Universität Hamburg
Fachbereich Informatik

Studienarbeit

im Arbeitsbereich Softwaretechnik

**Thema:** Entwicklung eines browserbasierten Client Front-End für das Equipment Management System, mit Unterstützung des Frameworks - JaverServer Faces

**eingereicht von:** Randolph Tendzegolskis <r.tendzegolskis@web.de>

**eingereicht am:** 27. März 2007

**Betreuer:** Herr Dr. Guido Gryczan
## Inhaltsverzeichnis

Vorwort ................................................. v
Abkürzungsverzeichnis ................................ vii
Abbildungsverzeichnis ................................ viii

### 1 Einleitung
1.1 Ausgangslage .................................... 1
1.2 Ziel .............................................. 2
1.3 Übersicht ....................................... 3

### 2 Einführung und Grundbegriffe
2.1 J2EE - Java2 Plattform Enterprise Edition .......... 4
2.2 Java Server Pages und Servlets .................... 10
2.3 Architekturen ................................... 12
   2.3.1 Das MVC-Paradigma ......................... 13
   2.3.2 JSP-basierte Model 1-Architektur .......... 14
   2.3.3 JSP-basierte Model 2 Architektur .......... 15
   2.3.4 Die Umsetzung des MVC-Paradigma im Framework - JavaServer Faces 16
2.4 Laufzeitumgebung ................................ 17
   2.4.1 Jakarta Tomcat Server ..................... 17
   2.4.2 Der JBoss Applikationsserver ............. 18
   2.4.3 IBM WebSphere Applikationsserver (WAS) ... 19
   2.4.4 Andere Applikationsserver ................. 19

### 3 Das Framework - JavaServer Faces
3.1 Konfigurationsdateien ............................ 20
   3.1.1 Der deployment Deskriptor (web.xml) ........ 21
   3.1.2 Die Anwendungskonfigurationsdatei (faces-config.xml) .. 23
3.2 Der Lebenszyklus der Anfrageverarbeitung .......... 25
3.3 Das UI-Komponenten-Modell ..................... 28
3.4 Das Rendering-Modell ........................... 29
3.5 Die JSP-Tag-Bibliotheken ....................... 30
3.6 Managed Beans .................................. 32
3.7 Konvertierung .................................. 35
3.8 Validierung .................................... 36
3.9 Eventhandling .................................. 38
# Inhaltsverzeichnis

3.10 Das Navigations-Modell .................................................. 39  
3.11 Lokalisierung ................................................................. 41  

4 Das Equipment Management System ........................................ 43  
4.1 Einführung ................................................................. 43  
4.2 EMS - Szenarien ............................................................. 44  
   4.2.1 Die zentralen Aufgaben des Equipment Management System .... 44  
   4.2.2 Editiere-Geräte (Editing a device) ............................ 46  
   4.2.3 Editiere-Raumpläne (Editing room maps) ..................... 46  
   4.2.4 Editiere-Mitarbeiter (Editing an employee) ................. 46  
   4.2.5 Weitere Tools ..................................................... 47  
4.3 Die Architektur des EMS ................................................... 47  
4.4 Das Employee-Tool .......................................................... 49  
   4.4.1 Ausgangssituation ............................................... 49  
   4.4.2 Umsetzung und Realisierung .................................. 51  

5 Zusammenfassung, Bewertung und Ausblick ............................ 56  

A Anhang ............................................................................ 58  
B Literaturverzeichnis ........................................................ 59  
C Ehrenwörtliche Erklärung .................................................. 63
Vorwort

An dieser Stelle möchte ich all jenen danken, die durch ihre fachliche und persönliche Unterstützung zum Gelingen dieser Studienarbeit beigetragen haben.

Bedanken möchte ich mich auch bei dem EMS-Team. In zahlreichen Gesprächen und Diskussionen konnten viele Probleme gelöst und Fragen geklärt werden.

Meiner Freundin danke ich ganz besonders für ihr Verständnis und ihre Geduld, gerade im letzten Abschnitt meiner Studienarbeit.

Besonderer Dank gebührt meinen Eltern, die durch ihre Unterstützung und Korrekturlesen mir diese Studienarbeit ermöglicht haben.

Des weiteren bedanke ich mich bei Herrn Dr. Guido Gryczan für die Betreuung meiner Studienarbeit und den zahlreichen wissenschaftlichen Ratschläge, welche stets zur Verbesserung der Arbeit beigetragen haben.
<table>
<thead>
<tr>
<th>Abkürzung</th>
<th>Deutscher Begriff</th>
</tr>
</thead>
<tbody>
<tr>
<td>API</td>
<td>Application Programming Interface</td>
</tr>
<tr>
<td>CGI</td>
<td>Common Gateway Interface</td>
</tr>
<tr>
<td>DNS</td>
<td>Domain Name System</td>
</tr>
<tr>
<td>EAI</td>
<td>Enterprise Application Integration</td>
</tr>
<tr>
<td>EIS</td>
<td>Enterprise Information System</td>
</tr>
<tr>
<td>EJB</td>
<td>Enterprise Java Bean</td>
</tr>
<tr>
<td>EMS</td>
<td>Equipment Management System</td>
</tr>
<tr>
<td>GUI</td>
<td>Graphical User Interface</td>
</tr>
<tr>
<td>HTTP</td>
<td>Hypertext Transfer Protocol</td>
</tr>
<tr>
<td>J2EE</td>
<td>Java 2 Enterprise Edition</td>
</tr>
<tr>
<td>JAAS</td>
<td>Java Authentication and Authorization Service</td>
</tr>
<tr>
<td>JCA</td>
<td>J2EE Connector Architektur</td>
</tr>
<tr>
<td>JDBC</td>
<td>Java Database Connectivity</td>
</tr>
<tr>
<td>JMS</td>
<td>Java Message Service</td>
</tr>
<tr>
<td>JMX</td>
<td>Java Management Extensions</td>
</tr>
<tr>
<td>JNDI</td>
<td>Java Naming and Directory Interface</td>
</tr>
<tr>
<td>JSF</td>
<td>JavaServer Faces</td>
</tr>
<tr>
<td>JSP</td>
<td>Java Server Pages</td>
</tr>
<tr>
<td>JSR</td>
<td>Java Specification Request</td>
</tr>
<tr>
<td>JTA</td>
<td>Java Transaction API</td>
</tr>
<tr>
<td>LDAP</td>
<td>Lightweight Directory Access Protocol</td>
</tr>
<tr>
<td>MVC</td>
<td>Model-View-Controller</td>
</tr>
<tr>
<td>NIS</td>
<td>Network Information Service</td>
</tr>
<tr>
<td>PDA</td>
<td>Personal Digital Assistant</td>
</tr>
<tr>
<td>RI</td>
<td>Refernzimplementeierung</td>
</tr>
<tr>
<td>RMI</td>
<td>Remote Method Invocation</td>
</tr>
<tr>
<td>SVG</td>
<td>Scalable Vector Graphics</td>
</tr>
<tr>
<td>TCK</td>
<td>Technology Compatibility Toolkit</td>
</tr>
<tr>
<td>UI</td>
<td>User interface</td>
</tr>
<tr>
<td>URI</td>
<td>Uniform Resource Identifiern</td>
</tr>
<tr>
<td>URL</td>
<td>Uniform Resource Locator</td>
</tr>
<tr>
<td>WAR</td>
<td>Java Archiv für Web-Applikationen</td>
</tr>
<tr>
<td>WML</td>
<td>Website META Language</td>
</tr>
<tr>
<td>XHTML</td>
<td>Extensible Hypertext Markup Language</td>
</tr>
</tbody>
</table>
XML  Extensible Markup Language
Abbildungsverzeichnis

2.1 Drei-Schichten-Architektur (grob) ................................. 5
2.2 Drei-Schichten-Architektur (fein) ................................. 6
2.3 Model-View-Controller ............................................. 13
2.4 Die Model 1 Architektur ............................................ 15
2.5 Die Model 2 Architektur ............................................ 16

3.1 Der Lebenszyklus der Anfrageverarbeitung ..................... 26
3.2 Die UI-Komponenten der Referenzimplementation ............. 29

4.1 Die Benutzeroberfläche des EMS ................................. 45
4.2 Basis-Struktur der Benutzer-Interface-Komponenten des EMS .... 47
4.3 Die „works on“ relationship von Tools und Materialen .......... 48
4.4 Der Anmelde-Mechanismus der Workbench (EMS) ............... 50
4.5 Der Employee (RT) ist bereit zur Bearbeitung ................... 52
4.6 Die emsEdit.jsp ...................................................... 54
1 Einleitung

Diese Studienarbeit ist im Rahmen des Projektes „Objektorientierte Softwareentwicklung“ am Fachbereich Softwaretechnik der Universität Hamburg entstanden. Lernziele dieses 2 Semester dauernden Projektes waren neben der Einarbeitung in fundamentale Methoden und Techniken der objektorientierten Konstruktion, auch die Einarbeitung, Erweiterung und das Refactoring einer schon bestehenden Anwendung wie das „Equipment Management System“.\(^1\)


Entwickelt worden ist das „Equipment Management System (EMS)“ von der C1 WPS GmbH\(^3\) in Zusammenarbeit mit der Uni-Hamburg. In diesem Projekt wurde gezielt auf die Zusammenarbeit zwischen Studenten und erfahrenen Entwicklern geachtet.


1.1 Ausgangslage


\(^1\)(vgl. wps07b)  
\(^2\)(vgl. Zül05)  
\(^3\)C1 WPS Workplace Solutions Gesellschaft für DV-Beratung mbH
1.2 ZIEL


Die ursprüngliche Architektur des EMS hatte eine Client-Server Struktur, welche die komplette Geschäfts- und Präsentationslogik im Client vereinte. Diese Architektur galt es nun so zu überarbeiten und erweitern, das die neue Architektur des EMS eine im Sinne „Suns J2EE Spezifikation“ thin-client Architektur aufweist.

1.2 Ziel

Web-Browser sind auf dem Weg das universelle Front-End für viele WEB-Anwendungen zu werden. Motivation dieser Studienarbeit ist es zu untersuchen, inwieweit sich das Framework - JaverServer Faces für die Entwicklung eines browserbasierten Client Front-End

4(vgl. Gam01)
5(vgl. Mic07c)
eignet, für eine schon bestehende Anwendung wie das „Equipment Management System“. Dabei dient das Framework - JaverServer Faces als Grundgerüst einer Model-View-Controller (MVC)-Architektur, im Sinne „Suns J2EE Spezifikation“.


1.3 Übersicht

In dieser Arbeit werden in Kapitel 2 zunächst grundlegende Begriffe erläutert, um ein Verständnis für die Struktur und das Verhalten von Frameworks und deren Komponenten, zu erhalten. Dabei wird insbesondere auf die verschiedenen Architektur-Modelle von Suns „J2EE Spezifikation“ eingegangen, sowie die Einbettung der JSF-Applikation in einem Java Servlet Container, der die Laufzeitumgebung liefert.


In Kapitel 5 wird diese Arbeit zusammengefasst und die gewonnenen Erfahrungen kritisch beurteilt. Im Ausblick werden die aufgetauchten Fragen, Grenzen und Wünsche zusammengefasst.
2 Einführung und Grundbegriffe

2.1 J2EE - Java2 Plattform Enterprise Edition

Die Java2 Plattform Enterprise Edition (J2EE) definiert einen Standard, um mit modula-
ren Komponenten verteilte, mehrschichtige, skalierbare und plattformunabhängige un-
ternehmensweite Anwendungen zu entwickeln. Dabei versteht sich J2EE nicht als Pro-
dukt im eigentlichen Sinne, sondern spezifiziert lediglich einen allgemeinen Rahmen zur
Anwendungsentwicklung in der Programmiersprache Java. Ziel ist es, einfach anpassbare
und leicht zu administrierende Komponenten zu entwickeln, welche plattformübergrei-
fend eingesetzt und an existierende Informationssysteme angekoppelt werden können.

Hierzu werden bestimmte Aspekte der Umgebung, in der die Applikation abgewickelt
wird, als auch in der Umgebung, in der entwickelt wird, standardisiert. Dadurch wird
die Umgebung austauschbar, da man sie durch eine gleichwertige Komponente erset-
zein kann. Klar definierte Schnittstellen zwischen den Komponenten und der Umgebung
sollen dafür sorgen, dass Softwarekomponenten unterschiedlicher Hersteller interoper-
rabel sind und dass die verteilte Anwendung gut skalierbar ist. Dies kann natürlich nur
funktionieren, wenn sich die Entwickler an die ausgegebene Spezifikation halten.

Darüber hinaus bietet die J2EE-Spezifikation ein festgelegtes Rollenmodell, welches
die einzelnen Rollen der Entwickler bei der Entwicklung von J2EE-Applikationen defi-
niert. Dabei sollte die Rollenverteilung so gewählt werden, dass sich die Entwickler auf
ihre speziellen Kenntnisse im jeweiligen Fachgebiet konzentrieren können. Das ange-
strebte Ziel ist somit, Überschneidungen im Domänenwissen zwischen den Rollen zu
minimieren.¹

J2EE ermöglicht die Realisierung unternehmensweiter Anwendungen durch eine 3-
Schichten-Architektur. Ziel ist es die drei Aufgaben Präsentationslogik (engl. Presentation-
Logic), Geschäftslogik (engl. Business-Logic) und Datenhaltung voneinander zu trennen.
Die Abbildung 2.1 veranschaulicht die Aufteilung der 3-Schichten als „grob“ Darstel-
lung.²

Im Gegensatz zu den Client-Server Architekturen aus den frühen 90igern, welche le-
diglich „nur“ 2-Schichten (Client/Server) besaßen, führt die 3-Schichten-Architektur ei-

¹(vgl. Tur01)
²(vgl. Bor07)
KAPITEL 2. EINFÜHRUNG UND GRUNDBEGRIFFE

Abbildung 2.1: Drei-Schichten-Architektur (grob)


Die 3-Schichten-Architektur verlagert hingegen die Präsentationslogik und Geschäftslogik in die mittlere Schicht. Damit liegt die Applikationslogik auf dem Server und nur die Darstellungsschicht auf dem Client. Diese Art der Anordnung wird in der Literatur auch als thin-client bezeichnet⁴.

Einer der gewonnenen Vorteile aus der Verlagerung der Präsentations- und Geschäftslogik ist, dass Änderungen an der Applikationslogik nur einmal vorgenommen werden müssen und damit allen Clients sofort zur Verfügung stehen. Hierbei erfolgt die Wartung und Skalierung zentral. Abbildung 2.2 zeigt die 3-Schichten als „fein“ Darstellung.⁵

Die clientseitigen Zugriffe auf eine J2EE-Anwendung können auf unterschiedliche Art hergestellt werden. Dazu zählen vor allem:


³(vgl. Zöl06)
⁴(vgl. Zöl06)
⁵(vgl. Bor07)
2.1. J2EE - JAVA2 PLATTFORM ENTERPRISE EDITION

Abbildung 2.2: Drei-Schichten-Architektur (fein)

- Java Applikations-Clients, laufend auf einem Rechner (Desktop).
- Portable Geräte (Devices), wie PDA’s (personal digital assistant) oder Handy’s.

Die EIS (Enterprise Information System) Schicht repräsentiert ein so genanntes Unternehmensinformationssystem zur geordneten Verwaltung von Wissen und Geschäftsdaten. Typischerweise werden diese Datenbestände meist in einem relationalen Datenbanksystem gehalten.

Die Ankopplung solcher Informationssysteme ist Einsatzgebiet der Enterprise Application Integration (EAI) und geschieht mit einer J2EE „Connector-Architecture“ (JCA). Die J2EE Connector Architecture ist eine Software-Architektur, die eine Programmierschnittstelle (API) zur Verfügung stellt, um die Integration von heterogenen Anwendungen in die J2EE-Plattform zu ermöglichen.\(^6\)

Die mittlere Schicht (Middle Tier) ist stellvertretend für die Serverumgebung einer J2EE Web Applikation und kann in diverse logische Segmente unterteilt werden. Eine Java J2EE Web Applikation besteht dabei entweder aus Servlets oder JSPs und benötigen zur Ausführung einen Container. Ein Container ist eine Serverapplikation, die diese Web Applikation verwaltet und ausführt. Diese Container liefern die Laufzeitumgebung für die Komponenten.

\(^6\text{(vgl. Wik07c)}\)
Das Konzept des Containers soll eine einheitliche Sicht auf die von tiefer liegenden Schichten angebotenen Diensten (services), zur Verfügung stellen. Die aktuelle Spezifikation erfordert die folgenden Container:

- einen Web-Container als Laufzeitumgebung für Servlets und Java Server Pages (JSP).
- einen EJB-Container als Laufzeitumgebung für Enterprise Java Beans (EJB).
- einen JCA-Container als Laufzeitumgebung für JCA Connectoren.\(^7\)

Ein WEB-Container besteht meist aus einem Servlet oder ein JSP-generiertes Servlet und ist stellvertretend für die serverseitige Präsentationslogik.

Der EJB-Container stellt eine Laufzeitumgebung für Enterprise JavaBeans (EJB) zur Verfügung, welche die GeschäftslOGik einer Applikation in sich vereint. Der Container kümmert sich sowohl um die Speicherung der Daten, als auch um deren Verfügbarkeit für jeden autorisierten Client. Ein EJB-Container arbeitet als Vermittlungsschicht zwischen Client und Datenbank, daher spricht man auch von Middleware.

Der JCA-Container hält eine Laufzeitumgebung für die JCA Connectoren bereit und soll wie vorher schon beschrieben, die Integration von heterogenen Anwendungen in die J2EE-Plattform ermöglichen.

Container sind normalerweise keine eigenständig lauffähige Software, sondern Teil eines Applikation-Servers nach der J2EE-Spezifikation. Bekannte Applikation-Server sind:

- JBoss (Open Source)
- BEA WebLogic (kommerziell)
- IBM WebSphere (kommerziell)

Da J2EE den Anspruch erhebt, konsequent auf einem Komponentenmodell zu basieren und portable zu sein, können Komponenten somit unabhängig von der verwendeten J2EE-Plattform die gleichen Dienste erwarten.

Sowohl der WEB-Container als auch der EJB-Container bieten Dienste an, die den Komponenten ständig zur Verfügung stehen müssen:

- Namens- und Verzeichnisdienste

\(^7\)Dieser ist zwar nicht explizit definiert, faktisch jedoch muss jeder Application-Server-Hersteller diesen implementieren. Denn im Enterprise Java Beans (EJB) sowie im Web-Container sind Restriktionen definiert, welche für die JCA-Laufzeitumgebung nicht gelten. Dabei handelt es sich beispielsweise um das Starten von Threads oder das Lesen und Schreiben in Dateien etc.
2.1. J2EE - JAVA2 PLATTFORM ENTERPRISE EDITION

- Transaktionsdienst
- Konfigurationsdienst
- Kommunikationsdienst zwischen J2EE-Komponenten
- Sicherheitsdienst (Security)

Darüberhinaus müssen die Hersteller der J2EE-Container folgende Implementierungen der J2EE-APIs zur Verfügung stellen:8

- **JNDI**
  
  Das *Java Naming und Directory Interface* ermöglicht den Zugriff auf Namens- und Verzeichnisdienste durch Java-Programme. Beispielsweise über LDAP zu LDAP-Servern, JDBC Datenquellen, EJB Homes, JMS (Java Messaging Service) Connections, Network Information Service (NIS) und Domain Name System (DNS).

- **Servlet**
  
  Die *Servlet* API stellt Klassen und Interfaces zur Erstellung serverseitiger Web-Applicationen zur Verfügung. Im allgemeinen erlaubt das Java-Servlet-API die Erweiterung von Servern, deren Protokoll auf Anfragen und Antworten basiert.

- **EJB**
  
  *Enterprise Java Beans* sind Softwarekomponenten, die die Geschäftslogik einer Applikation beinhalten und den Zugriff auf persistente Daten gestatten. Die Beans laufen in einem EJB-Container ab.

- **JSP**
  

- **JMX**
  
  *Java Management Extensions* ermöglicht es, die einzelnen J2EE-Komponenten und Anwendungen zu verwalten.

- **JMS**
  
  *Java Message Service* ist eine API für die asynchrone Nachrichtenverarbeitung.

---

8Die Aufzählung enthält nur auszugsweise eine Beschreibung der wichtigsten APIs. Eine komplette Liste der Java Platform Enterprise Edition (Version 5.0) API Spezifikation finden Sie unter (Mic07b)
KAPITEL 2. EINFÜHRUNG UND GRUNDBEGRIFFE

- **JDBC**
  Die Java Database Connectivity API bewerkstelligt, das Java-Applikationen mittels SQL auf eine relationale Datenbank zugreifen können.

- **JTA**
  Die Java Transaction API definiert eine Schnittstelle, die es der Applikation erlaubt, die Transaktionsverwaltung zu steuern. Die JTA ist somit die Java-Schnittstelle zu den Transaktionsmonitoren und regelt den Zugriff von Applikationen auf gemeinsam genutzte Ressourcen wie Datenbanken oder Message-Systeme.

- **JAAS**
  Der Java Authentication and Authorization Service ist eine Java-API, die es ermöglicht, Dienste zur Authentifikation und Zugriffsrechte in Java-Programmen bereitzustellen.

- **JCA**
  J2EE Connector Architecture ermöglicht die Integration non-Java Applikationen. Stichwort: Enterprise Application Integration (EAI).

- **JAXP**
  Die Java API for XML Processing hilft dem Entwickler bei der Bearbeitung von XML-Dokumenten und enthält eine Reihe von Möglichkeiten zum Validieren und Parsen.

- **JSF**

- **RMI-IIOP**
  RMI-IIOP ist die Abkürzung für Java Remote Method Invocation over the Internet Inter-ORB-Protocol und ist der Mechanismus, der die Kommunikation zwischen Java-Objekten in verschiedenen virtuellen Maschinen ermöglicht. RMI-IIOP ist, im Unterschied zu RMI (Remote Method Invocation) von Java, mit der API java.rmi zu CORBA kompatibel und ermöglicht so die Interaktion von CORBA (Common Object Request Broker Architecture) Objekten in eine J2EE-Applikation.

- **JavaMail**
2.2 Java Server Pages und Servlets


Das Framework JSF ist dabei nicht ausschließlich an die Java Server Pages als Präsentationsschicht gebunden, mögliche Optionen sind Renderkits für mobile Endgeräte, die beispielsweise WML oder XHTML Basic unterstützen. Dennoch lässt sich zunächst ein Trend verfolgen, der sich auf HTML-Web-Applikationen auf Basis von JSP konzentrieren wird.

Grundsätzlich lässt sich über beide Technologien (JSP und Servlets) sagen, dass sie das gleiche Ziel verfolgen, es nur auf unterschiedliche Weise erreichen. Beide Technologien sind Server-Programme, die dynamische HTTP Web-Seiten generieren und diese zum Client, als Folge einer Anfrage, zurückschicken.


---

9 Die Java Server Pages Technologie dient als Referenzimplementierung für die Präsentationsschicht im Framework JSF.
10 (vgl. Hai04, S.37)
11 Common Gateway Interface (CGI), ist eine schon länger bestehende Variante, Webseiten dynamisch bzw. interaktiv zu machen. (vgl. Wik07a)
12 (vgl. Wik07e)
KAPITEL 2. EINFÜHRUNG UND GRUNDBEGRIFFE

Die Java Server Pages (JSP) ist eine von Sun Microsystems entwickelte Technologie (Architektur), die die Generierung von serverseitigem dynamischem Inhalten erlaubt und konzeptionell vergleichbar mit der ASP.NET Technologie von Microsoft ist.\(^{13}\)

Eine JSP Seite ist hierbei ein textbasiertes Dokument, welches sowohl statischen als auch dynamischen Inhalt besitzen kann und für gewöhnlich die Erweiterung *\.jsp trägt. Fragmente, also Teile einer JSP Seite, die in andere einfließen, haben die Erweiterung *.jspx.

Der statische Inhalt einer JSP Seite kann aus einem Format wie HTML, SVG, WML oder XML gebildet werden und besteht aus all jenen Elementen, die von einem Webserver in die HTTP-Response (Antwort) ohne Veränderung übernommen werden (z.B. HTML-Tags).\(^{14}\) Der dynamische Inhalt wird in der Regel in Form von eingebetteten Java-Code in die JSP-Seite eingebunden.


JSP-Seiten werden vom Web-Server automatisch in Servlets übersetzt. Der Übersetzungsvorgang von JSP-Seiten in ein Servlet muss dann nur einmal getätigt werden, danach benutzt der Servlet-Container direkt die übersetzte Klasse.


Eine nahe liegende Schlußfolgerung wäre es daher, für die Web-Anwendungsentwicklung die prozedural aufgebauten, in Java geschriebenen Servlets frei von HTML-Anweisungen (Präsentationslogik) zu lassen und die JSP-Dokumente frei von Java-Code (Geschäftslogik). Und genau dies wird durch eine strikte Trennung der beiden Technologien in spezielle exclusive Aufgabenbereiche erreicht.

Servlets werden oft im Rahmen der J2EE-Spezifikation nach dem Model-View-Controller-Pattern (MVC)\(^{15}\) in Form von JSP verwendet. Während sich die Servlets nur auf

---

\(^{13}\)Ähnliche Ergebnisse können auch von den Skriptsprachen wie PHP oder Perl erzielt werden, diese sind aber keine Architekturen.

\(^{14}\)(vgl. Wik07d)

\(^{15}\)MVC steht für das Model-View-Controller Paradigma und wurde schon bei der Programmierung mit
2.3. ARCHITEKTUREN

die Ablaufsteuerung und Controllerfunktionalität konzentrieren, werden die JSPs einzig als Präsentationsschicht eingesetzt.

Das MVC-Paradigma spielt eine entscheidende Rolle, um das Zusammenwirken der Technologien Java Server Pages und Servlets und die Trennung von Präsentations-, Geschäfts- und Datenlogik zu verstehen. Es wird daher im nächsten Kapitel noch ausführlich betrachtet.

Damit die JSP-Seiten als Präsentationsschicht eingesetzt werden können, sollten inhaltlich keine Vermischungen mehr von Präsentationslogik und Geschäftslogik stattfinden. Da aber dennoch die Möglichkeit bestehen soll dynamischen Inhalt einzubringen, wurde der eingebettete Java-Code ausgelagert. Dies wurde durch die Einführung spezieller JSP-Tags erreicht.

Bei den JSP-Tags handelt es sich um spezielle Tags in XML-Notation, die eine vordefinierte Funktionalität liefern und in speziellen Tag-Bibliotheken (Tag-Library) gehalten werden. Tag-Bibliotheken sind ein integraler Bestandteil der JSP-Spezifikation.

Das Konzept der Tag-Bibliotheken ermöglicht somit die Auslagerung von Java-Code aus den JSP Seiten. Dadurch ist es möglich, JSP-Seiten zu entwickeln, die nur noch wenig bis gar keinen Java-Code beinhalten.\footnote{Smalltalk erfolgreich eingesetzt. Die J2EE-Community hat das MVC-Paradigma entsprechend ihren Gegebenheiten angepasst.}


2.3 Architekturen


Das Model-View-Controller (MVC) Architekturmuster für Web-Anwendungen wurde aus dem bekannten Model-View-Controller-Paradigma abgeleitet, das schon bei der Entwicklung von Benutzeroberflächen in Smalltalk angewandt wurde. Die Model 1- und Mo-
KAPITEL 2. EINFÜHRUNG UND GRUNDBEGRIFFE

del 2- Architektur sind eine für Webanwendungen spezialisierte Variante des Architekturmusters MVC und werden im diesem Kapitel noch genauer vorgestellt.

2.3.1 Das MVC-Paradigma


Abbildung 2.3: Model-View-Controller
2.3. ARCHITEKTUREN

- **Datentmodell (Model)**

- **Präsentation (View)**

- **Programmsteuerung (Controller)**
  Der Controller bestimmt das Verhalten, wie auf die Benutzeraktionen reagiert wird. Der Controller steuert und verwaltet die Views, nimmt die Benutzeraktionen entgegen, wertet diese aus und agiert entsprechend. Der Controller ändert abhängig vom Benutzerverhalten das Model oder ruft andere Views auf. Jedem Controller ist dabei genau eine View zugeordnet und mehrere Controller einem Model. Der Controller besitzt die Intelligenz und steuert somit den Ablauf der Präsentation.

### 2.3.2 JSP-basierte Model 1-Architektur

Die Model 1 Architektur ist eine 2-Schichte Architektur, die häufig nur für einfach übersehbar und kleine Web-Anwendungen verwendet wird. Die Model 1 Architektur trennt nicht die Zuständigkeiten nach dem MVC-Paradigma auf. In einem JSP-basierte Modell liegen die Zuständigkeiten hauptsächlich bei den JSP-Seiten und den Java Beans. Den Aufbau einer JSP-basierte Model 1 Architektur zeigt Abbildung 2.4.

Bei dieser Art von Aufbau ist die JSP-Seite sowohl für die Darstellung (View), als auch für die Programmsteuerung (Controller) zuständig. Die Java Bean kapselt den aktuellen Zustand und enthält das Datenmodell (Model). Charakteristisch für diese Architektur war, dass für nahezu jede angezeigte Seite eine eigene JSP-Seite notwendig war und dass die Navigation zumeist fest (dezentralisiert) in der JSP-Seite eingebunden wurde.
KAPITEL 2. EINFÜHRUNG UND GRUNDBEGRIFFE

Abbildung 2.4: Die Model 1 Architektur


2.3.3 JSP-basierte Model 2 Architektur


Innerhalb der Model 2 Architektur sind die JSPs nicht mehr für die Ablaufsteuerung (Controller) verantwortlich, sondern nur noch für die eigentliche Bildschirmrepräsentation (View) zuständig. Die Zuständigkeiten der Model-, View- und Controller-Komponenten sind wie im Kapitel Das MVC-Paradigma schon beschrieben, dem MVC-Paradigma
2.3. ARCHITEKTUREN

Abbildung 2.5: Die Model 2 Architektur

entsprechend verteilt. Durch die zunehmende Komplexität wächst der anfängs größere Arbeitsaufwand, wird aber durch die gewonnenen Vorteile wie Wiederverwendbarkeit, Erweiterbarkeit und Skalierbarkeit mehr als wieder wettgemacht.

2.3.4 Die Umsetzung des MVC-Paradigma im Framework - JavaServer Faces


2.4 Laufzeitumgebung


Da Webbrowser besonders beliebte thin-clients für Applikationsserver darstellen, sind die meisten Applikationsserver entweder Ergänzungen zu Webservern oder enthalten einen Webserver (z. B. Tomcat in JBoss, WebSphere oder WebLogic). In den folgenden Abschnitten sollen die gängigsten Applikationsserver vorgestellt werden.

2.4.1 Jakarta Tomcat Server


Der Tomcat Server besteht seit Version (4.x) aus zwei Hauptkomponenten Jasper (JSP-Compiler) und Catalina (Servlet Container). Mit Hilfe des JSP-Compilers Jasper können

\footnote{Informationen über die Apache-Foundation (vgl. Fou07c)\footnote{Informationen über das Jakarta-Projekt (vgl. Fou07b)}}
Java Server Pages in Servlets übersetzt und im Servlet Container Catalina zur Ausführung gebracht werden. Da Jasper eine eigenständige Komponente und als Servlet realisiert ist, besteht auch die Möglichkeit, ihn in anderen Servlet Container einzusetzen.

Der Catalina Servlet Container ist komplett in Java implementiert. Er deckt aktuell die Referenz Implementierung der Servlet (2.3) und JSP (1.2) Spezifikationen von Sun Microsystems ab. Der Catalina Servlet Container ist der im Jakarta-Projekt für Tomcat entwickelte Nachfolger von Apache JServ.

Tomcat bietet, wie fast alle Servlet Container einen eigenen, integrierten HTTP-Server an, der es erlaubt, neben dynamischen Inhalten, statische Elemente zu übertragen.


2.4.2 Der JBoss Applikationsserver


Für viele Web-Anwendungen würde auch nur der Tomcat mit seiner Funktionalität reichen.
KAPITEL 2. EINFÜHRUNG UND GRUNDBEGRIFFE

2.4.3 IBM WebSphere Applikationsserver (WAS)

Bei dem von IBM kommerziell vertriebenen WebSphere Applikationsserver (WAS) handelt es sich wie bei JBoss um einen Server, der die J2EE Spezifikation von Sun vollständig erfüllt. Die Plattform unterstützt Webservices und die moderne J2EE-Architektur, soll eine hervorragende Portabilität, ausgezeichnete Performance, höchste Skalierbarkeit und eine sehr gute Anschlussfähigkeit an andere Systeme garantieren. 21


2.4.4 Andere Applikationsserver

Neben den eben vorgestellten Applikationsserver (Open-Source bzw. kommerziell) sollen aber auch noch andere bekannte Firmen genannt werden, die sich einen Namen auf dem Applikationsserver Markt gemacht haben. Ferner konnten sich im kommerziellen J2EE Bereich Oracle mit dem Oracle Application Server, SUN iPlanet und BEA WebLogic positionieren. Bestehenden Open-Source-Alternativen zu JBoss sind Geronimo22, Glassfish23 und JOnAS. 24

21nach eigenen Angaben von IBM (vgl. IBM07)
22Apache Geronimo wird unter Federführung der Apache Software Foundation (ASF) entwickelt. (vgl. Fou07a)
23SUNs Open Source Java EE 5 Application Server. (vgl. Mic07a)
24Java Open Application Server. (vgl. Web07)
3 Das Framework - JavaServer Faces


In den folgenden Abschnitten werden alle relevanten Themen behandelt, die sich mit der Zusammensetzung und der Funktionalität des Frameworks JSF beschäftigen. Dazu zählen Themen wie die Konfigurationsdateien, der Lebenszyklus der Anfrageverarbeitung, das UI-Komponenten-Modell, das Rendering-Modell, die Validierung, das Eventhandling, das Navigations-Modell, die JSF-Tag-Bibliotheken, die Managed Beans und die Lokalisierung.

3.1 Konfigurationsdateien

Beim Aufbau großer WEB-Applikationen spielen die Konfigurationsdateien eine wichtige Rolle, um das System bzw. einzelne Komponenten optimal zu konfigurieren und an die systemspezifischen Gegebenheiten perfekt anzupassen.


Eine weitere nicht weniger unwichtige Datei ist die faces-config.xml. In dieser Konfigurationsdatei werden alle Parameter definiert, die für die JSF-Applikation notwendig sind. Vornehmlich werden hier alle Einstellungen vorgenommen, die sich mit dem Verarbeitungsprozess der ankommenden Requests an das FacesServlet beschäftigen. Dieser Verarbeitungsprozess kann in Phasen unterteilt werden, wird deswegen auch „Der Lebenszyklus der Anfrageverarbeitung“ genannt und soll später noch genauer beschrie-

\(^1\) (vgl. Hai04)
KAPITEL 3. DAS FRAMEWORK - JAVASERVER FACES

ben werden. Es sollen nun folgend beide Konfigurationsdateien vorgestellt werden.

3.1.1 Der deployment Deskriptor (web.xml)

Bei einer JSF-Applikation muss immer innerhalb der Datei web.xml, das FacesServlet de-
finiert werden. Dieses FacesServlet fungiert als zentraler Controller und nimmt alle An-
fragen an die JSF-Applikation entgegen. Das folgende Code-Beispiel zeigt auszugsweise
den typischen Aufbau einer Datei web.xml.

```xml
<web-app>
  ...
  <!-- Faces Servlet -->
  <servlet>
    <servlet-name>Faces Servlet</servlet-name>
    <servlet-class>javax.faces.webapp.FacesServlet</servlet-class>
    <init-param>
      <param-name>config</param-name>
      <param-value>/WEB-INF/faces-config.xml</param-value>
    </init-param>
    <load-on-startup>0</load-on-startup>
  </servlet>

  <servlet-mapping>
    <servlet-name>Faces Servlet</servlet-name>
    <url-pattern>*.faces</url-pattern>
  </servlet-mapping>
</web-app>

Im Folgenden sollen nun alle relevanten Elemente im einzelnen beschrieben werden.

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

Dieser Bereich definiert das FacesServlet, das in der Klasse javax.faces.webapp.FacesServlet
tenhalten ist und folgende Elemente enthält:

```xml
<servlet-name>Faces Servlet</servlet-name>
<servlet-class>javax.faces.webapp.FacesServlet</servlet-class>
```
Faces Servlet ist der Name des Servlets und javax.faces.webapp.FacesServlet ist der qualifizierende Java Klassenname des JavaFaces Kontrollers.

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


```xml
<load-on-startup>0</load-on-startup>
```

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

```xml
<servlet-mapping>
  <servlet-name>Faces Servlet</servlet-name>
  <url-pattern>*.faces</url-pattern>
</servlet-mapping>
```

Die Konfiguration des <servlet-mapping> beschreibt, in welchen Fällen der FacesServlet Controller eine Anfrage entgegen nimmt und eine Response generiert. Es gibt grundsätzlich zwei verschiedene Möglichkeiten zu definieren, welche URLs vom Controller verarbeitet werden: Extension-Matching und Präfix-Matching:


http://localhost:8080/EMS/login.jsp

**Präfix-Matching** Das Präfix-Matching leitet alle URLs, die nach dem Kontextpfad der Applikation mit einem bestimmten Wert anfangen, an den Controller weiter. Die Konfiguration des <servlet-mapping> würde dann wie folgt aussehen.
Die Anfrage vom WEB-Browser an den Controller würde dann so aussehen.

http://localhost:8080/EMS/faces/login

Grundsätzlich lässt sich festhalten, das die Art der Verwendung des Matching keinen Einfluss auf das Ergebnis hat, bzw. auf die Verarbeitung durch das FacesServlet.

### 3.1.2 Die Anwendungskonfigurationsdatei (faces-config.xml)


- **Application:**
  Hier können Einstellungen vorgenommen werden, die für die gesamte Web-Applikation gültig sind. Pro Web-Applikation gibt es immer nur eine Instanz, das heißt, dass die Applikation-Instanz als Singleton implementiert ist. Durch den modularen Aufbau können beinahe alle Klassen, die für die Steuerung nötig sind, durch Benutzer-definierte Klassen ausgetauscht werden.

- **Factory:**

- **Component**
  Selbstantwickelte benutzerdefinierte UI-Komponenten können über das `<component>`-Element der Web-Applikation zugänglich gemacht werden. Dabei werden die UI-Komponenten durch das `<component>`-Element registriert und stehen fortan der Applikation zur Verfügung.

²(vgl. Hai04)
3.1. KONFIGURATIONSDATEIEN

- **Converter**
  Mit dem `<converter>`-Element hat der Web-Entwickler die Möglichkeit, die Standard-Konverter um seine eigenen oder fremd entwickelten Konverter zu erweitern. Alle hier registrierten Konverter haben eine `<converter-id>`, über die sie eindeutig zu identifizieren sind.

- **Managed-bean**
  Das `<managed-bean>`-Element dient zur Registratur aller Java Beans einer JSF-Applikation. Sobald eine Managed-Bean zum erstenmal über eine JSP-Seite ange-sprochen wird, wird automatisch dynamisch eine Instanz von ihr erzeugt. Bei der Definition der Managed-Beans besteht auch die Möglichkeit, sie über ihre Properties vorzuintialisieren und anzugeben, in welchen Scope (Session, Request, Application oder none) sie abgelegt werden soll.

- **Navigation-rule**
  Jedes `<navigation-rule>`-Element repräsentiert eine individuelle Entscheidungsre-gel, die bestimmt, welche Seite als nächstes angezeigt werden soll. Abhängige Fak- toren für den Navigations-Fall sind, für welche aktuelle Seite gilt der Navigation-Fall `from-view-id`, der Rückgabewert vom NavigationHandler `from-outcome` und die nächste zu rufende Seite `to-view-id`.

- **Referenced-bean**
  Das Element `<referenced-Bean>` ist eine Referenz auf ein Java-Objekt, welche ei-ne konkrete Klasse, abstrakte Klasse oder ein Interface sein kann, das zur Laufzeit der Applikation unter dem Key `referenced-Bean-name` und Scope verfügbar sein soll.

- **Render-kit**
  Das `<render-kit>`-Element beschreibt die Auswahl des RenderKits und entschei-det damit über die Wahl der gewählten Bildschrimrepräsentation (HTML, XHTML, WML etc.) einer UI-Komponente. Innerhalb einer Applikation kann das Render-Kit über `<render-kit-id>` referenziert und benutzt werden. Der Zugriff erfolgt dann über die Methoden der RenderKit-Factory Klasse.

- **Lifecycle**
  Mir dem `<lifecycle>`-Element hat der Entwickler die Möglichkeit Veränderungen an der Standard-Anfrageverarbeitung vorzunehmen.

- **Validator**

### 3.2 Der Lebenszyklus der Anfrageverarbeitung


Bei einer Anfrage (engl. *Request*) an eine JSF-Applikation kann es grundsätzlich zu folgenden unterschiedlichen Szenarien kommen:

1. **Ein Non-Faces Request erzeugt eine Faces Response.**
   

2. **Ein Faces Request erzeugt eine Faces Response.**
   
   Eine *Faces Response* ist eine Antwort einer JSF-Applikation, die erzeugt wurde, durch die letzte Phase der Anfrageverarbeitung, die „*Render Response*“-Phase. Durchläuft die Anfrageverarbeitung nicht die letzte Phase, so spricht man von einer *Non-Faces Response*.

3. **Ein Faces Request erzeugt eine Non-Faces Response.**
   


Der Lebenszyklus der Anfrageverarbeitung enthält folgende Phasen:

---

3 (vgl. Hai04, S. 33ff.)

4 (vgl. Hai04, S. 43)
3.2. DER LEBENSDZKULUS DER ANFRAGEVERARBEITUNG

Abbildung 3.1: Der Lebenszyklus der Anfrageverarbeitung

1. **Wiederherstellung des UI-Komponenten-Baumes (engl. Restore View)**
   
   In der ersten Phase wird die JSP-Seite nach UI-Komponenten durchsucht und die gefundenen UI-Komponenten in eine Baum-Struktur gebracht (wiederhergestellt). An den Komponenten registrierte Validierungs-Klassen, Event-Listener oder Konverter würden dementsprechend auch als Elemente im Baum auftauchen. Die Aufteilung in eine Baum-Struktur hat den Vorteil, dass eine Applikation später sehr bequem und einfach auf die einzelnen Elemente im Baum zugreifen kann, um sie gegebenenfalls zu verändern. Die erstellte fertige Baum-Struktur wird im Anschluss im FacesContext abgelegt und ist damit später für weitere Verarbeitungsschritte jederzeit zugänglich.

2. **Übernahme der Anfrage-Werte (engl. Apply Request Value)**
   


4. Aktualisierung der Model-Objekte (engl. Update Model Values)

Die validierten Werte können nach ihrer Überprüfung an die entsprechenden Model-Objekte übertragen werden. Dabei findet unter Umständen ein Standard oder vom Benutzer explizit gewollte Konvertierung statt, die die in der Bildschirmpräsentation verwendeten Datentypen, auf die im Model verwendeten umwandelt.

5. Aufruf der Applikation (engl. Invoke Application)


6. Rendern der Response (engl. Render Response)


3.3 Das UI-Komponenten-Modell


Eine UI-Komponente kann hierbei ein einfacher Button oder eine zusammengesetzte Tabelle, die aus mehreren Einzelkomponenten bestehen kann, sein. UI-Komponenten sind leicht konfigurierbare, durch Komposition erweiterbare und wieder verwendbare Komponenten, aus denen das Benutzer-Interface einer JSF-Applikation gebildet wird. Zugänglich werden die UI-Komponenten über sogenannte JSF Custom Tags innerhalb einer JSP-Seite gemacht.


Festzuhalten bleibt, dass alle UI-Komponenten eins gemeinsam haben, sie definieren lediglich die Funktionalität einer UI-Komponente, lassen aber die Entscheidung ihrer Repräsentation noch offen. Das betrifft sowohl die Art der Darstellung einer UI-Komponente (eine UI-Komponente wie UISelectMany kann z.B. in HTML durch mehrere Check-boxen oder durch ein Drop-Down-Menü dargestellt werden.), als auch die benutzte Auszeichnungssprache (HTML, XHTML, WML etc.). Wie eine UI-Komponente dargestellt wird und in welcher Auszeichnungssprache, wird durch den benutzten Renderer entschieden.

KAPITEL 3. DAS FRAMEWORK - JAVASERVER FACES

Abbildung 3.2: Die UI-Komponenten der Referenzimplementierung

3.4 Das Rendering-Modell


Gleichartige Renderer werden in einem RenderKit zusammengefasst. Die JavaServer Faces Referenzimplementierung enthält bereits ein Standart-RenderKit, welches die UI-

29
Komponenten für ein HTML-Client darstellt.

3.5 Die JSF-Tag-Bibliotheken


Das folgende Code-Fragment zeigt auszugsweise den Aufbau und Umgang mit den Tags.

```html
<%@ taglib uri="http://java.sun.com/jsf/html" prefix="h" %>
<%@ taglib uri="http://java.sun.com/jsf/core" prefix="f" %>

<html>
<body>
  <f:view>
    <h:form>
      <h:outputText value="Username: "/>
      <h:inputText id="username" value="#{loginBean.username}"/>
      <h:outputText value="Passwort: "/>
      <h:inputSecret id="password" value="#{loginBean.password}"/>
      <br><br>
      <h:commandButton value="Abschicken>>" action="#{loginBean.getLoginResult}" actionListener="#{loginBean.login}"/>
    </h:form>
  </f:view>
</body>
</html>
```
Damit die JSF-Tag-Bibliotheken benutzt werden können, müssen sie am Anfang einer JSP-Seite mittels einer JSP-Direktive eingebunden werden.

```html
<%@ taglib uri="http://java.sun.com/jsf/html" prefix="h" %>
<%@ taglib uri="http://java.sun.com/jsf/core" prefix="f" %>
```

Das `uri`-Attribut definiert die Lage der Tag-Bibliothek. Das `prefix`-Attribut stellt einen Namensraum dar.

```html
<f:view>
  <h:form>
    ...
  </h:form>
</f:view>
```


```html
<h:inputText id="username" value="#{loginBean.username}"/>
```

- `id`
  Das Attribut `id` identifiziert die Komponente eindeutig.

- `value`
  Das Attribut `value` bindet den Wert der Komponente an die Property eines Model-Objekts.

Das `inputText`-Tag definiert eine UIInput-Komponente (Text-Eingabe-Feld) mit den Attributen `id` und `value`. Dabei wird durch das Attribut `id` die Komponente eindeutig identifiziert. Es dürfen keine zwei Komponenten die gleiche ID haben. Wenn keine ID
3.6 Managed Beans

Die Model-Objekte werden in JavaServer Faces durch Beans realisiert, genauer gesagt durch Managed-Beans. Eine Managed-Bean besteht dabei aus einer Menge von Properties und Methoden. Die Properties können sich entweder auf Werte von UI-Komponenten...
oder direkt auf Instanzen dieser Komponenten beziehen. Die Methoden einer Managed-
Bean können dabei verarbeitende Aufgaben übernehmen. Das können Aufgaben sein
wie die Validierung, das Event Handling und die Vorbereitung der Navigation.

Bevor eine Managed-Bean referenziert werden kann, muss sie in der JSF-Konfigura-
tionsdatei `faces-config.xml` bekannt gemacht werden. Der Aufbau folgt grundsätzlich fol-
gendem Schema:

```xml
<managed-bean>
  <description>
    Die loginBean überprüft das Login.
  </description>
  <managed-bean-name>loginBean</managed-bean-name>
  <managed-bean-class>
    de.itwps.ems.web.bean.EMSBeanLogin
  </managed-bean-class>
  <managed-bean-scope>request</managed-bean-scope>
</managed-bean>
```

- `<description>`
  Die `<description>` enthält eine Beschreibung über das Einsatzgebiet der Managed-
  Bean. (optional)

- `<managed-bean-name>`
  Der `<managed-bean-name>` ist ein eindeutiger Identifikator und dient zur Refe-
  renzierung, innerhalb der JSP-Seiten, einer Managed-Bean.

- `<managed-bean-class>`
  Die `<managed-bean-class>` gibt den vollständigen Klassenpfad zu der Implemen-
  tierung der Managed-Bean an.

- `<managed-bean-scope>`
  Hier wird der Scope eingestellt, in der sich die Managed-Bean befinden soll. Zuläs-
  sige Parameter sind: „request“, „session“, „application“ oder „none“.

Nachdem die Managed-Bean bekannt gemacht worden ist, kann sie zum Management
von UI-Komponenten eingesetzt werden. Zu klären bleibt: Wie wird die Bindung von UI-
Komponenten bzw. deren Wert an Properties der Managed-Bean erreicht und wie wer-
den die Methoden einer Managed-Bean über die UI-Komponenten-Tags referenziert.
3.6. MANAGED BEANS

Die Bindung von UI-Komponenten bzw. deren Wert an Properties der Managed-Bean, kann auf einer JSP-Seite auf zwei Arten hergestellt werden. Zum einen wird über das bindig-Attribut der Tags direkt eine Instanz einer Komponente in der Managed-Bean referenziert, zum anderem wird über das value-Attribut der Tags nur der Wert einer Komponente referenziert.

```xml
<h:inputText id="username"
    value="#{loginBean.username}"/>
```


Das referenzieren einer Methode einer Managed-Bean über die UI-Komponenten-Tags, wird über den Mechanismus des Method Binding Expressions erreicht. Dabei werden nicht wie beim Value Binding Expressions Properties einer Managed-Bean gebunden, sondern ihre Methoden.

```xml
<h:commandButton value="Abschicken>>"
    action="#{loginBean.getLoginResult}"
    actionListener="#{loginBean.login}" />
```

Wie schon erwähnt, können die Methoden einer Managed-Bean verschiedene Aufgaben übernehmen, wie z.B. auch das Event Handling. Mit dem actionListener-Attribut wird eine Methode referenziert, die die ActionEvents verarbeiten kann. Die Methode login der Managed-Bean loginBean wird also dementsprechend aufgerufen, nachdem der Benutzer den Button Abschicken gedrückt hat. Die Methode soll die Verarbeitung des Logins bewerkstelligen oder zu mindestens an Instanzen delegieren, die die Verarbeitung übernehmen. Die referenzierte Methode, die ActionEvents verarbeiten kann, hätte in unserem Beispiel folgendes aussehen.

```java
public void login(ActionEvent event) {

    [...] 
    checkUserAndPassword( getUsername(), getPassword() )
    [...] 
}
```
3.7 Konvertierung


Um diese Konvertierung zu ermöglichen, können sogenannte Konverter (engl. Converter) an den UI-Komponenten registriert werden. Dabei stellt die JSF Implementierung schon eine Reihe von Konvertern, die automatisch eine Konvertierung vornehmen und somit dem Entwickler eine menge Arbeit im Hintergrund abnehmen. Aber nicht immer reichen die angebotenen Konverter aus und deshalb besteht die Möglichkeit, sie um eigene Entwickelte zu erweitern.

Mit dem Interface javax.faces.convert.Converter liefert die JSF-Implementierung eine einfache Möglichkeit, Konverter zu erzeugen. Alle Konverter müssen dieses Interface implementieren. Das Interface definiert die zwei Methoden getAsObject() und getAsString() die von dem Entwickler nur implementiert werden müssen:

• getAsObject()

• getAsString()
  Die Methode getAsString() wandelt im Unterschied dazu die Properties eines Model-Objekts in einen String-Wert um. Das heißt für unser Beispiel, das der Datentyp java.util.Date zu einem String-Wert umgewandelt wird. Bei eigenen entwickelten Konvertern ist die Implementation dieser Methode meist die einfachere.

3.8 Validierung


```xml
<h:inputText id="password"
    value="#{loginBean.password}"
    required="true">
    <f:validateLength minimum="8"/>
</h:inputText>
```

Der Benutzer wird aufgefordert ein Passwort einzugeben, das mindestens aus acht Zeichen besteht. In diesem Beispiel wird eine Standard-Validierungs-Klasse validateLength benutzt. Das minimum-Attribut definiert, was die minimale Länge des Eingabe-Stings haben darf. Durch das required-Attribut des inputText-Tags kann entschieden werden, ob die „Leere“-Eingabe auch eine gültige ist. In unserem Fall bestehen wir auf eine Eingabe durch den Benutzer (required="true"). Demzufolge darf das Eingabefeld nicht leer sein.


```xml
<h:inputText id="password"
    value="#{loginBean.password}"
    required="true">
    <f:validator validatorId="PasswordValidator"/>
</h:inputText>
```


```xml
<validator>
    <description>Überprüft die Länge des Passwortes</description>
</validator>
```
Genau wie bei der Registrierung der eigenen Konverter, werden auch die Validatoren innerhalb der JSF-Konfigurations-Datei bekannt gemacht. Die dem entsprechende Klasse *PasswordValidator* sieht dann wie folgt aus.

```java
public class PasswordValidator implements Validator {

    public void validate(FacesContext context, UIComponent component, Object value)
            throws ValidatorException {

        String stringValue = (String) value;
        int laengeDerEingabe = stringValue.length();

        if (laengeDerEingabe < 8) {
            FacesMessage message = new FacesMessage("Die von ihnen gewählten länge des Passwortes ist zu kurz...");
            throw new ValidatorException(message);
        } else {
            //ok
        }
    }
}
```

Eigene Validierungs-Klassen müssen nur das Interface *Validator* mit der Methode *validate* implementieren. Die Variable *value* vom Typ *Object* liefert uns die Eingabe des Passwortes. Dieser Datentyp *Object* wird zunächst in einem String-Wert umgewandelt und dessen Länge wird bestimmt. Anhand des Entscheidungsfalls wird die eigentliche Überprüfung, ob das Passwort mindestens aus acht Zeichen besteht, entschieden. Wenn das Passwort aus acht oder mehr Zeichen besteht, ist die Eingabe korrekt und es muss nichts weiter passieren. Ist die Eingabe-Länge des Passwortes kleiner als acht Zeichen,
wird eine Fehler-Message (FacesMessage) erzeugt, die dann durch eine ValidatorException weitergereicht wird. Diese message kann dem Benutzer auf der JSP-Seite angezeigt werden, der dann dementsprechend reagieren kann.

3.9 Eventhandling


actionListener="#{myBean.myMethod}"


Eine referenzierte Methode müsste z.B. um das ActionEvent per Method Binding Expressions verarbeiten zu können, die Form besitzen:

```java
public void methodenName{ActionEvent event}
{
    //Event-Verarbeitung
}
```
Die Implementierung des Interfaces ActionListener und der einzigen Methode action-Performed einer eigenen entwickelten Listener-Klasse, würde wie folgt aussehen:

```java
public class myActionListener implements ActionListener
{
    public void actionPerformed(ActionEvent event)
    throws AbortProcessingException
    {
        //Event-Verarbeitung
    }
}
```

### 3.10 Das Navigations-Modell


```html
<h:commandButton value="Abschicken>>"
    action="#{loginBean.getLoginResult}"
    actionListener="#{loginBean.login}" />
```


```java
public void login(ActionEvent event) {
    if ( checkUserAndPassword( getUsername(), getPassword() )
    {
        setLoginResult("erfolgreich"); //Login ok
```

```xml
<navigation-rule>
  <description>
    Was soll nach der Anmeldung passieren? (erfolgreich/misserfolg)
  </description>
  <from-view-id>/login.jsp</from-view-id>
  <navigation-case>
    <description>
      Anmeldung erfolgreich, weiter mit der Seite willkommen.jsp
    </description>
    <from-outcome>erfolgreich</from-outcome>
    <to-view-id>/willkommen.jsp</to-view-id>
  </navigation-case>
  <navigation-case>
    <description>
      Anmeldung fehlgeschlagen, weiter mit der Seite loginFalsch.jsp
    </description>
    <from-outcome>misserfolg</from-outcome>
    <to-view-id>/loginFalsch.jsp</to-view-id>
  </navigation-case>
</navigation-rule>
```

- `<navigation-rule>`

  Das `<navigation-rule>`-Element definiert die Regel für ein Navigations-Fall. Innerhalb der `faces-config.xml` können beliebig viele Navigations-Fälle definiert werden.

- `<from-view-id>`
Dieses Element kennzeichnet, auf welche Seite sich diese Navigations-Regel bezieht. In unserem Fall ist es die login.jsp-Seite. Wenn das Element nicht angegeben ist, so gelten die Navigations-Regeln global für die gesamte Applikation.

- `<navigation-case>`


### 3.11 Lokalisierung


Beim den ResourceBundles handelt es sich um zentrale Text-Dateien, die nach dem key=Value-Schema aufgebaut sind. Der Aufbau der Dateibezeichnung folgt dem Schema resources_xx.properties. Für die Variable xx stehen dann die verschiedenen Abkürzungen für die verschiedenen Sprachen, wie deutsch (de), englisch (en) usw. und für resources der Name des Bundles.

```properties
# resources_de.properties
userNameField=Benutzername
passwordField=Passwort
loginButton=Login
[...]
```

3.11. LOKALISIERUNG

FacesContext.getCurrentInstance().
   getViewRoot().setLocale(Locale.GERMAN)

Die Sprach-Einstellungen können natürlich auch entsprechend geändert werden. Diese Anweisung würde die Sprach-Einstellungen auf deutsch setzen.

<f:loadBundle basename="de.pixxi.resources" var="messages"

Das Laden des ResourceBundle geschieht über das basename-Attribut und bezieht sich auf die Dateien resources des ResourceBundles, die sich im Verzeichnis de.pixxi des Klassenpfads befinden müssen. Das var-Attribut definiert einen Namensraum innerhalb der JSP-Seite, mit dem dann der Zugriff möglich ist.

<h:outputText value="#{messages.userNameField}"/>

Der Ausdruck #{messages.userNameField} referenziert über den Mechanismus Value Binding Expression den String-Wert, der unter dem Schlüssel-Wort userNameField zu finden ist. Im unseren Beispiel würde die UI-Komponente outputText eine Beschriftung mit dem String „Benutzername“ bekommen.
4 Das Equipment Management System


Entwickelt worden ist das „Equipment Management System (EMS)“ von der C1 WPS GmbH in Zusammenarbeit mit der Uni Hamburg. In diesem Projekt wurde gezielt auf die Zusammenarbeit zwischen Studenten und erfahrenden Entwicklern geachtet.

4.1 Einführung

Dieser Teil der Studienarbeit ist im Rahmen des Projektes „Objektorientierte Softwareentwicklung“ am Fachbereich Softwaretechnik der Universität Hamburg entstanden. Lernziele dieses 2 Semester dauernden Projektes waren neben der Einarbeitung in fundamentale Methoden und Techniken der objektorientierten Konstruktion, auch die Einarbeitung, Erweiterung und das Refactoring einer schon bestehenden Anwendung, wie das „Equipment Management System“.

Der Schwerpunkt meiner Person an diesem Projekt war, die Möglichkeit einzuschätzen, inwieweit sich die Client-Server Architektur des EMS eignet, die Steuerung auch über andere Benutzeroberflächen bzw. Präsentationsarten abzuwickeln. Bisher war dies nur möglich mittels der GUI-Swing Benutzeroberfläche. Damit war die komplette Geschäfts- und Präsentationslogik im Client vereint.

Im Rahmen dieser Studienarbeit wurde deshalb von mir ein browserbasierter Client entwickelt. Im Hinblick auf die Entwicklung des Clients, wird uns das WEB-Framework -

\( ^1 \text{(vgl. Zülb05)} \)

\( ^2 \text{C1 WPS Workplace Solutions Gesellschaft für DV-Beratung mbH} \)
JaverServer Faces die Architektur liefern. Die Client-seitigen Zugriffe erfolgen dann durch einen Webbrowser.

Im Folgenden soll nun das EMS vorgestellt werden. Hierbei wird besonders auf die verschiedenen Szenarien des EMS eingegangen. Sie sollen ein Gefühl für die Funktionsweise und das Verhalten des EMS vermitteln. Es folgt dann eine Beschreibung der Architektur des EMS. Zu berücksichtigen ist, das im Rahmen dieser Studienarbeit es mir nicht möglich war, alle Aspekte des Systems komplett zu beschreiben. Viele Beschreibungen werden sich vielmehr nur auf die Teile stützen, die von mir bearbeitet worden sind. Abgeschlossen wird dieses Kapitel, am Beispiel des employee editor-Tool, das zeigt, wie meine Realisierung und Umsetzung eines Tools des EMS aussieht.

4.2 EMS - Szenarien

Die Szenarien beschreiben die Aufgaben, Eigenschaften und das Verhalten des Equipment Management System. Dabei soll kurz auf die Funktionen der einzelnen Komponenten eingegangen werden, sowie eine kurze Erklärung zu den einzelnen Tools.

4.2.1 Die zentralen Aufgaben des Equipment Management System

Bevor ein Benutzer das Equipment Management System benutzen kann, wird er gebeten, seinen Benutzernamen und Passwort einzugeben. Bei erfolgreichen Login kann der „equipment Manager“ (Benutzer) das System nutzen. Er hat nun die Möglichkeit, die Geräte (Rechner), Mitarbeiter und Räume zu Verwalten bzw. zu organisieren.

Dem Benutzer präsentiert sich eine Benutzeroberfläche (Workbench), die in verschiedene Segmente unterteilt ist. Die unterschiedlichen Segmente sind der Local Desktop, die Registry, die Toolbox, der Shredder und verschiedene Buttons zur Steuerung. Zu dem kommt noch eine Menü-Leiste, die viele Aktionen bietet wie z.B. das Ändern des Benutzers oder die des Schrifttyps. Die Abbildung 4.1 zeigt eine Ansicht auf die Benutzeroberfläche des EMS.

- **Local Desktop**

  Auf der linken Seite der Benutzeroberfläche, innerhalb des Local Desktop (material box) werden alle „lokalen“ Materialien angezeigt. Das sind jene Materialien, die sich mit den verschiedenen Tools der Toolbox bearbeiten lassen.
• **Registry**


• **Toolbox**

In der *Toolbox* werden alle Tools angezeigt, mit denen der Benutzer die Materialien bearbeiten kann. Dem Benutzer stehen folgende Tools zur Verfügung: Kopier-Materialen (copying materials), Editiere-Geräte (Editing a device), Editiere-Raumpläne (Editing room maps), Erzeuge-Raumpläne (RoomMap Designer), Editiere-Mitarbeiter (Editing an employee) und Erzeuge-Werte (Value Definer).
4.2. EMS - Szenarien

- Shredder


Das erzeugen von neuen Materialien und ändern dieser, ist eine der zentralen Aufgaben, die die EMS-Tools unterstützten. Auf die Funktionsweise der Tools soll im folgenden noch genauer eingegangen werden.

4.2.2 Editiere-Geräte (Editing a device)


4.2.3 Editiere-Raumpläne (Editing room maps)

Der Benutzer hat mit dem room map designer-Tool die Möglichkeit room maps anzulegen und zu ändern. Über die Option „Edit Room Map Properties“ kann der Benutzer Einfluss nehmen auf die Größe sowie auf den Namen der room map. Innerhalb einer room map können nun neue Räume (rooms) mit der Option New Room angelegt werden. Die Räume können dann entsprechend mit Geräten (devices) und Mitarbeitern (Members) belegt werden.

4.2.4 Editiere-Mitarbeiter (Editing an employee)

Ein Mitarbeiter wird über das employee editor-Tool geöffnet. Wie die vorherigen Tools auch, dient dieses Tool der Erzeugung und Änderung. Das employee editor-Tool kann nur von Materialien vom Typ employee verarbeiten. Das Tool definiert Eigenschaften eines Mitarbeiters wie Vorname, Nachname, Anschrift, E-mail, Telefonnummer, Tag der Einstellung und Position (Manager, Team Member). Nach Beendigung der Änderungen
KAPITEL 4. DAS EQUIPMENT MANAGEMENT SYSTEM

kann der Benutzer durch drücken des save-Button die Angaben abspeichern. Ist der angelegte Mitarbeiter noch nicht vorhanden, wird er dem Local Desktop hinzugefügt.

4.2.5 Weitere Tools

Es gibt noch eine Reihe weiterer Tools, die im Rahmen dieser Arbeit nicht weiter beschrieben werden können, aber der Vollständigkeit halber, noch kurz erwähnt werden sollen. Zu dem zählen die Tools: Copying materials vervielfältigt ein ausgewähltes Material, Value Definer erstellt eindeutige Werte für die Gerätebezeichnungen, Device Organizer können die Geräte auf die verschiedenen Räume verteilt werden.

4.3 Die Architektur des EMS

Hauptkomponente und zentrale Anlaufstelle des EMS ist das Workbench-Tool. Sie beinhaltet den Local Desktop (MaterialBox), die Registry (RemoteMaterialBox), die ToolBox und den Shredder. Innerhalb der ToolBox befinden sich alle angemeldeten und registrierten Tools.

Abbildung 4.2: Basis-Struktur der Benutzer-Interface-Komponenten des EMS
4.3. DIE ARCHITEKTUR DES EMS

Die Abbildung 4.2 zeigt die Basis-Struktur der Benutzer Interface Komponenten des Equipment Management System, nachdem das System abgemeldet und initialisiert worden ist.\(^3\) Zu beachten ist, dass die Abbildung keine Materialien enthält. Die Pfeile innerhalb der Abbildung zeigen eine contains relationships. Die meisten Elemente in dieser Abbildung sind Tools, folglich wurden die Tools nach ihrer korrespondierenden Klasse benannt. Die RegistryServices-Komponente beschreibt die sechs verschiedenen services, die von der RemoteMaterialBox genutzt werden: RegistryService, RegistryServiceFile, RegistryServiceJDO, RegistryServiceListener, RegistryServiceProxy and RegistryServiceServer. Diese service-Architektur hat den Vorteil, dass die Speicherung der Materialien auf der Server-Seite unabhängig und verborgen vom Client durchgeführt werden können. Somit kann die Art der Speicherung (Flat-file bzw. JDO) flexibel ausgetauscht werden, ohne am Client große Änderungen vornehmen zu müssen.

Die Abbildung 4.3 zeigt die Abhängigkeiten von Tools und Materialen.\(^4\) Sie lässt bewusst die service-Architektur außen vor. Die Pfeile beschreiben eine „works on“ relationship.

Abbildung 4.3: Die „works on“ relationship von Tools und Materialen

\(^3\) (vgl. wps07a)  
\(^4\) (vgl. wps07a)
4.4 Das Employee-Tool

Dieser Abschnitt beschreibt die Umsetzung und Realisierung des Employee-Tool. Bevor das Employee-Tool benutzt werden kann, muss ein Systemzustand geschaffen werden, indem das EMS arbeitsfähig ist. Arbeitsfähig ist das EMS und damit bereit für die Benutzeraktionen, wenn alle Initialisierungen abgeschlossen sind und die Workbench auf Methodenaufrufe reagieren kann. Um diesen Systemzustand zu schaffen, muss sich der Benutzer vorerst einmal anmelden.

4.4.1 Ausgangssituation


Die Abbildung 4.4 zeigt den Verlauf der Anmeldung eines Benutzers, bis hin zu dem Punkt, an dem die Workbench des EMS einsatzbereit ist, um Methoden-Aufrufe vom der WorkbenchBean bzw. des EMSControllers entgegen nehmen zu können.

1. Der HTML-Client sendet ein Request an die Login.jsp. Auf dieser Seite kann der Benutzer seinen Benutzernamen und das Passwort eintragen.


5. Jeder \textit{WebUser} bekommt ein \textit{Controller}-Objekt, das später als Vermittler zwischen der \textit{WorkbenchBean} und der \textit{Workbench} des EMS fungieren soll. Hierzu verwaltet er die Referenz der \textit{Workbench} des EMS.

6. Der \textit{EMSController} startet den \textit{EMSClient} in einem eigenen Thread. Dieser \textit{EMSClient} startet unabhängig seine Initialisierung und stellt die Verbindung mit dem \textit{EMSServer} her.

7. Im Verlauf der Initialisierung des \textit{EMSClient} ist die \textit{Workbench} soweit fortgeschritten, dass sie sich bei der zentralen \textit{EMSAdmin} registrieren kann.

8. Die \textit{EMSAdmin} übergibt die Referenz der \textit{Workbench} an das entsprechende \textit{EMS-
Controller-Objekt. Dieses verwaltet fortan diese Referenz, um spätere Methoden-Aufrufe von der WorkbenchBean weiter leiten zu können.


10. Der Navigationsfall verweist bei einem erfolgreichen „Login“ auf die Workbench.jsp-Seite.


13. Das EMSController-Objekt leitet dann wiederum die Verarbeitung an die Methoden der Workbench weiter, die dann letztendlich die eigentliche Funktionalität enthalten.

4.4.2 Umsetzung und Realisierung

Nachdem wir die Ausgangslage geschaffen haben, kann der Benutzer das EMS über die UI-Komponenten der Workbench.jsp-Seite „steuern“. In diesem Abschnitt wird die Umsetzung und Realisierung des Employee-Tool beschrieben. Ausgegangen wird von folgender Situation: Ein Employee (initialen RT) wurde aus der Registry (remote material box) „ausgecheckt“ und befindet sich im Local Desktop (material box). Die Abbildung 4.5 zeigt die Ausgangslage.

Die Bearbeitung des Employee wird gestartet, durch das Drücken des „CommandLink-Edit“.

```
<h:commandLink id="Edit"
    action="edit"
    actionListener="#{editBean.edit}"
>
    <h:outputText value="Edit" />
    <f:param id="editThingDescriptionDV"
        name="thingDescriptionDV"
        value="#{localObj.thingDescriptionDV}" />
</h:commandLink>
```
4.4. DAS EMPLOYEE-TOOL

Abbildung 4.5: Der Employee (RT) ist bereit zur Bearbeitung.

Der „CommandLink-Edit“ ist verbunden mit der Action-Methode edit innerhalb der editBean.

```java
public void edit(ActionEvent event){
    UIParameter component = (UIParameter)
    event.getComponent().findComponent("editThingDescriptionDV");
    ThingDescriptionDV thingDescriptionDV = (ThingDescriptionDV)
    component.getValue();
    employee = emsController.getEmployeeForEdit(thingDescriptionDV);
    this.setEditBean(employee);
}
```

Damit wir das Material Employee überhaupt bearbeiten können, müssen wir zunächst einmal das Material aus der MaterialBox (Lokal) auslesen. Ziel ist es, die „EditBean“ mit den Attributen eines Employee zu füllen. Die EditBean definiert genau die Attribute, die ein Employee besitzen kann. Die EditBean ist somit das Model-Objekt für die emsEdit.jsp-Seite, die die View-Komponente darstellt.
UIParameter component = (UIParameter)
    event.getComponent().findComponent("editThingDescriptionDV");

Als erstes wird die Komponente identifiziert, über die der Benutzer das Event ausgelöst hat, mittels des Parameters „editThingDescriptionDV“. Die UI-Komponente ist über das Tag <f:param id="editThingDescriptionDV" identifizierbar.

ThingDescriptionDV thingDescriptionDV = (ThingDescriptionDV)
    component.getValue();

Der Aufruf component.getValue() bezieht sich demnach auf den Wert der Komponen-
te, der unter dem Tag <f:param value angegeben ist. Das value-Attribut verweist auf die Lokalen-Objekte und dessen Attribut .thingDescriptionDV.

disController.getEmployeeForEdit(thingDescriptionDV);

Jedes Material (Employee) hat eine Beschreibung vom Typ ThingDescriptionDV. Mit dieser ThingDescriptionDV ist es uns möglich, den Employee aus der MaterialBox der Workbench auszulesen.

this.setEditBean(employee);

Der übergebene „employee“ wird mittels der Methode this.setEditBean(employee) dem Model-Objekt EditBean zugewiesen. Nachdem das Model-Objekt mit den Attributen des Employee initialisiert worden ist, kann die JSP-Seite emsEdit.jsp zur Bearbeitung aufge-
rufen werden. Die Abbildung 4.6 zeigt die emsEdit.jsp-Seite.

Der Benutzer hat nun die Möglichkeit den Employee zu bearbeiten. Dabei wird die Eingabe, wie z.B. die des „HiringDate“ wie im Beispiel zu sehen, schon auf seine „Rich-
4.4. DAS EMPLOYEE-TOOL

Abbildung 4.6: Die emsEdit.jsp

public class MonthValidator implements Validator {

    public void validate( FacesContext context,
        UIComponent component,
        Object value)
        throws ValidatorException {

        int intValue = (Integer) value;

        if ( (intValue < 1) | (intValue > 12) ) {
            FacesMessage message = new FacesMessage("Die Benutzereingabe des" +
                "von Ihnen gewählten Monats ist falsch! Bitte beachten Sie, " +
                "dass der von Ihnen gewählte Wert im Intervall von 1 - 12 bleibt " +
                "und das ganze Datum im Format dd.mm.yyyy gehalten wird."
            );
            throw new ValidatorException(message);
        } else {
            //ok
Nachdem der Benutzer seine Eingaben vervollständigt hat, kann er die Änderungen „verwerfen“ oder „speichern“. Durch drücken des save-Button werden die aktuellen Properties von der View in den Employee gespeichert und der Employee wird danach wieder in die lokale Registry „eingechockt“.

In diesem Beispiel wurde die Umsetzung und Realisierung eines Tools des EMS auszugsweise verdeutlicht. Im Rahmen dieser Studienarbeit war es leider nicht möglich, alle beteiligten Komponenten (Klassen und Methoden) komplett aufzuzeigen. Beteiligte Komponenten, wie z.B. die Konfigurationsdateien oder Validatoren, wurden aber zumindest schon in vorangegangenen Abschnitten erläutert. Es bleibt festzuhalten, dass es möglich ist, mit dem HTML-Client das EMS zu steuern.
5 Zusammenfassung, Bewertung und Ausblick


weiter zu verfeinern oder es mit anderen Auszeichnungssprachen wie z.B. WML zu implementieren.
Sämtlicher im Rahmen dieser Studienarbeit entstandener Quellcode ist auf der beigelegten CD vorhanden. Der Quellcode für das EMS wurde der Workbench der Eclipse-Programmierumgebung entnommen.

- **EMS-Projekt**
  Das EMS-Projekt befindet sich im Verzeichnis: `workspace/XEMSweb/source/de/itwps/ems`

- **Web-Komponenten**
  Die Web-Komponenten (EMSAdmin, EMSController, WEBUser) befindet sich im Verzeichnis: `workspace/XEMSweb/source/de/itwps/ems/web`

- **Beans**
  Die Beans befindet sich im Verzeichnis: `workspace/XEMSweb/source/de/itwps/ems/web/bean`

- **Converter**
  Die Converter befindet sich im Verzeichnis: `workspace/XEMSweb/source/de/itwps/ems/web/converter`

- **Validator**
  Die Validatoren befindet sich im Verzeichnis: `workspace/XEMSweb/source/de/itwps/ems/web/validator`

- **Die JSP-Seiten**
  Die JSP-Seiten befindet sich im Verzeichnis: `workspace/XEMSweb/WebRoot`

- **faces-config.xml + web.xml**
  Diese Dateien befinden sich im Verzeichnis: `workspace/XEMSweb/WebRoot/WEB-INF`

B  Literaturverzeichnis


[Hal04] HALL, Larry Brown; M.: *Core Servlets and Java Server Pages*. Sun Microsystems, 2004


[Ull06] ULLENBOOM, Christian: Java ist auch eine Insel, Programmieren für die Java 2-Plattform in der Version 5. Galileo Computing, 2006


[Zül05] ZÜLLIGHOVEN, Heinz: *Object-Oriented Construction Handbook - Developing Application-Oriented Software with the Tools and Materials Approach*. dpunkt Verlag, 2005

C Ehrenwörtliche Erklärung


Hamburg, 27. März 2007

 Ort, Datum Unterschrift