vergleich der Durchschnittswerte beider Varianten zeigt, dass die JSP-Variante nur minimal schneller ist als die Struts-Variante. Dieser Unterschied könnte aber auch durch

Einflüsse von Aussen zustande kommen. Es scheint als würde der Dispatch-Mechanismus des Struts-Controllers keine zusätzliche Zeit benötigen. Zumindest ist der Unterschied kaum messbar. Ein direkter Aufruf einer JSP-Seite auf dem Server ist nahezu gleich schnell wie der Aufruf einer Struts-Action. Der Einsatz von Struts schlägt sich also nicht messbar auf die Performance nieder.

5.1.3 Entwicklungsaufwand

Beim Vergleich des Entwicklungsaufwands der beiden Varianten gehe ich davon aus, dass ein Entwickler gute Java- und J2EE-Kenntnisse hat. Wie bei jedem Framework ist auch bei Struts zu Beginn der Entwicklung einer Applikation der Zeitaufwand grösser, weil das Framework zuerst erlernt werden muss. Bis der Entwickler sich in Struts eingearbeitet hat vergehen einige Tage, in denen bei der Entwicklung einer Model 1 Applikation mit JSPs schon erste Ergebnisse sichtbar wären. Soll sich der Einsatz eines Frameworks lohnen, muss diese Zeit wieder kompensiert werden können.

Wenn man die beiden im Rahmen der Diplomarbeit entstandenen Varianten der Beispielapplikation ADB-Info vergleicht, wirkt die Struts-Variante auf den ersten Blick extrem komplex. Es scheint nachteilig, dass aus 11 JSPs der Model 1 Variante eine Struts Applikation wird, die aus 37 Java-Klassen und ebenfalls 11 JSPs besteht. Die Aufteilung der Funktionalität in Java-Klassen nach dem MVC-Prinzip bringt allerdings schnell Vorteile mit sich. Alleine die Möglichkeit der getrennten Entwicklung von Benutzeroberflächen und Geschäfts- und Steuerungslogik beim Einsatz von Struts bedeutet bei Projekten mit mehreren Entwicklern schnell eine Zeitersparnis durch kürzere Entwicklungszeit. HTML-Programmierer können parallel zu den Java-Programmierern arbeiten und müssen keine Java-Kenntnisse erwerben, da bei der Entwicklung der Views nur noch mit Tags gearbeitet werden kann. Allerdings entstehen durch den Einsatz eines Frameworks wegen der grösseren Komplexität auch neue Fehlerquellen. Bei kleineren Applikationen lohnt sich der Mehraufwand für das Einarbeiten in Struts nicht. Hier ist eine JSP-Variante viel schneller entwickelt und auch relativ übersichtlich. Der Einsatz von Struts macht erst ab einer gewissen Grösse der Applikation Sinn.

Ist ein Entwickler erst mal mit Struts vertraut, benötigt die Entwicklung einer Struts-Applikation kaum mehr Zeit als die Entwicklung einer Model 1 Applikation. Spätestens nachdem im ersten Projekt Erfahrungen gesammelt wurden, profitiert jedes weitere Projekt vom Einsatz von Struts.

Meiner Erfahrung nach lohnt sich der Einsatz von Struts für erfahrene J2EE-Programmierer bereits ab einer Grösse von 10 Views mit Benutzerinteraktion. Dies entspricht der Definition von grossen bzw. komplexen Web-Applikationen in der ZKB SAR[1].

[1] Siehe Definition der ZKB SAR für komplexe Web-Applikationen in Kapitel 4.2

5.1.4 Wartbarkeit

Was die Wartbarkeit anbelangt, ist eine Struts Applikation einer reinen JSP-Lösung weit überlegen. So muss beispielsweise für eine Änderung der Ausgabetexte von Views der Struts-Variante lediglich die Ressourcen-Datei verändert werden. Durch Hinzufügen einer neuen Ressourcen-Datei in einer anderen Sprache wäre die Applikation innerhalb von Minuten in einer anderen Sprache benutzbar. Sämtlicher Java- und JSP-Code bleibt in beiden Fällen unverändert. Ebenso könnte sich durch das Ausgliedern von Datenbank-Code in der Struts-Applikation die Datenbank ändern, ohne dass Veränderungen an allen Views nötig wären.

Eine Struts-Applikation ist besser strukturiert und somit die Architektur auch besser nachvollziehbar als bei einer JSP-Applikation, bei der sämtliche Funktionalität in einer Datei steht. Struts-Programmierer pflegen und erweitern eine Struts-Applikation wesentlich schneller, als es ein JSP-Programmierer in einer Model 1 Applikation schafft. Diese Erfahrung konnte ich im Rahmen dieser Diplomarbeit selbst machen, nachdem ich an beiden Varianten nachträglich Veränderungen vornehmen musste.

5.2 Die Zukunft von Struts

Über die Zukunft eines Open-Source[1]-Frameworks lassen sich nur schwer Aussagen machen. Die Tatsache, dass Struts in der Entwicklergemeinde sehr populär ist und von namhaften Herstellern bekannter Applikations-Server unterstützt wird, spricht für den Fortbestand von Struts. Im Apache bzw. Jakarta Umfeld werden schon seit langem zuverlässige Technologien entwickelt. Das bekannteste und älteste Beispiel ist der Apache Webserver, der im Internet sehr stark verbreitet ist. Ich gehe davon aus, dass auch Struts noch für lange Zeit weiterentwickelt wird und somit eine sichere Zukunft hat. Dass der Chefentwickler von Struts Sun Mitarbeiter ist und an diversen Spezifikationen im J2EE-Umfeld mitarbeitet, beeinflusst Struts sicher positiv. So wird Struts Sun’s neue JSF-Technologie schon mit dem nächsten Release unterstützen, obwohl JSF noch in der Beta-Phase ist. Da die Entwickler von Apache Projekten ausnahmslos auf freiwilliger Basis an den Projekten mitwirken, kann man den Fortbestand allerdings nie mit 100%-iger Sicherheit garantieren. Aber das ist selbst bei kommerziellen Projekten grosser Softwarefirmen nicht immer der Fall.

5.3 Struts in der ZKB

Es wurde für die HTML-Onlinebank der ZKB bereits ein Framework mit ähnlicher Funktionalität wie Struts entwickelt. Dieses Framework setzt ebenfalls das MVC-Paradigma um. Es lässt sich über XML-Dateien konfigurieren, um eine "Lose Kopplung" der Komponenten zu erreichen. Die Entwicklung dieses Frameworks begann schon bevor Struts erhältlich war. Wäre Struts zu diesem Zeitpunkt erhältlich gewesen, wäre laut Aussage der Entwickler Struts eingesetzt worden. Zum jetzigen Zeitpunkt lässt sich das in der HTML-Onlinebank eingesetzte Framework nicht einfach durch Struts ersetzen, weil es sehr stark mit

[1] Der Einsatz von Open-Source Software ist in der ZKB unter Einhaltung von Rahmenbedingungen erlaubt. Siehe dazu 5.3.

dem restlichen Code verbunden ist. Ein Einsatz von Struts in der HTML-Onlinebank ist also erst nach erheblichem Aufwand möglich – jedoch nicht unmöglich. Inwiefern sich der Aufwand lohnt, kann ich nicht ausreichend beurteilen.

Neben der grössten Web-Applikation der ZKB, der HTML-Onlinebank, gibt es noch eine ganze Reihe weiterer grosser Web-Applikationen, bei denen der Einsatz von Struts sinnvoll ist. Auch hier ist es schwer den Aufwand für die Portierung zu beurteilen. Bei einer Neuentwicklung grösserer Web-Applikationen[1] ist der Einsatz von Struts ohne Einschränkung zu empfehlen.

Für den Einsatz des Open-Source-Frameworks Struts in der ZKB müssen Rahmenbedingungen geschaffen werden. So muss ein System-Owner (Verantwortlich für wirtschaftlichen Einsatz) bzw. ein System-Engineer (Verantwortlich für die technischen Aspekte) für Struts gefunden werden. Dies setzt voraus, dass das Informatik-Management der ZKB den Einsatz der Technologie bewilligt. Diese Bewilligung steht noch aus. Da im Rahmen dieser Diplomarbeit nur positive Erfahrungen mit Struts gemacht wurden und Struts problemlos mit der ZKB SAR vereinbar ist, kann davon ausgegangen werden, dass diese Voraussetzungen bald geschaffen werden. Somit könnte der Einsatz von Struts für die Entwicklung grösserer Web-Applikationen[1] bald zum Standard in der ZKB werden.

[1] Siehe Definition der ZKB SAR für komplexe Web-Applikationen in Kapitel 4.2

6 Fazit

Struts ist ein MVC-Framework, das dem Entwickler viel Arbeit bei der Entwicklung einer Model 2 Applikation abnimmt, durch klare Strukturen trotzdem nicht zu komplex und somit schnell erlernbar ist. Die J2EE Spezifikation sowie auch das vergleichbare .Net-Framework von Microsoft beinhalten beide kein MVC-Framework. Entwickler sind somit auf Produkte von Drittanbietern angewiesen.

Der Einsatz von Frameworks ist immer mit einem anfänglichen Lernaufwand verbunden. Er lohnt sich aber in den meisten Fällen, da sich die Entwicklungszeit meist schnell verkürzt. Es sollte das Ziel sein, die Applikation von Anfang an auf einer soliden und gut wartbaren Architektur zu entwickeln. Dadurch wird die Wartbarkeit der Software verbessert und eine effiziente Rollentrennung zwischen Web-Designer und Java-Programmierer ermöglicht.

Ob das hier vorgestellte Framework Struts nun tatsächlich am geeignetsten ist, kommt sicher auf den Einzelfall an. Oftmals ist sicher der Grad der notwendigen Flexibilität entscheidend. Hier bietet Struts aufgrund seiner Einfachheit relativ viel Raum für individuelle Anpassungen. Auf der anderen Seite wurden beispielsweise Turbine und Expresso mit zahlreichen Zusatz-Funktionen ausgestattet, was in manchen Fällen von Vorteil sein kann. Zu bedenken ist, dass der Einarbeitungsaufwand um so beträchtlicher wird, je umfangreicher das Framework ist. Es ist abzuwägen, ob man die mitgelieferten Funktionen eines Frameworks verwenden möchte, oder lieber auf die Persitenz-, Logging- und Security-Mechanismen des Applikations-Servers zurückgreift.

Weiterhin spielen auch die bisherigen Erfahrungen und Vorkenntnisse der Projektmitarbeiter eine Rolle. Der angesprochene Lernaufwand minimiert sich bei dem Einsatz eines bekannten Frameworks. Die Entscheidung für ein Framework hängt ausserdem von der Art der Gestaltung der Views ab. Hat man bei Expresso noch die Wahl zwischen JSP, Templates und XSLT, so kann Struts 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 an den Projektzielen. Den Web-Designer entlastet ein MVC-Framework in allen Fällen, da seine Unabhängigkeit erklärtes Ziel eines jeden Frameworks ist.

Wie bereits erwähnt war Struts im Jahr 2001 eines der populärsten Frameworks. Das zeigt schon die offizielle Unterstützung in den Applikations-Servern von IBM und Bea. Ausschlaggebend dafür ist die Tatsache, dass es als reines MVC-Framework konzipiert wurde. Dadurch verfügt es nicht über den Overhead an Zusatz-Services und ist zudem schneller erlernbar.

Ich konnte im Rahmen meiner Diplomarbeit durchgehend gute Erfahrungen mit Struts machen und kann das Framework für die Neuentwicklung grosser Web-Applikationen[1] empfehlen. Es war richtig, dass die ZKB Struts im Rahmen einer Diplomarbeit untersucht hat. Ich empfehle Struts in die ZKB SAR aufzunehmen.

[1] Siehe Definition der ZKB SAR für komplexe Web-Applikationen in Kapitel 4.2

Anhang

A Sämtlicher im Rahmen der Diplomarbeit entstandener Quellcode ist auf der beigelegten CD vorhanden. Der Quellcode wurde sowohl als WSAD-Projekt wie auch als WAR-Datei abgelegt. Somit können die Applikationen in jedem beliebigen J2EE-kompatiblen Applikationsserver betrachtet werden. Der Quellcode befindet sich im Verzeichnis SOURCE:

1. Beispiel-Applikation aus Kapitel 3.8
2. ADB-Info JSP Version
3. ADB-Info Struts Version
4. HTTP-Benchmark

B Auf der CD befindet sich im Verzeichnis SOFTWARE folgende Software:

1. jakarta-struts-1.0.2.zip (Jakarta Struts Framework 1.0.2., Open-Source)
2. jakarta-struts-1.0.2-src.zip (Struts Quellcode 1.0.2., Open-Source)
3. jakarta-tomcat-4.0.3.zip (Jakarta Tomcat Server 4.0.3., Open-Source)
4. struts-console-3.1.zip (Struts Console 3.1, Freeware)
5. Camino_2_0.exe (Scioworks Camino 2.0, Trial Version)
6. j2sdk-1_3_1-win.exe, (Sun’s J2SE 1.3.1.)
7. j2sdk-1_3_1-doc.zip, (Sun’s J2SE 1.3.1. Documentation)
8. j2sdkee-1_3_1-win.exe (Sun’s J2EE 1.3.1.)
9. j2sdkee-1_3_1-doc-win.exe (Sun’s J2EE 1.3.1. Documentation)

Quellenverzeichnis

Literaturverzeichnis

[ADBI1] M. Roger, ADB-Info - Anforderungsspezifikation, Version 2.0, Zürcher Kantonalbank, LILP, Januar 2001

[ADBI2] M. Roger, ADB-Info - Detailspezifikation, Version 2.1, Zürcher Kantonalbank, LILP, August 2001

[ADBI3] D. Dagne, ADB-Info - Dokumentation zur Webapplikation, Version 2.0, Zürcher Kantonalbank, LILP, Dezember 2002

[Busc02] A. Buschbacher: Diplomarbeit: Datenzugriffsmechanismen für Java/WebSphere-basierte Anwendungen, Fachhochschule Konstanz/Zürcher Kantonalbank, 2002

[Gamm01] E. Gamma / R. Helm / R. Johnson / J. Vlissides: Entwurfsmuster: Elemente wiederverwertbarer objektorientierter Software, Deutsche Übersetzung von D. Riehle, Addison-Wesley, 2001

[Kamm02] C. Kamm / C. Klein: Komplexe Web-Anwendungen mit "Struts", in: Objekt-Spektrum 3/2002

[SAR02] SAR – Die Systemarchitektur der ZKB, Zürcher Kantonalbank, LIXA, 2002

[Tura01] V. Turau / M. Schmidt / K. Saleck: Java Server Pages und J2EE: Unternehmensweite Web-basierte Anwendungen, dpunkt, 2001

[Will01] S. Wille: GoTo Java Server Pages, Addison-Wesley, 2001

Internetquellen

[@Apac] Apache Software Foundation, http://www.apache.org/

[@Barr] Barracuda Framework, http://barracuda.enhydra.org/

[@Barr2] Barracuda - Surveying the Landscape, http://barracuda.enhydra.org/cvs_source/Barracuda/docs/landscape.html

[@Bea] Bea Application Server, http://www.bea.com/

[@Comp] Web-Application-Server im Labortest, http://www.computerwoche.de/

[@Cons] Struts Console, http://www.jamesholmes.com/struts/console/

[@Enhy] Enhydra Application Server, http://www.enhydra.org/

[@Expr] Expresso Framework, http://www.jcorporate.com/

[@Expr2] MVC Application Framework Matrix Review, http://www.jcorporate.com/html/products/expresso/matrix_compare.html

[@Hits] HiT Software, http://www.hitsw.com/

[@IBM] IBM WebSphere Application Server, http://www.ibm.com/

[@IBM2] IBM Redbooks, http://www.redbooks.ibm.com/

[@J2EE] Sun J2EE Overview, http://java.sun.com/j2ee/overview.html

[@J2SE] Sun Java 2 Standard Edition, http://java.sun.com/j2se/

[@JBos] JBoss Application Server, http://www.jboss.org
[@JSF] JSR 127 – Java Server Faces, http://jcp.org/jsr/detail/127.jsp
[@Macr] Macromedia Coldfusion Application Server, http://www.macromedia.com/
[@Orio] Orion Application Server, http://www.orionserver.com/
[@Resi] Resin Application Server, http://www.caucho.com/
[@Scio] Scioworks Camino, http://www.scioworks.com/
[@Stru] Jakarta Struts Framework, http://jakarta.apache.org/struts/
[@Stru2] Struts and JSF, http://jakarta.apache.org/struts/proposals/struts-faces.html
[@Stru3] The Struts User's Guide, http://jakarta.apache.org/struts/userGuide/
[@Stru4] Struts Controller UML diagrams, http://rollerjm.free.fr/pro/Struts.html
[@Tomc] Apache Tomcat Server, http://jakarta.apache.org/tomcat/
[@Turb] Turbine Framework, http://jakarta.apache.org/turbine/
[@Veloc] Velocity template engine, http://jakarta.apache.org/velocity/
[@Wafe] Waferproject Feature Matrix, http://www.waferproject.org/feature-matrix2.html
[@Webw] WebWork Framework, http://opensymphony.com/webwork/

Stichwortverzeichnis

Ehrenwörtliche Erklärung

Hiermit erkläre ich, Markus Herrmann, geboren am 06.03.1976 in Heidelberg, ehrenwörtlich,

(1) dass ich meine Diplomarbeit mit dem Titel:

"Einsatz des J2EE Frameworks Jakarta Struts"

bei der Zürcher Kantonalbank in Zürich/Schweiz unter Anleitung von Professor Dr. Eduard Klein und Dipl. Ing. Bernd Reichert selbständig und ohne fremde Hilfe angefertigt habe und keine anderen als in der Abhandlung angeführten Hilfen benutzt habe;

(2) dass ich die Übernahme wörtlicher Zitate aus der Literatur sowie die Verwendung der Gedanken anderer Autoren an den entsprechenden Stellen innerhalb der Arbeit gekennzeichnet habe.

Ich bin mir bewusst, dass eine falsche Erklärung rechtliche Folgen haben wird.

Zürich, 31. Januar 2003