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
Fakultät: Informatik und Automatisierung
Matrikelnummer: 23300

21. März 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 Terminplannung mit Anbindung an ein medizinisches Informationssystem. Dafür soll einen 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 Erweiterbarkeit der Anwendung Wert gelegt worden. Dafür ist es sinnvoll auf Erfahrungen von anderen zurück zu greifen. Diese Erfahrungen sind in Entwurfsmuster 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 Artzpraxis. Diese wurde nach dem Model-View-Controller Muster und den hier vorgestellten Technologien realisiert.

1.2 Umfeld

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

2 Web-Architektur

2.1 Überblick

In diesen 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, sonder 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, sodass dieser z. B. eine Datenbank abfragen kann oder Programme ausführen kann, die auf dem Server installiert sind. Application-Server bieten häufig zusätzlich Sicherheitsmerkmale, Lastenverteilung (load balancing) und Ausfallmechanismen (failover mechanisms) 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 diesen Server läuft eine Datenbank, die diese Aufgaben übernimmt.
eMail-Server:	Dieser Server stellt Dienste für das Verschicken und Empfangen von von eMail's 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 einen Rechner betrieben werden.

3 JDBC

JDBC steht für Java Database Connectivity und ist die Datenbankschnittstelle für Java. Ab den JDK 1.1 bietet Java durch JDBC eine vollständige Datenbankunterstützung zum Zugriff auf relationale Datenabanken 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 eine 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. Durch die JDBC-Spezifikation wird eine maximale Flexibilität sowohl für Datenbankbenutzer als auch für Datenbankanbieter gewährleistet.

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.

- Clientseitig Verarbeitung
  Die Antwort des Servers wird erst auf dem Client verarbeitet. Ein Beispiel für diese Techniken wären Javascripts und Applets.

- Serverseitig 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 Softwarekompoenete 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. Session dienen dazu, Sitzungen eines Benutzers zu verfolgen und um 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 mehrer Anfragen an das Servlet gestellt werden, ohne das es dabei neu initialisiert wird. Das bedeutet, das ein Servlet mehrere Anfrage und damit mehrere Benutzer bedienen kann, obwohl es nur einmal im Speicher geladen wird.

Bekommt eine Servlet eine Anfrage, so geschieht das mit service(ServletRequest, ServletResponse). Das Servlet liest aus den 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 Ergebis (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 vertstärkte Wunsch nach einer Widerverwendung 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 das Reflectionmechanismus von Java oder bei Nichteinhaltung über die BeanInfo-Schnittstelle geschehen.

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

5.3 Konvensionen

JavaBeans folgen im allgemeinen einen 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 eine Menge von Properties
- für den lesenden und schreibenden Zugriff auf die Properties werden get- und set-Methoden definiert

Werden die get- bzw. set-Methoden weggelssen, so hat man entweder nur lesend oder nur schreiben auf die Eigenschaften (Properties) zugriff.

Weiter Information zu Konvensionen 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 basiert auf die Servlettechnologie von Sun. In diesen Kapitel wird die JSP-Technologie kurz vorgestellt. Es werden aber keine Vergleiche zu anderen Techniken behandelt. Es wird auch keine Sprachbeschreibung für JSP hier geben. Hier wird nur das prinzipielle Vorgehen der JSP-Technologie in Hinblick für die Trennung der Ansicht und der Application 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 eingebetteten 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 exestiert oder der JSP-Quelltext geändert wurde. Dieses Servlet wird bearbeitet und das Ergebnis wird in den Tag's der JSP-Seite hinzugefügt. Erst diese generierte Seite wird dem Benutzer auf seine Anfrage zurück geschickt.

Dies ist aus 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 kann über das Bean die Methoden der Klasse genutzt werden. Auf die Beans kann 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 habe 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 den der Tagname Gültigkeit hat. Dies dient dazu, das es keine Namenskonflikte gibt, 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 noch auf keine 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 in dem 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 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. Darauf hin ändert jede View ihre Ansicht und stellt die neuen Daten dar. Die Views kommunizieren nicht mit einander und das Modell weiß nicht, in welche 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 verwendeten Komponenten.

7.4 Bewertung des MVC-Musters

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

Vorteile:

- Mehrere Ansichten des selben Modells: Das MVC-Muster trennt das Modell strikt von der Benutzungsschnittstelle. Das ermöglicht es, mehrere Ansichten der selben 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 Ärtze anzulegen und zu löschen.
2. Die Ärtze haben die Möglicheit, sich nach einer Authentifizierung Ihre Daten anzuschauen und zu ändern.
3. Die Patienten haben die Möglichkeiten, sich die Ärtzte anzuschauen, nach Ärtzte zu suchen und zu den ausgesuchten Artz sich die freien Termin anzuschauen.
4. Weiterhin besteht die Möglichkeit für den Patient, sich einen Termin bei einen Artzt geben zu lassen. Dazu muß er die Beschwerde und seine eMail-Adresse angeben. Dieser Termin wird als Wunsch in die Datenbank geschrieben und zusätzlich als eMail an den Artz geschickt.
5. Der Artzt hat wiederum die Möglichkeit, den 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 Artzt 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 des Patienten gerecht werden. Die Behandlungszeit der Patieneten ist halt doch zu unterschiedlich und dies kann effektiv nur der Arzt oder die Krankenschwester entscheiden. Darum ist 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 folgenden Schema ab. Der Client stellt eine Anforderung an den Server. Der Server antwortet an den Client. Es ist aber

keine Kommunikation vom Server zu einen bestimmten Client vorgesehen, d.h. der Server kann keine Kommunikation mit den 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 umgestezt:

- Beobachter
  Keine Implemtierung eines 'Beobachters'. Im Standard MVC-Muster soll bei Änderung des Modells die Views benachrichtigt werden. Dies wird hier nicht umgesetzt.

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

Hier wird in Bezug auf die Webanwendung (JSP) die verwendeteten Komponenten des MVC-Musters erklärt. Zusätzlich kommt die Komponente des Servlets dazu. Diese hat die Aufgabe die Kommunikatuion zwischen Client und Server herzustellen. Um ein einfaches Design, aber auch eine Trennung zwischen Modell und Ansicht wurde das Design folgender Maßen umgestezt:

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

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

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

- Servlet
  Dient für die Kommunikation der Webanwendung und zum Kontroller. Hier werden die Parameter in eine Hashtabelle geschrieben. Zu den Parameter gehören:

  - Model
  - Action

– Controller

– sonstige Parameter (aus Eingabefelder)

Die Paremeter werden an den entsprechenden Controller gesendet. Dieser führt entsprechend der Action entsprechende Aktionen im Modell aus. Als Ergebnis wird dem Servlet die Seite bekanntgegeben, die es als nächstes ausgeben soll.

Das MVC-Model sieht dann für die Umsetzung der Webapplication mittels JSP folgender Maßen 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 einen 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 arbeit. Am Anfang muß nur ein entsprechender Treiber geladen werden. Danach ist der Zugriff über diese Schnittstelle auf alle Datenbanken gleich. Dies Erfogt über die JDBC-API. Wird in der Anwendung darauf geachtet, daß der Zugriff auf die Datenbank nur über den Standard SQL(Structured Query Language) erfolgt, so ist im allgenmeinen der Austausch der Datenbank unproblematisch. Einzig bei Verwendung von SQL-Konstrukte, 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 aufwendig Operation und dauert entsprechend lang. 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-Pool. 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 gibt 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 einen neue Verbindung aufgebaut, sondern eine bestehende genutz. Dies bedeutet bei vielen Anfragen einen enormen Geschwindigkeitsvorteil der Anwendung. Ich habe keine Geschwingkeitstest für Connection-Pools gemacht. In der Literatur wird dieser aber mit bis zu 40 mal schneller angegeben. Einzig die Initialisierung des Connection-Pools dauert etwas länger. Dies passiert aber nur nach einen Neustart des Applicationserver und der ersten Datenbankanfrage bzw. nach einen „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. Dies wurde speziell für das Modul angelegt. Als erstes wird hier das allgemeine Vorgehen in Java und als zweites wird 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 Sachen sind für den Prototyp zu betrachten:

- Protokoll
  In dem 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 von Authenticator abgeleietet 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 die den Quelltext von 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 möchte ich vorher noch kurz folgende Begriffe erklären:

- Session, Session-Id
  Eine Session ist ein Sitzung zwischen einen Server und einen 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 abzurufen. Somit wird es auf einfachem Wege möglich den jeweiligen Client wieder zuerkennen, in dem 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 den Cookie und anhand der Session-Id weiß der Server, das er sich in der gleichen Session befindet. Das Problem ist, das 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 personenbezogene 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 ist 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 auf die Verzeichnisstruktur, wie sie unter Tomcat benutz 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 diesen 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 Betriebsystemen und Tools kann die Konfiguration anders aussehen. Im Prinzip sollte aber der Ablauf ähnlich aussehen.

Abgrenzung: An dieser Stelle wird nicht die Installation und Konfiguration des Betriebssystems beschrieben. Weiterhin wird hier auch nicht die Installation der verwendeten Tools behandeltet. 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 in unsern Fall der Webserver und der Applicationserver von Tomcat abgedeckt wird. Im normalen Betrieb wird der Webserver und der Applicationserver getrennt und nur die Servlet- und JSP-Anfragen werden zum Applicationserver weitergeleitet.

Die Beispielkonfiguration wurde unter folgenden Betriebssstem und folgenden Tools realisiert:

Tabelle 3: Tabelle zur Toolauswahl

Aufgabe	**Name**	**Version**	**Beschreibung**
Betriebssstem	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	

9.3 Vorrausetzung

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

- installiertes Java
- Applicationserver (Tomcat) läuft auf Port 8080
- Datenbank (mySQL) läuft
- Webanwendung ist auf dem Server unter folgenden Verzeichnis verfügbar:

  /home/resmedicinae

  Dazu muß von SourceForge von den 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-Systaxbaum vorzunehmen:

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

      </Host>
    </Engine>
  </Service>
</Server>
```

Folgende Änderung ist in den 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.xnml

Schlüssel	**Bedeutung**
path	Mappingpfad zum Stanmmverzeichnis
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 wurden, 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 WEinstellungen von Nöten. Das einzigste, 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.

Die Konfigurationsdatei ist im Anhang sichtbar. An dieser Stelle werden die Einträge beschrieben:

Tabelle 5: Einträge der Konfigurationsdatei Sektion „loging“

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)
pwd	Verwendete Benutzer der Datenbank (resdata)
pwd	Verwendetet 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-Verschieckung Terminwunsch (Web Terminwunsch)
subject_confirm	Betreff für eMail-Verschieckung Terminbestätigung (Terminbestätigung)
subject_cancel	Betreff für eMail-Verschieckung 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 Termin (blue)
status_request	Farbe für angeforderte Termin (red)
status_confirmed	Farbe für bestätigte Termine (yellow)

10 Zusammenfassung

Java Server Pages stellen eine gute Möglichkeit zur Umsetzung von Webapplication dar. In ihr lassen sich durch die Verwendung von JavaBeans und Tag-Libraries sehr gut die Darstelung von der Verarbeitungslogik trennen. Somit steht der Umsetzung des MVC-Musters nichts im Wege. Weiterhin ist durch die verwendung von Java eine Plattformunabhängigkeit gewährleistet.

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