Technische Universität Ilmenau
Fakultät für Informatik und Automatisierung

Studienarbeit

Erstellung intuitiver Web Frontends zur Terminplanung und Verwaltung administrativer Daten, basierend auf einem Webserver mit JSP Technologie, sowie Anbindung an ein medizinisches Informationssystem

Rolf Holzmüller

Betreuer:	Dipl.-Ing. Christian Heller
verantwortlicher Hochschullehrer:	Prof. Dr.-Ing. habil. Ilka Philippow

Ilmenau, den 10.April 2003

Inhaltsverzeichnis

1 Einführung

1.1 Ziel

Im Rahmen der vorliegenden Studienarbeit soll untersucht werden, wie intuitive Frontends über einen Webserver mit Hilfe der JSP-Technologie realisiert werden können. Der Aufgabenbereich ist dabei die Verwaltung administrativer Daten und die Terminplanung mit Anbindung an ein medizinisches Informationssystem. Dafür soll eine Referenzimplementierung umgesetzt werden.

Zum besseren Verständnis wird in den ersten Kapiteln auf allgemeine Technologien für Webanwendungen eingegangen. Dies umfaßt eine allgemeine Web-Architektur sowie die verwendeten Technologien (Servlets, JSP, JDBC, Java Beans, Tags).

Des weiteren ist auf ein flexibles Design und die Erweiterbarkeit der Anwendung Wert gelegt worden. Dafür ist es sinnvoll, auf Erfahrungen von anderen zurück zu greifen. Diese Erfahrungen sind in Entwurfsmustern enthalten. Ein gängiges Entwurfsmuster ist das Model-View-Controller Muster (MVC-Muster). Dabei wird eine Trennung von Inhalt, Darstellung und Interaktion gemacht. Diese Trennung ist wichtig, um schnell Änderungen an einzelnen Komponenten, wie z.B. der Ansicht, zu realisieren.

Um die hier beschriebenen Verfahren und Technologien zu veranschaulichen, gehört zu dieser Arbeit eine Referenzimplementierung einer Terminvergabe für eine Arztpraxis. Diese wurde nach dem Model-View-Controller Muster und den hier vorgestellten Technologien realisiert.

1.2 Umfeld

Die Studienarbeit wird im Rahmen des Projektes Res Medicinae realisiert. Dieses Open Source Projekt verfolgt das Ziel, medizinische Anwendung als freie Software zur Verfügung zu stellen. Sie soll in Debian als ein Paket integriert werden. Dazu ist es notwendig, dass die erstellten Programme und Dokumentationen unter die GNU-Lizenz gestellt sind.

2 Web-Architektur

2.1 Überblick

In diesem Abschnitt wird eine einfache Hardwarearchitektur für die Webanwendung beschrieben. Die Web-Architektur gehört zu den Client-Server-Architekturen. Da für eine Webanwendung verschiedene Komponenten zusammenarbeiten, werden diese Komponenten hier vorgestellt und das Zusammenwirken der Komponenten beschrieben. Da es eine Vielzahl von Möglichkeiten und Kombinationen der Komponenten gibt, kann dies hier nicht umfassend dargestellt, sondern nur angerissen werden.

2.2 Client-Server-Architektur

Hier ist die Definition aus dem Buch [1]

> „Unter der Client-Server-Architektur (engl.: client-server architecture) versteht man eine kooperative Informationsverarbeitung, bei der die Aufgaben zwischen Programmen auf verbundenen Rechnern aufgeteilt werden. In einem solchen Verbundsystem können Rechner aller Art zusammenarbeiten. Server (= Dienstleister; Backend) bieten über das Netz Dienstleistungen an, Clients (= Kunden; Frontend) fordern diese bei Bedarf an. Die Kommunikation zwischen einem Client-Programm und dem Server-Programm basiert auf Transaktionen, die vom Client generiert und dem Server zur Verarbeitung überstellt werden. Eine Transaktion (engl.: transaction) ist eine Folge logisch zusammengehöriger Aktionen, beispielsweise zur Verarbeitung eines Geschäftsvorfalls. Client und Server können über ein lokales Netz verbunden sein oder sie können über große Entfernungen hinweg, zum Beispiel über eine Satellitenverbindung, miteinander kommunizieren. Dabei kann es sich um Systeme jeglicher Größenordnung handeln; das Leistungsvermögen des Clients kann das des Servers also durchaus übersteigen. Grundidee der Client-Server-Architektur ist eine optimale Ausnutzung der Ressourcen der beteiligten Systeme.“

2.3 Verwendetet Komponenten

Ein Server ist ein Prozess, Programm oder Computer zur Bearbeitung der Anforderungen eines Clients bzw. zur Bereitstellung von Diensten, die von einem Client genutzt werden können.

Um die typische Webanwendung umzusetzen, bedarf es folgender Komponenten:

1. Webserver

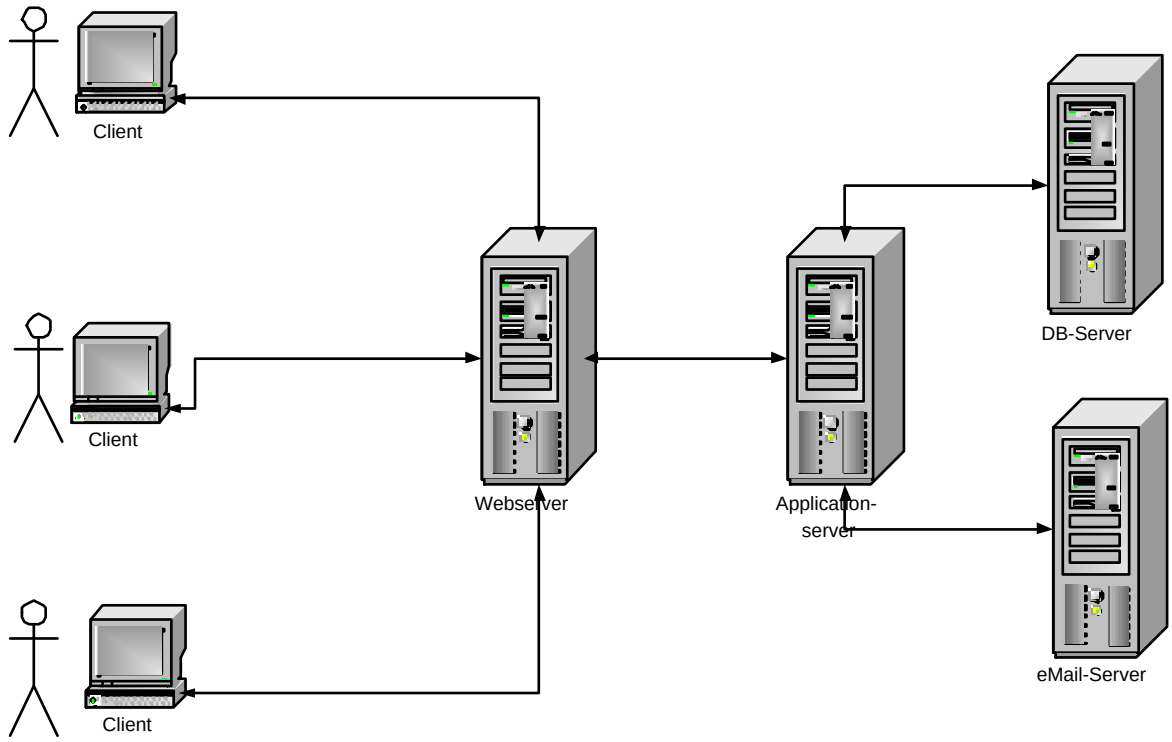
2. Applicationserver
3. Datenbankserver
4. eMail-Server
5. Browser

Tabelle 1: Tabelle der verwendeten Komponenten

Komponente	**Beschreibung**
Webserver	Ein Server, der auf Anforderung Web-Seiten zu einem HTML-Browser mittels dem Protokoll HTTP überträgt.
Applicationserver	Ermöglicht die Ausführung komplexerer Aufgaben an einem entfernten Rechner oder über das Internet. Auf diese Weise wird die Kommunikation zwischen Server und Benutzer einer Webseite verbessert, so dass dieser z. B. eine Datenbank abfragen oder Programme ausführen kann, die auf dem Server installiert sind. Application-Server bieten häufig zusätzlich Sicherheitsmerkmale, Lastenverteilung (load balancing) und Ausfallmechanismen (failover mechanism) sowie Skalierungs- und Integrationsfunktionen.
Datenbankserver:	Die Aufgabe des Servers ist die Verwaltung und Organisation der Daten, die schnelle Suche, das Einfügen und das Sortieren von Datensätzen. Auf diesem Server läuft eine Datenbank, die diese Aufgaben übernimmt.
eMail-Server:	Dieser Server stellt Dienste für das Verschicken und Empfangen von eMails bereit.
Browser:	Ein Browser ist in erster Linie ein Anwendungsprogramm zur Anzeige von HTML-Seiten. Mit zunehmender Verbreitung der Internet-Technologien entwickelt sich der Browser zum Universal-Client und dient als Schnittstelle für sämtliche Informationen und Anwendungen, die auf Internet-Technologien basieren.

2.4 Zusammenspiel der Komponenten

Dies ist nur eine Beispielarchitektur. In dem Bild wird für jeden Server ein eigener Rechner verwendet. Dies wird in den meisten Fällen nicht so sein. Es können auch mehrere Server (z.B. Webserver und Applicationserver) auf einem Rechner betrieben werden.


Client
Client
Client
Webserver
Application-
server
DB-Server
eMail-Server

3 JDBC

3.1 Überblick

JDBC steht für Java Database Connectivity und ist die Datenbankschnittstelle für Java. Ab JDK 1.1 bietet Java durch JDBC eine vollständige Datenbankunterstützung zum Zugriff auf relationale Datenbanken an. Dazu stellt JDBC folgende Funktionalität zur Verfügung:[2]

- Objekte für die Verbindung mit Datenbanken
- Ausführen von SQL-Anweisungen mit gespeicherten Prozeduren
- Methoden zur Ermittlung von Informationen über Objekte in der Datenbank

Mittels JDBC wird ein Datenbankzugriff von Java-Anwendungen über verschiedene Treiber möglich. Es gibt sowohl spezielle Treiber für einen native-Zugriff auf Datenbanken, als auch allgemeine Treiber für den Datenbankzugriff über ODBC (siehe Treibertypen). Durch die JDBC-Spezifikation wird eine maximale Flexibilität sowohl für Datenbankbenutzer als auch für Datenbankanbieter gewährleistet.

Um eine Austauschbarkeit der Datenbanken halbwegs zu gewährleisten, fordert SUN von den JDBC-Treiberherstellern, mindestens den SQL-2 Entry-Level-Standard von 1992 zu erfüllen. Verwendet der Programmierer darüber hinaus proprietäre SQL-Anweisungen der Datenbanken, so ist eine Austauschbarkeit nur mit erheblichem Mehraufwand zu erreichen.

3.2 Treibertypen

JDBC ist keine eigene Datenbank, sondern eine Schnittstelle zwischen einer SQL-Datenbank und der Applikation, die sie benutzen will. Bezüglich der Architektur der zugehörigen Verbindungs-, Anweisungs- und Ergebnisklassen unterscheidet man vier Typen von JDBC-Treibern [8]:

- Steht bereits ein ODBC-Treiber zur Verfügung, so kann er mit Hilfe der im Lieferumfang enthaltenen JDBC-ODBC-Bridge in Java-Programmen verwendet werden. Diese Konstruktion bezeichnet man als Typ-1-Treiber. Mit seiner Hilfe können alle Datenquellen, für die ein ODBC-Treiber existiert, in Java-Programmen genutzt werden.
- Zu vielen Datenbanken gibt es neben ODBC-Treibern auch spezielle Treiber des jeweiligen Datenbankherstellers. Setzt ein JDBC-Treiber auf einem solchen proprietären Treiber auf, bezeichnet man ihn als Typ-2-Treiber.

- Wenn ein JDBC-Treiber komplett in Java geschrieben und auf dem Client keine spezielle Installation erforderlich ist, der Treiber zur Kommunikation mit einer Datenbank aber auf eine funktionierende Middleware angewiesen ist, handelt es sich um einen Typ-3-Treiber.
- Falls ein JDBC-Treiber komplett in Java geschrieben ist und die JDBC-Calls direkt in das erforderliche Protokoll der jeweiligen Datenbank umsetzt, handelt es sich um einen Typ-4-Treiber.

4 Servlets

4.1 Überblick

Eine normale Anfrage über das Web folgt immer dem gleichen Schema. Ein Benutzer stellt eine Anforderung (request) an einen Server. Dieser Server antwort (response) auf diese Anforderung und schickt das gewünschte Ergebnis an den Benutzer zurück. Der Browser stellt dann diese Antwort dar. An dieser Stelle wird zwischen zwei verschiedenen Verarbeitungsformen unterschieden.

- Clientseitige Verarbeitung
  Die Antwort des Servers wird erst auf dem Client verarbeitet. Ein Beispiel für diese Techniken wären Javascripts und Applets.
- Serverseitige Verarbeitung
  Die Anfrage des Benutzer wird auf dem Server verarbeitet. Der Benutzer bekommt die generierte Antwort zurück und muß diese nur noch darstellen. Ein Beispiel für diese Technik wäre PHP, ASP und Servlets.

Ein Servlet ist eine in Java geschriebene Softwarekomponente in Java geschrieben. Die Firma Sun entwickelte diese Anwendungsschnittstelle (API), um eine einfache Möglichkeit der serverseitigen Programmausführung bereitzustellen. Über diese Api ist es möglich, auf Objekte der Webkommunikation (request, response) zuzugreifen. Zu Ausführung eines Servlets wird eine Servlet-Engine gebraucht.

Servlets unterstützen Session und Cookies. Sessions dienen dazu, Sitzungen eines Benutzers zu verfolgen und eventuell Informationen zu der Sitzung zu speichern. Cookies sind kleine Informationshappen, die auf der Clientseite gespeichert werden können.

4.2 Arbeitsweise

Wird eine Anfrage an das Servlet gestellt, so wird diese durch die Servlet-Engine (falls noch nicht geladen) mit init(ServletConfig) initialisiert. Nach der Initialisierung können mehrere Anfragen an das Servlet gestellt werden, ohne dass es dabei neu initialisiert wird. Das bedeutet, dass ein Servlet mehrere Anfragen und damit mehrere Benutzer bedienen kann, obwohl es nur einmal im Speicher geladen wird.

Bekommt ein Servlet eine Anfrage, so geschieht das mit service(ServletRequest, ServletResponse). Das Servlet liest aus dem Request-Object die Parameter und schreibt in das Response-Object die Ausgabe an den Client. Dazwischen kann es jede Anwendungslogik ausführen, die mit Java realisierbar ist. Davon bekommt der Benutzer (Client) nichts mit, weil zum Client nur das Ergebnis (z. B. HTML) geschickt wird.

Wird das Servlet nicht mehr gebraucht, so wird es mit destroy() freigegeben.

5 JavaBeans

5.1 Überblick

Spricht man von Java Beans, so ist die komponentenbasierte Anwendungsentwicklung in der Sprache von SUN Microsystems gemeint. Ausgehend von der von JavaSoft vorgestellten Programmiersprache „JAVA“ wurde die Objektorientierung zu einer Komponentenorientierung weiter entwickelt.

Grundidee des Übergangs zu Komponenten ist der verstärkte Wunsch nach einer Wiederverwendung bestimmter, allgemeingültiger Programmteile. So schreibt M. Morrieson: „Eine Komponente ist ein wiederverwendbares Stück Software, das leicht mit anderen zusammengebaut werden kann, um auf diese Weise Anwendungen viel effizienter zu erstellen.“ [3] Ziel der Wiederverwendung ist also eine schnellere kostengünstigere Softwareentwicklung.

5.2 Beschreibung

Beans wurden vor allem für die visuelle Softwareentwicklung entwickelt. Die wichtigsten Konzepte von JavaBeans sind Ereignisse (events) und Eigenschaften (properties). Diese werden hier im folgenden kurz beschrieben.

- Ereignisse
  JavaBeans definiert ein Standardmodell zur Kommunikation von Komponenten über einen Ereignismechanismus. Der Ereignismechanismus erlaubt die weitgehende Entkopplung von Sender- und Empfängerkomponenten. Die Sender definieren eine Ereignisschnittstelle, welche die Anmeldung von Empfängern für ein Ereignis erlaubt. Empfänger definieren Methodenschnittstellen, die bei Auftreten der Ereignisse angesprochen werden.

- Eigenschaften
  Eigenschaften sind benannte Attribute eines Beans, die das Erscheinungsbild und/oder das Verhalten beeinflussen können. Sie sind Teil des Zustands von Beans.

Um auf die JavaBeans zugreifen zu können, müssen Informationen zu diesen ermittelbar sein. Dies geschieht durch ein Introspektionsmechanismus. Das kann bei Einhaltung der Namenskonvention durch den Reflectionmechanismus von Java oder bei Nichteinhaltung über die BeanInfo-Schnittstelle geschehen.

Weitergehende Informationen zu JavaBeans sind z. B. unter [4] und [5] nachzulesen.

5.3 Konventionen

JavaBeans folgen im allgemeinen einem speziellen Design- und Namensmuster. Für Java Server Pages sind vor allem die Eigenschaften von JavaBeans von Bedeutung. Für die Verwendung von JavaBeans in den Java Server Pages sind folgende Konventionen einzuhalten:

- müssen im Package sein
- Default-Konstruktor (Konstruktor ohne Parameter)
- Definieren einer Menge von Properties
- für den lesenden und schreibenden Zugriff auf die Properties werden get- und set-Methoden definiert

Werden die get- bzw. set-Methoden weggelassen, so hat man entweder nur lesend oder nur schreibend auf die Eigenschaften (Properties) Zugriff.

Weitere Informationen zu Konventionen zu JavaBeans entnehmen Sie bitte der Spezifikation von Sun. Diese ist unter http://java.sun.com/beans/spec.html erreichbar.

6 Java Server Pages (JSP)

6.1 Überblick

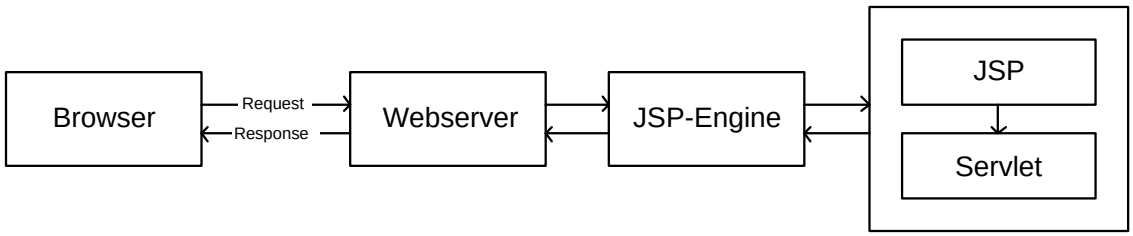
Java Server Pages basieren auf der Servlettechnologie von Sun. In diesem Kapitel wird die JSP-Technologie kurz vorgestellt. Es werden aber keine Vergleiche zu anderen Techniken behandelt. Es wird hier auch keine Sprachbeschreibung für JSP geben. An dieser Stelle wird nur das prinzipielle Vorgehen der JSP-Technologie in Hinblick auf die Trennung der Ansicht und der Applikation erklärt. Dazu gehören die Techniken JavaBeans und Tags.

6.2 Beschreibung

JSP sind eine verbreitete Möglichkeit, Webseiten dynamisch zu gestalten. Fokussiert wurde dabei ein möglichst einfaches Zusammenspiel zwischen HTML-Code und des leistungsstarken Java API. Zusammen mit der erleichterten Einbindung von JavaBeans lassen sich mit erheblich verringertem Aufwand selbst komplexe Webapplikationen erstellen. Syntaktisch gesehen handelt es sich bei einer JSP-Seite um ein HTML-Dokument mit eingebetteten Anweisungen einer Programmiersprache und zusätzlichen Tags.

6.3 Typischer Ablauf einer JSP-Anfrage

JSP-Seiten sind eine Mischung aus HTML und eingebettetem Java-Code, die durch JSP-Tags eingefügt werden. Ruft ein Benutzer eine JSP-Seite auf, sendet der Webserver nicht wie bei einer HTML-Seite einfach den Quelltext an den Browser. Stattdessen reicht er den Aufruf an eine JSP-Engine weiter. Sie übersetzt das JSP mit einem JSP-Compiler in ein Java-Servlet. Die Übersetzung des JSP in ein Servlet findet nur statt, wenn das Servlet noch nicht existiert oder der JSP-Quelltext geändert wurde. Dieses Servlet wird bearbeitet, und das Ergebnis wird in den Tags der JSP-Seite hinzugefügt. Erst diese generierte Seite wird dem Benutzer auf seine Anfrage zurück geschickt.



Dies ist dem Buch [7] entnommen.

6.4 Java Beans

Um ein Bean in JSP nutzen zu können, muß es in der JSP-Seite definiert sein.

```
<jsp:useBean
  id="myModelDoctor"
  scope="session"
  class="org.resmedicinae.application.healthcare.resdata.model.ResDataDoctor"
/>
```

Nach der Definition können über das Bean die Methoden der Klasse genutzt werden. Auf die Beans kann auf folgende Weise zugegriffen werden:

- Normaler Zugriff auf die Beans z.B. mit

  ```
  <\% bean.getDoc_name(); \%>
  ```

- Zugriff auf die Beans mittels Tags, z.B.

  ```
  <jsp:getProperty name="myModelDoctor"  property="doc_name" />
  ```

Leider haben Beans auch Nachteile. Sie können nicht auf implizite Objekte wie 'response' oder 'session' zugreifen, und sie können keine direkten Ausgabe mittels 'out.println()' machen.

6.5 Tags

Eine wirkliche gelungene Sache zur Trennung von Ansicht und Verarbeitungslogik ist die Integration von TagLibs in JSP ab der Version 1.1. JSP's sind nun durch eigene Tags und Funktionen erweiterbar. Der in den ersten Versionen mögliche Zugriff auf JavaBeans bezog sich auf Properties. Mit den TagLibs lassen sich Elemente für Funktionen ebenfalls integrieren. Die Tags sind XML-konforme Erweiterungen der bisherigen Standard-Tags. Diese Syntax erleichtert die Zusammenarbeit zwischen HTML-Designer und Entwicklern. Es können leicht spezielle Vereinfachungen für die dynamischen Inhalte der Site zur Verfügung gestellt werden, ohne dass Hunderte von Zeilen Codes oder mehrere include-Anweisungen in die Seite einzubauen wären. Der Entwickler kann eine neue Eigenschaft oder einen Fehler korrigieren, und das Ergebnis wird sofort in allen Seiten aktiviert, ohne sie zu modifizieren.

Tags dienen also dazu, Anwendungslogik und Darstellung zu trennen. Sie unterliegen nicht der Beschränkung wie die JavaBeans. Tags können auf implizite Objekte zugreifen und können HTML-Text über out.println() ausgeben.

Ein Tag ist ein Stück Text in der Form

```
<Präfix:Tagname attr1="wert1" attr2="wert2" ... > </Präfix:Tagname>
bzw. die Kurzform
<Präfix:Tagname attr1="wert1" attr2="wert2" ... />
```

Dabei steht der Präfix für den Namensraum, in dem der Tagname Gültigkeit hat. Dies dient dazu, Namenskonflikte zu vermeiden, weil der Tagname für einen speziellen Namensraum definiert wird.

7 Model-View-Controller (MVC)

7.1 Überblick

Als erstes wird das allgemeine Modell beschrieben. Dabei wird nicht auf die Umsetzung des Modells für die Webapplication eingegangen. Es werden Vor- und Nachteile genannt. Für die Webapplication sind nicht alle Teile des MVC-Musters umgesetzt. Dies liegt an den Besonderheiten der Kommunikation innerhalb der Webanwendung. Darauf wird im Abschnitt 8 eingegangen.

7.2 Beschreibung

Das Model-View-Controller Muster, im folgenden kurz MVC-Muster, wurde ursprünglich für die Erstellung von Benutzeroberflächen mit Smalltalk-80 verwendet. Ziel des MVC-Musters ist die Zerlegung von interaktiven Applikationen in Teilsysteme mit abgegrenzter Funktionalität und hohem Grad der Wiederverwendung. Das MVC-Muster überträgt die Trennung von Inhalt, Darstellung und Interaktion auf GUI-basierte Systeme. Dabei entspricht der Controller der Eingabe und der Ablaufsteuerung, das Model der Verarbeitung und die View der Ausgabe. Die drei Komponenten werden dabei separat behandelt und implementiert. Dies macht nicht nur die Erstellung von Anwendungen mit mehreren synchronen Ansichten auf die gleichen Daten leichter, sondern auch die Behandlung von Fehlern, da solche leichter zu lokalisieren sind. Das MVC-Modell garantiert, dass Änderungen im Modell zu den entsprechenden Änderungen in den Views führen. Hier nun eine kurze Beschreibung der drei Komponenten Model, View und Controller.

7.2.1 Model

Das Model repräsentiert die Daten einer Anwendung. Neben der Verwaltung der Daten führt es auch alle Änderungen auf diese Daten aus. Das Model liefert auf Anfrage den Zustand der Daten und reagiert auf Befehle, diese zu ändern.

Zwischen Model und View gibt es eine Registrations- und Benachrichtigungs-Interaktion. Wenn Daten im Modell sich ändern, werden die Views benachrichtigt. Daraufhin ändert jede View ihre Ansicht und stellt die neuen Daten dar. Die Views kommunizieren nicht miteinander und das Modell weiß nicht, in welcher Form die Views die Daten darstellen. Dies ist auch als Beobachter-Muster bekannt.

7.2.2 View

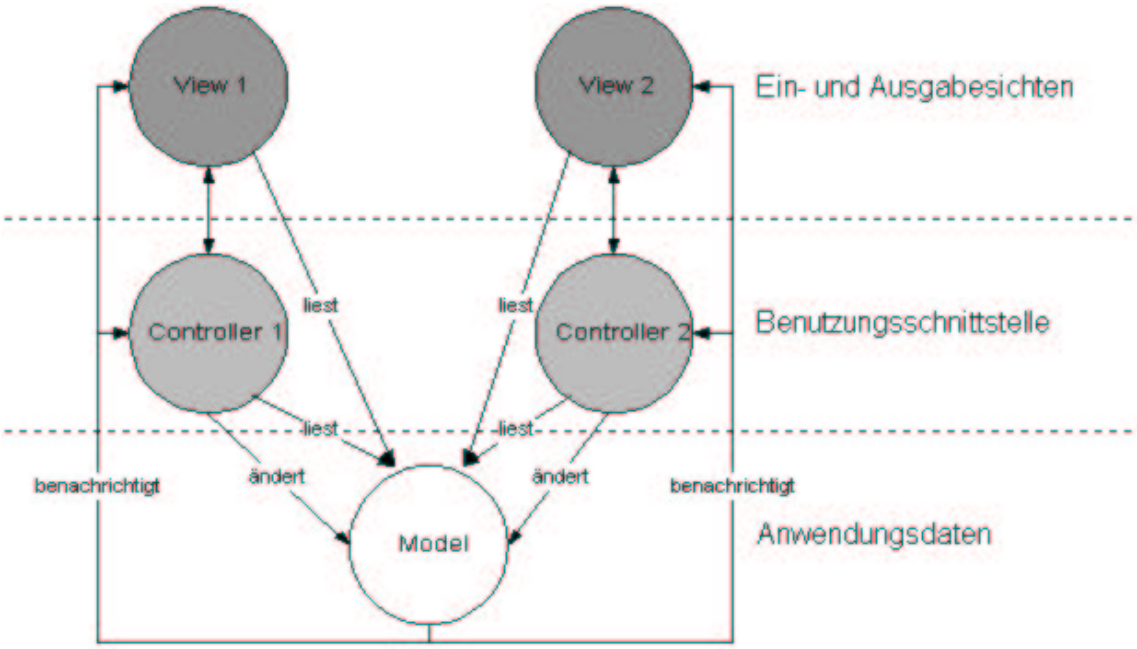
Das View-Objekt übernimmt die grafische Darstellung der Daten. Es stellt eine mögliche visuelle Abbildung des Models dar. Ein View-Objekt ist dabei immer mit genau einem Model verbunden. Für den Fall, dass sich das Model ändert, erstellt das View-Objekt automatisch eine neue passende Darstellung des Models und stellt diese grafisch dar. Das View-Objekt enthält einen Zeiger auf sein Model. Es besitzt also immer eine Referenz auf ein konkretes Model. Über diese Referenz kann es die Methoden des Models direkt aufrufen. Jedes View-Objekt muss die Möglichkeit zum Selbstupdate besitzen, damit es das Model immer korrekt darstellen kann. Normalerweise besitzt ein View-Objekt auch eine Referenz auf den Controller. Über diese sollte das Objekt aber immer nur die Basisschnittstelle des Controllers benutzen, da sonst der Austausch des Controllers nicht mehr so einfach durchzuführen ist.

7.2.3 Controller

Der Controller definiert, wie ein Benutzer mit der Applikation interagieren kann. Er nimmt Eingaben vom Benutzer entgegen und bildet diese auf Funktionen des Models oder der Views ab. Damit dies funktionieren kann, muss der Controller natürlich Zugriff sowohl auf das Model als auch auf die Views haben.

7.3 visuelle Darstellung des MVC-Muster

Hier sehen Sie die Verwendung der benutzten Komponenten.



7.4 Bewertung des MVC-Musters

Hier werden die wichtigsten Vor- und Nachteile des MVC-Musters kurz erläutert. Die Bewertung wurde dem Buch [6] entnommen.

Vorteile:

- Mehrere Ansichten desselben Modells: Das MVC-Muster trennt das Modell strikt von der Benutzungsschnittstelle. Das ermöglicht es, mehrere Ansichten derselben Daten in verschiedenen Formen zu erzeugen.
- Synchronisierte Ansichten: Das Änderungs-Konzept (mittels z. B. des Beobachter-Musters) stellt sicher, dass alle Ansichten von Änderungen an den Daten Kenntnis bekommen.
- Austauschbarkeit der Ansichten: Eine Portierung des Systems betrifft nur die Ansichten (Views) und Bediener (Controller), nicht jedoch die Kernfunktionalität.
- Framework-fähig: Es ist möglich, komplette Anwendungs-Frameworks auf dieses Muster zu stützen.

Nachteile

- Erhöhte Komplexität: Die Komplexität wird durch eine Vielzahl von Klassen erhöht.
- Gefahr von exzessiv vielen Änderungen: Wenn ein Benutzereingriff in zahlreichen Aktualisierungen mündet, ist zu überlegen, wie unnötige Aktualisierungen vermieden werden können. Eventuell sind nicht alle Ansichten an jeder Art von Änderungen interessiert, da sie nur einen Ausschnitt der Daten repräsentieren.
- Private Beziehungen zwischen Ansicht (View) und Bediener (Controller): Eine Ansicht und ein Bediener können in einer Beziehung stehen, die verhindert, dass die Komponenten individuell wiederverwendet werden können.
- Enge Kopplung der Ansichten (Views) und Bediener (Controller) an das Modell (Model): Beobachter unternehmen direkte Methodenaufrufe beim Modell. Falls die Schnittstelle des Modells verändert werden sollte, so sind von dieser Änderung ebenso alle Ansicht- und Bedienerklassen betroffen.
- Ineffizienter Datenzugriff in Ansichten (Views): Je nach Schnittstelle des Modells sind gegebenenfalls mehrere Funktionsaufrufe beim Modell erforderlich.

8 Prototyp Resdata

8.1 Überblick

An dieser Stelle wird das Software-Design der Webanwendung beschrieben. Die Anwendung wurde mittels der Technologie Java Server Pages (JSP) umgesetzt. Als Softwaremuster wurde das Model-View-Control (MVC) verwendet. Zur Erklärung der allgemeinen Techniken (JSP, MVC) siehe bitte in den entsprechenden Abschnitten.

8.2 Aufgabe

Um das Ziel umzusetzen, wurde folgende Aufgabe für die Beispielwebanwendung definiert:

1. Ein Administrator hat die Möglichkeit, Ärzte anzulegen und zu löschen.
2. Die Ärzte haben die Möglichkeit, sich nach einer Authentifizierung ihre Daten anzuschauen und zu ändern.
3. Die Patienten haben die Möglichkeiten, sich die Ärzte anzuschauen, nach Ärzten zu suchen und sich freie Termine zum ausgesuchten Arzt anzuschauen.
4. Weiterhin besteht die Möglichkeit für den Patienten, sich einen Termin bei einem Arzt geben zu lassen. Dazu muß er die Symptome und seine eMail-Adresse angeben. Dieser Termin wird als Wunsch in die Datenbank geschrieben und zusätzlich als eMail an den Arzt geschickt.
5. Der Arzt hat wiederum die Möglichkeit, dem Terminwunsch des Patienten zu entsprechen oder den Termin abzusagen. Dies geschieht wieder durch das Verschicken einer eMail an den Patienten. Erst nach einer Bestätigung durch den Arzt ist der Termin als verbindlich zu betrachten.

Es bestände auch die Möglichkeit, die Termine automatisch durch das System bestätigen oder absagen zu lassen. Doch dies würde nicht den unterschiedlichen Beschwerden der Patienten gerecht werden. Die Behandlungszeit der Patienten ist zu unterschiedlich. Darum kann die Entscheidung effektiv nur durch den Arzt oder die Krankenschwester getroffen werden. Darum wurde die manuelle Bestätigung für die Webanwendung umgesetzt.

Abgrenzung der Aufgabe:
Es wird keine Patientenverwaltung realisiert. Weiterhin ist die Umsetzung nur als Prototyp zu betrachten.

8.3 Besonderheiten des MVC-Musters für die Webanwendung

Ein typischer Ablauf einer Webanwendung läuft nach folgendem Schema ab. Der Client stellt eine Anforderung an den Server. Der Server antwortet an den Client. Es ist aber keine Kommunikation vom Server zu einem bestimmten Client vorgesehen, d.h. der Server kann keine Kommunikation mit dem Client betreiben, wenn der Client nicht vorher eine Anforderung an den Server gestellt hat. Damit ist das Beobachtungsmuster, das im normalen MVC verwendet wird, nicht umsetzbar. Der Server kann von sich aus, z.B. durch Datenänderung, nicht automatisch alle Clients zu einer Aktualisierung veranlassen.

Darum wird hier folgendes vom allgemeinen MVC-Model nicht umgesetzt:

- Beobachter
  Keine Implementierung eines 'Beobachters'. Im Standard MVC-Muster sollen bei Änderungen des Models die Views benachrichtigt werden. Dies wird hier nicht umgesetzt.

- Registrierungsmechanismus
  Es wird kein Registrierungsmechanismus implemetiert. Im normalen Framework werden die Views, Models und Controller in Listen registriert. Dies ist für den 'Beobachter' notwendig. Da dies aber nicht umgesetzt wird, so ist auch der Registrierungsmechanismus nicht erforderlich.

Hier werden in Bezug auf die Webanwendung (JSP) die verwendeten Komponenten des MVC-Musters erklärt. Zusätzlich kommt die Komponente des Servlets dazu. Diese hat die Aufgabe, die Kommunikation zwischen Client und Server herzustellen. Um ein einfaches Design, aber auch eine Trennung zwischen Modell und Ansicht zu gewährleisten, wurde das Design folgendermaßen umgesetzt:

- View
  Die Views sind meine JSP-Seiten. Da die Views keine Java-Anwendung sind, gibt es auch keine Hierarchie innerhalb der Views. In den Views werden das jeweilige Modell und der Controller bekanntgegeben.

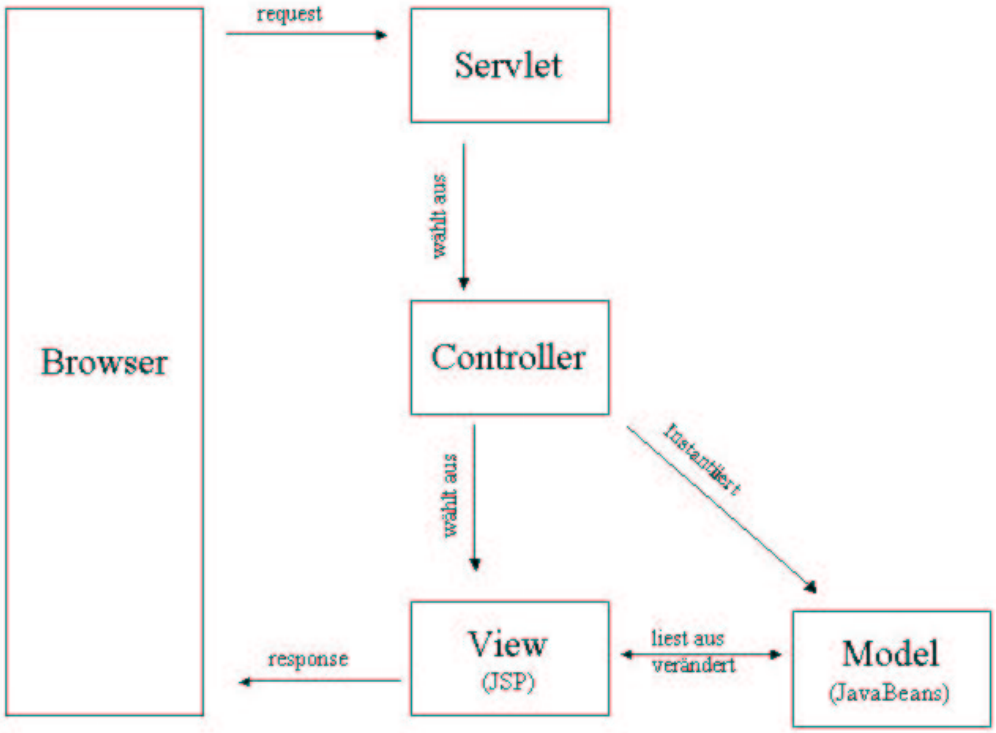
- Controller
  Empfängt die gesamten Parameter vom Servlet. Entscheidet über die Parameter, was und mit welchem Modell etwas durchgeführt wird. Als Ergebnis gibt es eine Seite an das Servlet zurück (Fehlerseite, Anzeigeseite, ...).

- Model
  Dieses führt entsprechend den Anweisungen des Controllers seine Aufgaben aus. Die Zustände und Werte sind nur in dem Model bekannt. Darauf können der Controller und die View über Schnittstellen zugreifen. Weiterhin ist vom Modell der Zugriff auf die Daten in der Datenbank und das Verschicken von eMails möglich.

- Servlet
  Dient zur Kommunikation der Webanwendung und des Controllers. Hier werden die Parameter in eine Hashtabelle geschrieben. Zu den Parametern gehören:
  - Model
  - Action
  - Controller
  - sonstige Parameter (aus Eingabefeldern)

  Die Parameter werden an den entsprechenden Controller gesendet. Dieser führt für den Parameter 'Action' die entsprechenden Aktionen im Model aus. Als Ergebnis wird dem Servlet die Seite bekannt gegeben, die es als nächstes ausgeben soll.

Das MVC-Model sieht dann für die Umsetzung der Webapplication mittels JSP folgendermaßen aus.



8.4 Zugriff auf die Datenbank

8.4.1 Überblick

Für die Datenhaltung des Moduls ResData des Projektes Resmedicinae wird die Datenbank mySQL genutzt. Auf diese muß von Java aus zugegriffen werden. Dies erfolgt über die standardisierte Schnittstelle JDBC (Java Database Connectivity). Da das allgemeine Vorgehen (Aufbau einer Verbindung, Anfrage/Anforderung stellen, Schließen einer Verbindung) sehr langsam ist, wird ein sogenannter Connection-Pool verwendet.

8.4.2 JDBC

JDBC kapselt die Datenbank von der Anwendung. Durch diese standardisierte Schnittstelle ist es fast unwichtig, welche relationale Datenbank hinter der Schnittstelle arbeitet. Am Anfang muß nur ein entsprechender Treiber geladen werden. Danach ist der Zugriff über diese Schnittstelle auf alle Datenbanken gleich. Dies erfolgt über die JDBC-API. Wird in der Anwendung darauf geachtet, daß der Zugriff auf die Datenbank nur über Standard SQL(Structured Query Language) erfolgt, so ist im allgemeinen der Austausch der Datenbank unproblematisch. Einzig bei Verwendung von SQL-Konstrukten, die nicht dem Standard angehören, gibt es Probleme bei Umstellungen der Datenbanken.

8.4.3 Connection-Pool

Für jede Transaktion (Anfrage/Anforderung) wird eine neue Verbindung aufgebaut, benutzt und wieder geschlossen. Dies ist eine sehr aufwendige Operation und dauert entsprechend lange. Wird eine Seite viel besucht, so wird die Seite durch den ständigen Auf- und Abbau der Datenbankverbindung in die Knie gezwungen. Eine übliche Lösung für dieses Problem ist die Verwendung eines Connection-Pools. Der Connection-Pool hält eine definierte Anzahl von Verbindungen zur Datenbank. Wird von der Anwendung eine Verbindung gebraucht, so baut sie die Verbindung nicht mehr auf, sondern stellt eine Anfrage an den Connection-Pool. Dieser stellt eine unbenutzte Verbindung der Anwendung zur Verfügung. Nach dem Ende der Benutzung gibt die Anwendung die Verbindung zur weiteren Benutzung an den Connection-Pool zurück. Dadurch wird nicht jedesmal eine neue Verbindung aufgebaut, sondern eine bestehende genutzt. Dies bedeutet bei vielen Anfragen einen enormen Geschwindigkeitsvorteil der Anwendung. Einzig die Initialisierung des Connection-Pools dauert etwas länger. Dies passiert aber nur nach einen Neustart des Applicationservers und der ersten Datenbankanfrage bzw. nach einem „Reconnect“ zu der Datenbank.

8.4.4 Besonderheiten zu mySQL

MySQL trennt standardmäßig die Datenbankverbindung nach 8 Stunden Nichtbenutzung. Dadurch muß ein Wiederverbinden nach der Trennung erfolgen. Dies passiert mit dem Parameter 'autoReconnect' beim Aufbau der Datenbankverbindung. In dem Modul ResData wurde dies folgendermaßen umgesetzt:

```
jdbcProperties.put("user",     ResDataConfig.getConfig().getDatabaseUser() );
jdbcProperties.put("password", ResDataConfig.getConfig().getDatabasePwd()  );
jdbcProperties.put("autoReconnect", "true");
```

8.5 Mail

8.5.1 Überblick

Für das Modul Resdata des Projektes ResMedicinae wird ein eMail-Account gebraucht. Diesr wurde speziell für das Modul angelegt. Als erstes wird hier das allgemeine Vorgehen in Java und als zweites der spezielle eMail-Account beschrieben.

8.5.2 Vorgehen

Um eMails in Java zu verschicken, wird die mailapi.jar von SUN genutzt. Diese stellt die Funktionen zum Versenden von eMails schon bereit. Folgende Punkte sind für den Prototyp zu betrachten:

- Protokoll
  Im Prototyp wird das Versenden der eMails über das Protokoll SMTP (simple mail transfer protocol) realisiert. Andere Protokolle werden in dem Prototyp nicht unterstützt. SMTP sollte für fast alle Fälle ausreichend sein.
- Authentifizierung
  Die meisten eMail-Provider schützen ihr SMTP-Protokoll durch eine Authentifizierung. Dies wird auch durch die mailapi.jar abgedeckt. Dazu muß eine Klasse vom Authenticator abgeleitet werden. In dieser muß die Funktion getPasswordAuthentication definiert werden, die die Anmeldung vornimmt.

Für weitere Infos zur Umsetzung des eMail-Versands in Java lesen Sie den Quelltext folgenden zwei Klassen:

- org.resmedicinae.application.healthcare.resdata.ResDataMailAuthenticator
- org.resmedicinae.application.healthcare.resdata.model.ResDataMail

8.5.3 Technische Daten

Hier sehen Sie die technischen Kenndaten des Accounts.

Tabelle 2: eMail-Account

Schlüssel	**Wert**
eMail-Adresse	resdata@web.de
Pop3-Server	pop2.web.de
SMTP-Server	smtp.web.de
Login	resdata
Paßwort	resmedicinae

8.6 URL-Rewriting

Für das URL-Rewriting sind vorher kurz folgende Begriffe zu erklären:

- Session, Session-Id
  Eine Session ist eine Sitzung zwischen einem Server und einem Client. Das Erkennen der gleichen Session wird über eine Session-Id realisiert.
- Cookie
  Cookies erlauben es Servern, Informationen auf dem Client abzulegen. Diese Information kann der Server wieder abrufen. Somit wird es auf einfachem Wege möglich, den jeweiligen Client wiederzuerkennen, indem die Session-Id im Cookie auf dem Client gespeichert wird.

Das normale Verhalten eines Servers ist das Ablegen der Session-Id in ein Cookie. Wird von dem gleichen Client wieder eine Anfrage gestartet, so liest der Server das Cookie, und anhand der Session-Id weiß der Server, dass er sich in der gleichen Session befindet. Das Problem ist, dass Cookies clientseitig behandelt werden. Es gibt die Möglichkeit im Browser, Cookies prinzipiell zu verbieten. Damit würde der Server die Session-Id nicht auslesen können, und für den Server wäre jede Anfrage des Clients eine neue Anfrage und somit eine neue Session. So kann man aber keine Information (wie z.B. Benutzer X am System angemeldet) speichern und keine personenbezogenen Dienste (Webshop, Ärzteverwaltung,...) anbieten.

Eine andere Möglichkeit der Übertragung der Session-Id ist das URL-Rewriting. Dabei wird die Session-Id als Parameter in der URL mit übergeben. Dazu ist aber eine Anpassung jeder URL notwendig, damit die Session-Id mit übergeben wird. Dies geschieht mit der Java-Funktion

```
response.encodeURL(<URL>),
```

wobei <URL> durch die entsprechende URL zu ersetzen ist. Response ist ein implizites Objekt und somit in jeder JSP-Seite bekannt.

8.7 Verzeichnisstruktur

Hier wird auf die Verzeichnisstruktur der Web-Anwendung eingegangen. Dies kann von Applicationserver zu Applicationserver unterschiedlich sein. Darum wird hier nur die Verzeichnisstruktur, wie sie unter Tomcat benutzt wird, behandelt.

/WEB-INF Dieses Verzeichnis muss im Wurzelverzeichnis jeder Web-Applikation vorhanden sein. Es enthält Ressourcen, welche nicht direkt an Clients geschickt werden.

Der Zugriff auf Dateien in diesem Verzeichnis über HTTP ist nicht möglich. In WEB-INF werden unter anderem übersetzte Java- Klassen, JAR-Archive und Deployment-Deskriptoren abgelegt. Da Dateien im Verzeichnis WEB-INF nicht über HTTP erreichbar sind, stehen HTML-Dokumente, Bilder, CSS-Stylesheets etc. nicht in diesem Verzeichnis. Alle serverseitig ausgeführten Programme werden in WEB-INF gespeichert.

9 Konfiguration

9.1 Inhalt

In diesem Abschnitt kann nur eine Beispielkonfiguration für die Webapplication beschrieben werden. Dies beschränkt sich auf ein bestimmtes Betriebssystem und auf bestimmte Tools. Bei anderen Betriebssystemen, bei anderen Tools und anderen Versionen von Betriebssystemen und Tools kann die Konfiguration anders aussehen. Im Prinzip sollte aber der Ablauf ähnlich sein.

Abgrenzung: An dieser Stelle wird nicht die Installation und Konfiguration des Betriebssystems beschrieben. Weiterhin wird hier auch nicht die Installation der verwendeten Tools behandelt. Dies würde den Rahmen des Kapitels sprengen. Dies ist die Aufgabe des Administrators.

9.2 Toolauswahl

Für die Webapplication werden folgende Teile gebraucht.

- Betriebssystem
- Java-Version
- Webserver
- Servlet- und JSP-Engine (Applicationserver)
- Datenbank,

wobei bei der Beispielkonfiguration der Webserver und der Applicationserver von Tomcat abgedeckt werden. Im normalen Betrieb werden Webserver und Applicationserver getrennt und nur die Servlet- und JSP-Anfragen werden zum Applicationserver weitergeleitet.

Die Beispielkonfiguration wurde unter folgendem Betriebssystem und folgenden Tools realisiert:

9.3 Voraussetzung

Für die Konfiguration und den Betrieb der Webanwendung wird folgendes vorausgesetzt:

- installiertes Java

Tabelle 3: Tabelle zur Toolauswahl

Aufgabe	**Name**	**Version**	**Beschreibung**
Betriebssystem	Linux Debian	3.0	Codename Woody
Java-Version (Standard)	Java 2 SDK	1.4.1	
Java-Version (Enterprise)	Java 2 SDK EE	1.3.1	
Web- und Applicationserver	Tomcat	4.0.6	Servlet 2.3, JSP 1.2
Datenbank	mySQL	3.23	

- Applicationserver (Tomcat), läuft auf Port 8080
- Datenbank (mySQL), läuft
- Webanwendung ist auf dem Server unter folgendem Verzeichnis verfügbar:

  /home/resmedicinae

  Dazu müssen von SourceForge vom Projekt „Resmedicinae“ die Module

  – lib
  – etc
  – share
  – src
  – bin

  heruntergeladen werden.

9.4 Konfiguration Tomcat

Die allgemeine Serverkonfiguration von Tomcat erfolgt in folgender Datei:

${TOMCAT_HOME}/conf/server.xml

Die Datei ist eine XML-Datei und folgt also der XML-Syntax. Die Änderung ist an folgender Stelle im XML-Syntaxbaum vorzunehmen:

```
<Server ...>
  <Service name="Tomcat-Standalone" ...>
    <Engine name="Standalone" ...>
      <Host name="localhost" ...>

      </Host>
    </Engine>
```

```
  </Service>
</Server>
```

Folgende Änderung ist in dem Baum einzutragen:

```
<Context path="/resdata"
         docBase="/home/resmedicinae/share/application/healthcare/resdata">
  <Parameter name="resdata.xml"
             value="/home/resmedicinae/etc/resdata.xml">
  </Parameter>
</Context>
```

wobei die Werte folgende Bedeutungen haben:

Tabelle 4: Tabelle zur Beschreibung der Werte für die server.xml

Schlüssel	**Bedeutung**
path	Mappingpfad zum Stammverzeichnis
docbase	Stammverzeichnis der Webapplication
name	Name des Parameters. Dieser ist für die Webapplication „resdata“ immer „resdata.xml“
value	Wert des Parameters - für den Parameter „resdata.xml“ der Ort der Konfigurationsdatei

Weiterhin sind die Variablen JAVA_HOME und der CLASSPATH für TOMCAT zu setzen. Dies geschieht am besten in folgender Datei:

${TOMCAT_HOME}/bin/catalina.sh

- JAVA_HOME
  Wird gleich am Anfang des Skriptes gesetzt: z.B. export JAVA_HOME=/usr/local/java
- CLASSPATH
  Wird nach der Sektion ”# Add on extra jar files to CLASSPATHëingefügt. Ist alles standardmäßig eingerichtet worden, so muß hier folgende Zeile ergänzt werden:

  CLASSPATH=”$CLASSPATH”:
  /home/resmedicinae/lib/protomatter-1.1.7.jar:
  /home/resmedicinae/lib/mm.mysql-2.0.14-bin.jar:
  /home/resmedicinae/lib/mailapi.jar

9.5 Konfiguration mySQL

Für die Konfiguration von mySQL sind keine speziellen Einstellungen nötig. Das einzige, was gemacht werden muß, ist die Erstellung der Datenbank. Dies geschieht mit den mitgelieferten Scripten.

${RESMEDICINAE_HOME}/bin/application/healthcare/resdata/build_database.sh

Ist mySQL im Standardverzeichnis installiert, sollte das Skript sofort funktionieren. Ansonsten muß in der

${RESMEDICINAE_HOME}/bin/set_home.sh

der Parameter MYSQL_HOME gesetzt werden.

9.6 Konfiguration der Webapplication

Für die Anwendung werden alle veränderlichen Größen in einer Konfigurationsdatei abgelegt. Diese Konfigurationsdatei ist im XML-Schema abgespeichert.

An dieser Stelle werden die Einträge beschrieben:

Tabelle 5: Einträge der Konfigurationsdatei Sektion „logging“

Schlüssel	**Beschreibung**
log_level	Loglevel der Application (1..15)
log_file	Datei der Logeinträge

Tabelle 6: Einträge der Konfigurationsdatei Sektion „database“

Schlüssel	**Beschreibung**
driver	Treiber der JDBC-Verbindung (org.gjt.mm.mysql.Driver)
url	Verwendungs-URL für die JDBC-Verbing (jdbc:mysql://localhost/resdata)
user	Verwendete Benutzer der Datenbank (resdata)
pwd	Verwendetes Paßwort der Datenbank (resdata)

Tabelle 7: Einträge der Konfigurationsdatei Sektion „mail“

Schlüssel	**Beschreibung**
protocol	Protokoll der eMail-Verschickung (smtp)
host	Host für die eMail-Verschickung (mail.gmx.de)
user	Login für das eMail-Konto (xxx)
pwd	Paßwort für das eMail-Konto (???)
fromadresse	Absenderadresse für die eMail-Verschickung (xxx@yyy.net)
subject_desired	Betreff für eMail-Verschickung Terminwunsch (Web Terminwunsch)
subject_confirm	Betreff für eMail-Verschickung Terminbestätigung (Terminbestätigung)
subject_cancel	Betreff für eMail-Verschickung Terminabsage (Terminabsage)

Tabelle 8: Einträge der Konfigurationsdatei Sektion „path“

Schlüssel	**Beschreibung**
servlet_path	Pfad für das Servlet (http://localhost:8080/resdata/servlet/ org.resmedicinae.application.healthcare.resdata.controller)
jsp_path	Pfad für JSP-Dateien (/jsp/)
css_path	Pfad für CSS-Dateien (http://servername:8080/resdata/css/)
image_path	Pfad für Bild-Dateien (http://servername:8080/resdata/images/)

Tabelle 9: Einträge der Konfigurationsdatei Sektion „navigation“

Schlüssel	**Beschreibung**
resmedicinae_home	URL der Home-Seite für Resmedicinae (http://resmedicinae.sourceforge.net/index.html)
resdata_home	URL der Home-Seite für ResData (http://servername:8080/resdata/jsp/ResDataStart.jsp)

Tabelle 10: Einträge der Konfigurationsdatei Sektion „parameter“

Schlüssel	**Beschreibung**
control_param	Instanzbezeichnung für den Controller (myController)
loginmodel_param	Instanzbezeichnung für das Modell Login (myModelLogin)
doctormodel_param	Instanzbezeichnung für das Modell Doctor (myModelDoctor)
datemodel_param	Instanzbezeichnung für das Modell Date (myModeldate)
mailmodel_param	Instanzbezeichnung für das Modell Mail (myModelMail)

Tabelle 11: Einträge der Konfigurationsdatei Sektion „color“

Schlüssel	**Beschreibung**
status_no	Farbe für freie Termine (blue)
status_request	Farbe für angeforderte Termine (red)
status_confirmed	Farbe für bestätigte Termine (yellow)

10 Zusammenfassung, Ausblick

Zusammenfassend läßt sich sagen, dass die Umsetzung von Webapplikationen mit JSP gut realisierbar sind. Dazu trägt einerseits die plattformunabhängige Programmiersprache Java und ihre definierten Schnittstellen (JDBC, Reflection, Tag) bei. Andererseits bietet JSP die Möglichkeit, über JavaBeans und Tags die Anwendungslogik und die Darstellung zu trennen. Damit ist eine getrennte Entwicklung (Programmierer, Designer) der Verarbeitungslogik und des Designs möglich. Darauf muß aber bei der Erstellung der JSP-Seiten Rücksicht genommen werden. Ziel ist es, so wenig wie möglich Quelltext in den JSP-Seiten zu verwenden. Dies ist aber nicht zwingend bei JSP vorgeschrieben. Programmierer und Designer müssen sich auf eine Schnittstelle für die Übergabe der Parameter und den Zugriff auf Modell und Controller einigen.

Das MVC-Muster konnte teilweise für die Entwicklung der Webapplikationen umgesetzt werden. Eine vollständige Realisierung war aufgrund der speziellen Kommunikationswege der Webapplikation nicht möglich. Es besteht keine Möglichkeit, Aktualisierungen der Darstellung nach Änderungen von Daten auf dem Sever vom Server zu veranlassen. Damit kann kein Beobachtungsmuster implementiert werden. Die Umsetzung des MVC-Muster wurde für die Webapplikation neu erstellt. Eine weitere Möglichkeit wäre, auf ein bestehendes Framework die Applikation aufzubauen. Damit würde man auf die Ressourcen und Erfahrungen von mehreren Entwicklern zugreifen und Entwicklungsaufwand für ein eigenes Framework minimieren.

Durch die definierte Schnittstelle JDBC von SUN ist es möglich, durch Einsatz entsprechender Treiber die Datenbank flexibel auszutauschen. Die Anwendung ist SQL-2 Entry-Level-Standard von 1992 konform umgesetzt worden, so daß keine datenbankspezifische Anpassungen vorzunehmen sind. In diesem Prototyp ist die Speicherung der Daten nicht in XML gemacht worden. Dies wäre eine weitere Möglichkeit, die Daten abzulegen.

Im Prinzip ist der Prototyp in dem jetzigen Umfang einsetzbar. Zu beachten ist aber, das für die Konfiguration und Wartung der Server (Webserver, Applikationserver) der Administrator verantwortlich ist. Für Webapplikationen sind gewisse Sicherheitsrichtlinien einzuhalten. Weiterhin ist eine Kontrolle der Sicherheitsrichtlinien und Wartung des Servers Aufgabe des Administrators. Dies ist nicht immer ganz einfach und deswegen ist hier Vorsicht geboten.

Literatur

[1] H.R.Hansen: Wirtschaftsinformatik I 7. Auflage

[2] Hobbs, Ashton: JDBC in 21 Tage

[3] Michael Morris: JavaBeans

[4] David Flanagan: Java in a Nutshell

[5] Axel Faltin: http://www.java-beans.com/

[6] Buschmann, Meunier, Rohnert,...: Pattern-orientierte Software-Architektur

[7] Stefan Wille: Go To Java Server Pages

[8] Guido Krüger: Handbuch der Java-Programmierung

A GNU Free Documentation License

Version 1.2, November 2002

PREAMBLE

The purpose of this License is to make a manual, textbook, or other functional and useful document "free" in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or non-commercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

A.1 APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you". You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A "Modified Version" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A "Secondary Section" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "Invariant Sections" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The "Cover Texts" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A "Transparent" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not "Transparent" is called "Opaque".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The "Title Page" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

A section "Entitled XYZ" means a named subunit of the Document whose title either

is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as “Acknowledgements”, “Dedications”, “Endorsements”, or “History”.) To “Preserve the Title” of such a section when you modify the Document means that it remains a section “Entitled XYZ” according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

A.2 VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section A.3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

A.3 COPYING IN QUANTITY

If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document’s license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque

copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

A.4 MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections A.2 and A.3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.

B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.

C. State on the Title page the name of the publisher of the Modified Version, as the publisher.

D. Preserve all the copyright notices of the Document.

E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.

F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.

G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.

H. Include an unaltered copy of this License.

I. Preserve the section Entitled "History", Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.

J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.

K. For any section Entitled "Acknowledgements" or "Dedications", Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.

L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.

M. Delete any section Entitled "Endorsements". Such a section may not be included in the Modified Version.

N. Do not retitle any existing section to be Entitled "Endorsements" or to conflict in title with any Invariant Section.

O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section Entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties–for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already

includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

A.5 COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section A.4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections Entitled “History” in the various original documents, forming one section Entitled “History”; likewise combine any sections Entitled “Acknowledgements”, and any sections Entitled “Dedications”. You must delete all sections Entitled “Endorsements”.

A.6 COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

A.7 AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an "aggregate" if the copyright resulting from the compilation is not used to limit the legal rights of the compilation's users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

If the Cover Text requirement of section A.3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document's Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

A.8 TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section A.4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

If a section in the Document is Entitled "Acknowledgements", "Dedications", or "History", the requirement (section A.4) to Preserve its Title (section A.1) will typically require changing the actual title.

A.9 TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided for under this License. Any other attempt to copy, modify, sublicense or distribute the Document is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

A.10 FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See http://www.gnu.org/copyleft/.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation.

ADDENDUM: How to use this License for your documents

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

> Copyright © YEAR YOUR NAME. Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.2 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled "GNU Free Documentation License".

If you have Invariant Sections, Front-Cover Texts and Back-Cover Texts, replace the "with...Texts." line with this:

with the Invariant Sections being LIST THEIR TITLES, with the Front-Cover Texts being LIST, and with the Back-Cover Texts being LIST.

If you have Invariant Sections without Cover Texts, or some other combination of the three, merge those two alternatives to suit the situation.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.