Entwurf und Implementierung eines Systems zur Administration von Benutzeranträgen für die Rechner-Infrastruktur eines Großforschungsinstituts

Diplomarbeit

zur Erlangung des Grades eines Diplom Informatikers (FH)

an der

Hochschule Bremerhaven
Fachbereich II

Studiengang Informatik/ Wirtschaftsinformatik

vorgelegt von: Tim Perkuhn
Matr.-Nr.: 21389

aus: Buchtstraße 38
27570 Bremerhaven
Tel.:(0471) 207685

Referent: Prof. Dr. Thomas Umland
Korreferent: Prof. Dr. Peter Kelb

Bremerhaven, 29.07.2004

Inhaltsverzeichnis

Abbildungsverzeichnis

1.0 Einleitung

1.1 Motivation

Das Rechenzentrum stellt den Mitarbeitern und Gastwissenschaftlern des Alfred-Wegener-Instituts für Polar- und Meeresforschung eine umfangreiche Rechnerinfrastruktur zur Verfügung. Um diese als Anwender in Anspruch nehmen zu können, ist ein Benutzerkonto nötig. Ein Benutzerkonto wird derzeit noch mittels eines klassischen Papierformulars beantragt. Dieses Benutzerkonto wird dann, unter Berücksichtigung der durch den künftigen Anwender beantragten Zugriffsrechten, von den Mitarbeitern des Rechenzentrums entsprechend eingerichtet. Zusätzlich werden die Informationen des Formulars für das Erzeugen eines Personeneintrages in einer Datenbank verwendet.

Die gestellte Aufgabe bestand darin, das bestehende Formular und den damit verbundenen Arbeitsablauf elektronisch abzubilden. Realisiert wurde die Aufgabe von mir durch eine Webapplikation. Hierbei fanden die Programmiersprache Java sowie die Technologien Java Servlet und JavaServer Pages Verwendung. Um das Model-View-Controller-Konzept umzusetzen, wurde das Framework Struts eingesetzt. Das Framework bietet für die View- und Controller-Schicht vorgefertigte Komponenten, die durch den Anwendungsentwickler durch entsprechende Programmierung um spezifisches Verhalten erweitert werden. Die Umsetzung der Model-Schicht wird durch das Framework nicht unterstützt und liegt somit vollständig in dem Verantwortungsbereich des Entwicklers.

1.2 Ziel der Arbeit

In der Erprobungsphase der Webapplikation ergab es sich, dass der Zugriff auf die Datenbank unter bestimmten Voraussetzungen einen Fehlerfall erzeugt. Während der Analyse der Fehlerquelle stellte sich heraus, dass die starke Kopplung zwischen der Controller- und Model-Schicht den erforderlichen Austausch der fehlerhaften Komponenten nicht zulässt.

Die vorliegende Arbeit zeigt den Entwurf und die Implementierung einer zusätzlichen Schicht, die den Zugriff auf das Model kapselt und dadurch die Kopplung zwischen den Schichten verringert.

1.3 Gliederung der Arbeit

Ferner erfolgt in Kapitel 2 eine Einführung in die von der Webapplikation verwendeten Technologien. In Kapitel 3 wird von mir ein Verfahren erläutert, das die in einer Struts-Anwendung implementierte Dialogreihenfolge analysiert und veranschaulicht. Das Kapitel 4 beschreibt die Anforderungen an die Anwendung. In Kapitel 5 wird die Entwicklung und Implementierung einer zusätzlichen Schicht dargestellt.

2.0 Einführung der verwendeten Technologien

2.1 Das HTTP-Protokoll

Das Hypertext Transfer Protocol (HTTP) ist ein zustandsloses, generisches Protokoll auf der Ebene der Verarbeitungsschicht. Eine Kommunikation mit dem HTTP wird durch einen Client initiiert, indem er eine HTTP-Anfrage an einen Server sendet. Eine HTTP-Anfrage besteht aus einer Anfragemethode, eine Uniform Resource Identifier (URI) [1], Header-Felder (header fields) und einen Rumpf der leer bleiben darf.[2] Der Server reagiert darauf mit dem Senden einer HTTP-Antwort. Die Kommunikation ist nach diesem Anfrage/Antwort-Zyklus abgeschlossen. Die Zustände von Client und Server ändern sich nicht; es gibt keinen Zusammenhang zwischen den vorherigen Anfrage/Antwort-Zyklen. Der HTTP State Management Mechanism erweitert die HTTP/1.0 Spezifikation um zwei Header, Cookie und Set-Cookie, die es ermöglichen Informationen über den Zustand zu übertragen. Der Server setzt in der Antwort ein Cookie und initiiert damit eine Session. Dem Client ist es freigestellt, ob er an der Session teilnimmt. Akzeptiert dieser den Cookie, können anschließende Anfrage/Antwort-Zyklen als zusammengehörig identifiziert werden.[3] Die Anwendungslogik von Client und Server ist für die Umsetzung der Zustände verantwortlich, das Protokoll als solches bleibt ohne Zustand.
Das Protokoll lässt sich durch das Anhängen von Zeichenketten an die URI und der Angabe von benannten Parametern um beliebige Befehlswörter erweitern. Auch hier liegt es in dem Verantwortungsbereich des Clients und Servers diese Befehlswörter zu interpretieren.

2.2 Die Auszeichnungssprache HTML

Die Auszeichnungssprache HyperText Markup Language (HTML) beschreibt Text und Interaktionselemente sowie dessen Formatierung und Positionierung. Die Struktur einer HTML-Seite wird durch Tags[4] beschrieben. Obligatorisch für jedes

[1] Siehe dazu [RFC2396]

[2] Vgl. [RFC2616, S. 8 und S. 16]

[3] Vgl. [RFC2109, S. 1]

[4] tag (engl.), dt.: Kennzeichen, Marke

HTML-Dokument ist das HTML-Tag, sowie die darin geschachtelten HEAD- und BODY-Tags.[5]

Für Webapplikationen sind die Link- und Formular-Formatierungselemente von Interesse. Ein Link stellt ein Verweis auf eine andere URI dar; Interaktionselement wie Schaltflächen oder Auswahllisten werden in dem Formular-Tag zusammengefasst. Initiiert durch den Benutzer, erzeugt der Webbrowser aus diesen Elementen eine HTTP-Anfrage.

2.3 Die Servlet- und JSP-Technologie

Ein Servlet ist eine in Java Bytecode compilierte Klasse um auf eine Anfrage eine Antwort mit dynamischem Inhalt zu generieren. Das Servlet wird von einem Servletcontainer verwaltet und ausgeführt. Der Servletontainer ist wiederum Bestandteil oder Erweiterung eines Webservers.[6]

Die Klasse `GenericServlet` ist an kein Protokoll gebunden und ermöglicht es beliebige Protokolle der Verarbeitungsschicht zu bearbeiten. Die von ihr abgeleitete Klasse `HttpServlet` ist auf die Verarbeitung des HTTP spezialisiert. Die Methoden `init` und `destroy` leiten das Initialisieren und Zerstören der Servletinstanz ein und werden während des Lebenszyklus eines Servlets genau einmal aufgerufen. Um die Anfragen eines Client anzunehmen dienen die zu überschreibenden Methoden `doGet`, `doPost`, `doPut` und `doDelete`. Für jede HTTP-Anfrage ruft der Servletcontainer eine dieser Methoden auf, die der Anwendung als Einstiegspunkt für die weitere Verarbeitung dient. Die Verarbeitung wird dann z.B. durch Delegation an gewöhnliche Klassen weitergeleitet. Als Parameter erhalten die Methoden Objekte der Klassen HttpRequest und HttpResponse, die das HTTP kapseln.

JavaServer Pages stellen eine Erweiterung der Servlet Technologie dar. Sie werden während der Laufzeit der Anwendung von der JSP-Engine zu Servlets übersetzt und stehen dann dem Servletcontainer zur Verfügung. JavaServer Pages und Servlets unterscheiden sich aus der Sicht des Entwicklers lediglich in der syntaktischen

[5] Vgl. [Mue01]

[6] Vgl. [ServletSpec, S. 11]

Notation.[7] Um dies darzustellen zeige ich hier den Quellcode von einem Servlet und einer JSP die eine identische Ausgabe erzeugen.

Ausgabe der Zahlen 0 bis 9 mittels einer JSP:

```
<html>
    <head><title>HelloJSP</title></head>
    <body>
           <%for (int i=0; i<10; i++){%>
              <%=i%><br/>
           <%};%>
    </body>
</html>
```

Ausgabe der Zahlen 0 bis 9 durch ein Servlet:

```
import java.io.IOException;
import java.io.PrintWriter;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

public class HelloServlet extends HttpServlet
{
    protected void doGet(HttpServletRequest req,
                         HttpServletResponse res)
           throws IOException, ServletException
    {
           res.setContentType("text/html");
           PrintWriter writer = res.getWriter();
           writer.println("<html>");
           writer.println("<head><title>HelloServlet</title><head>");
           writer.println("<body>");
           for (int i = 0; i < 10; i++)
           {
                  writer.println(i);
                  writer.println("<br/>");
           }
           writer.println("<body>");
           writer.println("</html>");
           writer.close();
    }
}
```

Wie das Beispiel zeigt, ist das Erzeugen der HTML-Ausgabe innerhalb des Servlets umständlich. Die Ausgabe wird in die Servlets fest einprogrammiert, eine Änderung der Ausgabe ist sehr aufwendig. Wohingegen die Schreibweise der HTML-Befehlswörter in der JSP dem Aufbau von HTML-Seiten gleichkommt, die

[7] Vgl. [Cav02, S. 5]

Programmiersprache Java aber in sogenannten Scriptlets gekapselt wird. Die Compilerfehler entstehen erst zur Laufzeit. Ein zusätzlicher Nachteil ist, dass der Programmcode in dieser Form nicht wiederverwendbar ist. JSP-Tags schwächen diesen Effekt ab, indem Klassen durch XML-Ausdrücke aufgerufen werden. Klasse und XML-Ausdrücke werden durch einen Tag Library Descriptor miteinander verbunden.

2.4 Das Model-View-Controller-Konzept

Die Organisation einer Anwendung in Schichten verringert die Komplexität. Durch die genau definierte Funktion jeder Schicht, wird es dem Entwickler erleichtert die Anwendung zu verstehen und zu verändern.

Vorbild für alle Systeme die eine Interaktion mit Benutzern führen ist das Model-View-Controller-Konzept (MVC-Konzept) bzw. die MVC-Triade. Dieses Konzept wurde mit der Programmiersprache Smalltalk-80 in den 90er Jahren eingeführt. Es definiert drei Schnittstellenkomponenten die das Erstellen von interaktiven Anwendungen erleichtert.[8] Die Aufgaben der MVC-Komponenten werden von Hans-Jürgen Hoffman folgendermaßen wiedergegeben:

„Model Mit diesem Begriff wird die Komponente einer interaktiven Anwendung bezeichnet, in der Leistungen (Funktionalität) der Anwendung zusammengefaßt sind.[...]

View Mit diesem Begriff wird die Komponente bezeichnet, in der die Bildschirmfläche für ein Fenster und die Aufteilung des Fensters in Teilflächen definiert ist. Ein View beschreibt die Position und die Größe einer Fensterfläche sowie die Beziehung der darin liegenden Teilflächen.[...]

Controller Die Controller-Komponente steuert die Interaktion des Benutzers mit der Anwendung. [...] Benutzeraktionen werden einerseits in anwendungsbezogene Aktionen durch das Senden von Botschaften an das Model, andererseits in darstellungsbezogene Aktionen durch das Senden von Botschaften an die View-Komponenten umgesetzt.“ [9]

[8] Vgl. [Mak90, S. 32]

[9] [Hoff87, S.99]

View und Controller wurden in Smalltalk-80 durch abstrakte Oberklassen umgesetzt, welche durch spezifische Klassen erweitert werden. Bereits in Smalltalk/V wurden diese durch die Klassen Pane und Dispatcher ersetzt. Die JFC/Swing-Komponenten der Programmiersprache Java kombinieren die Aufgabe von View und Controller in Delegate-Komponenten.[10] Für Java Webapplikation wird empfohlen den View durch JavaServer Pages und den Controller durch ein Servlet umzusetzen. Die Umsetzung des MVC-Konzeptes ist somit von der verwendeten Programmiersprache und der graphischen Oberfläche abhängig, allein die ursprüngliche Idee bleibt bestehen.

2.5 Frameworks

Ein Framework wird von Erich Gamma folgendermaßen definiert[11]:
Ein Framework besteht aus einer Menge von Klassen, die in Zusammenarbeit das Problem in einem speziellen Anwendungsbereich lösen. Es bestimmt die Architektur einer Anwendung auf oberster Ebene indem es die Struktur und die Zuständigkeiten von Klassen und den Kontrollfluss von Objekten festlegt. Ein herkömmliches Programm nutzt die Funktionen von Klassenbibliotheken, dabei bestimmt es die Reihenfolge der Aufrufe. Bei einem Framework schreibt der Entwickler Programme die von dem Framework aufgerufen werden. Es tritt eine Umkehrung der Steuerung[12] auf. Die von dem Framework vorgegebenen Konventionen müssen von den selbstentwickelten Programmen eingehalten werden um die Zusammenarbeit zu gewährleisten.
Frameworks erfordern einen Einarbeitungsaufwand, der im voraus erbracht werden muss. Hierfür wird der Entwickler von grundlegenden Entwurfsentscheidungen befreit und kann direkt von den Erfahrungen der Frameworkentwickler profitieren. Eine auf diese Weise entstandene Anwendung ist für einen anderen Entwickler der das Framework kennt leichter zu verstehen als eine beliebige Eigenentwicklung.

---

[10] Vgl. [Mag99]

[11] Vgl. [Gam96, S. 37]

[12] inversion of control (engl.)

2.6 Struts

Struts ist ein Open Source Framework der Apache Software Foundation für das Erstellen von Webapplikationen mittels der Programmiersprache Java. Struts bildet eine stabile und wiederverwendbare Basis der Komponenten, die sich aus den Gemeinsamkeiten vieler Webapplikationen gebildet hat. Die Architektur von Struts unterstützt eine Variante des MVC-Konzeptes, des von Sun Microsystems vorgestellten Model-2-Ansatzes.[13]

Die View-Schicht wird durch JavaServer Pages (JSP) gebildet. Eine Reihe von vorgefertigten Struts-Tags vereinfachen die Integration von dynamischen Daten.

Die Controller-Schicht wird hauptsächlich durch zwei Komponenten realisiert:

- Eine Instanz der Klasse `ActionServlet`, die alle HTTP-Anfragen des Clients entgegennimmt und die Verarbeitung an eine Aktion weitergibt.
- Aktionen[14], die für die Bearbeitung genau einer spezifischen HTTP-Anfrage ausgelegt sind. Sie werden von der Klasse `Action` abgeleitet und müssen die Methode `execute` überschreiben, die von dem Framework aufgerufen wird. Aktionen verarbeiten die Daten der View-Schicht und rufen das Model auf. Jede Aktion wird durch die Weiterleitung an ein View (in manchen Fällen an eine Aktion) beendet.

Der Begriff ActionForm bezeichnet ein von der Klasse `ActionForm` abgeleitetes JavaBean. Sie nehmen die Daten der HTML-Formulare auf und dienen als Datenlieferant für die Action-Klassen. Sie stellen eine Schnittstelle zwischen View- und Controller-Schicht dar. Die Member-Variablen sind durch getXXX- und setXXX-Methoden gekapselt. Die Signatur dieser Methoden muss mit den Parameternamen des HTML-Formulares übereinstimmen um von dem Framework mit Daten befüllt zu werden. Durch Überschreiben der Methode `validate` ist es möglich eine Vorprüfung der Daten auf Existenz und Typ durchzuführen. Bei einem Fehlschlag wird ein `ActionError` Objekt generiert. Der Inhalt dieser Fehlermeldung wird per Zugriffsschlüssel aus einer externen Datei geholt und kann durch einen Struts-Tag in einer JSP dargestellt werden.

---

[13] Vgl. [Struts01]

[14] Synonym zu Aktion wird der Begriff Action-Klasse (engl.: action class) verwendet

3.0 Darstellung einer Struts-Anwendung als Zustandsautomat

In diesem Kapitel wird durch den Verfasser gezeigt, wie sich auf anschauliche Weise die View- und Controller-Komponenten einer Struts-Anwendung als endlicher Zustandsautomat darstellen läßt. Diese Notation erfolgt in der UML[15].

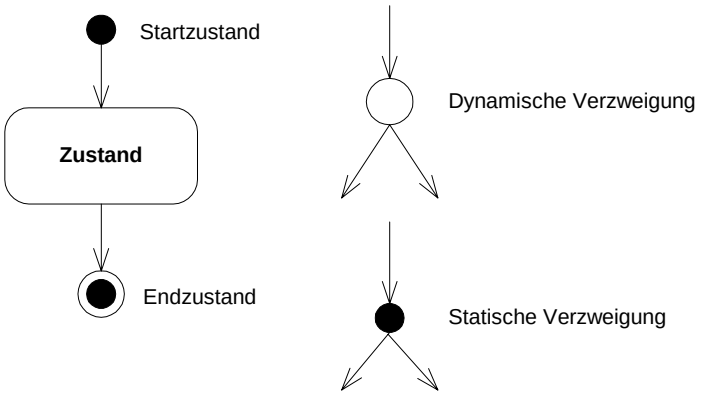


Abb. 1: Legende der Notation für Zustandsautomaten in UML

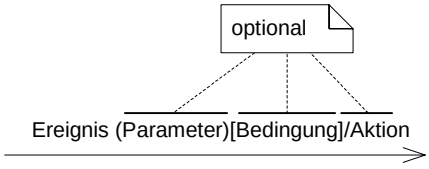


Abb. 2: Übergang oder Transition

Die Ausdrücke und Anweisungen, aus denen sich der Zustandsautomat zusammensetzt, sind auf mehrere Dateien der View- und Controller-Schicht verteilt. Die Bestandteile dieser Anwendung, welche hierbei in Betracht gezogen werden, sind die Konfigurationsdateien `struts-config.xml` und `web.xml` sowie alle verwendeten JSP, HTML-Seiten, Action-Klassen und ActionForms[16].

Als Beispiel wurde von mir die Anwendung aus dem Tutorial „Struttin' with Struts – Lesson II" von Rick Reumann verwendet[17]. Dargestellt werden von mir nur die Bestandteile, welche für das Erzeugen des Zustandsautomaten erforderlich sind, d.h.

---

[15] Siehe dazu [UMLSpec, S. 3-136]

[16] Die Bezeichnung für JavaBeans die von der Klasse org.apache.struts.action.ActionForm ableiten.

[17] Siehe dazu [Reu]

sämtliche Klassen der Geschäfts- und Datenzugriffsschicht (sprich die Model-Schicht) werden ausgelassen.

Zusammenfassung der Funktionalität:

Name, Alter und Abteilung eines Angestellten werden von der Anwendung erfasst und in einer Datenbank gespeichert. Die Seite `index.jsp` begrüßt den Anwender und leitet zur Seite `employeeForm.jsp` weiter, die ein Eingabeformular präsentiert. Nach Eingabe der Daten informiert die Seite `confirmation.jsp` den Anwender über den korrekten Empfang der Daten, wohingegen die Seite `error.jsp` dem Anwender signalisiert dass ein Fehler im Zusammenhang mit der Datenbank aufgetreten ist.

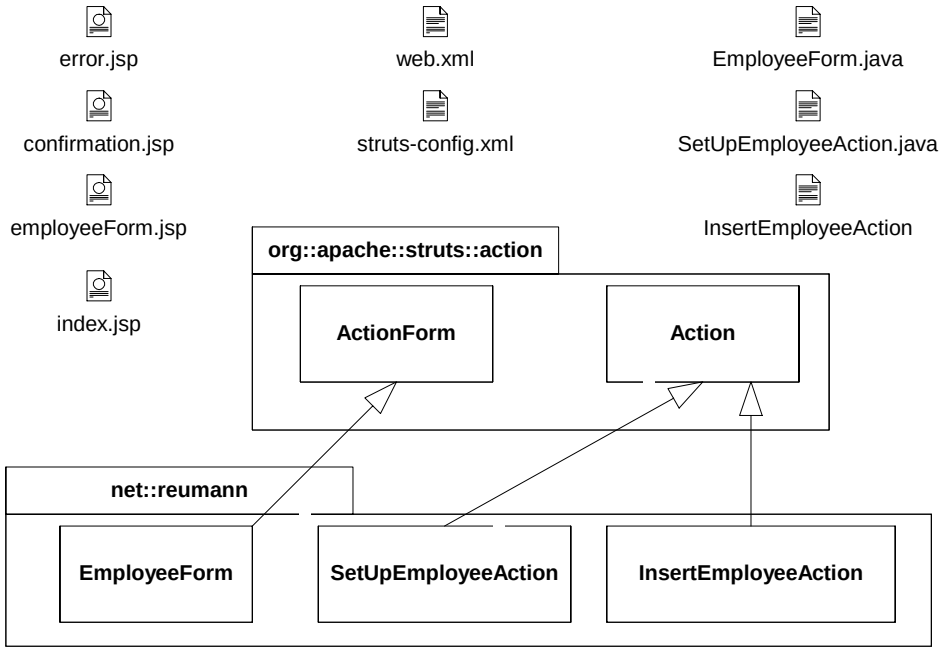


Abb. 3: Bestandteile des Beispiels

3.1 Zustand und ausgehende Transition

JSPs stellen die Zustände dar, die der Automat einnehmen kann. Die Tags `html:link` und `html:page` werden in dem Browser als Schaltflächen dargestellt und erzeugen durch Betätigung des Anwenders Ereignisse, die einen Übergang in einen Folgezustand auslösen.

index.jsp:

```
<html:link page="/do/setUpEmployeeForm">Add An Employee</html:link>
```

employeeForm.jsp:

```
<html:form action="insertEmployee" focus="name">
    <!-- [...] -->
</html:form>
```

Die Werte der Attribute `page` und `action` geben den Ereignisnamen in Form einer URI an.

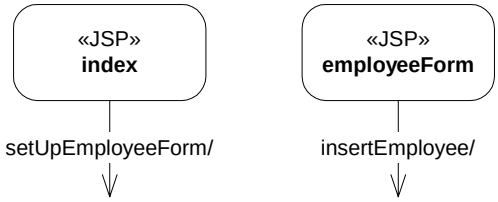


Abb. 4: JSP als Zustand mit Ereignis und ausgehender Transition

Die Dateien `confirmation.jsp` und `error.jsp` enthalten weder `html:link` noch `html:page` Tags. Die Seiten stellen deshalb keine Schaltelemente zur Verfügung die ein Ereignis auslösen könnten. Die Zustände werden mit einem Übergang ohne auslösendes Ereignis dargestellt, der in einen Endzustand führt.

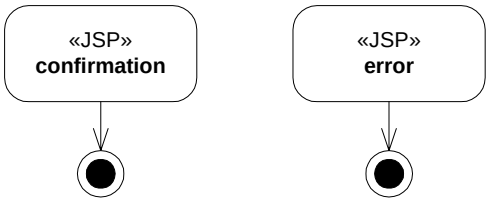


Abb. 5: JSP als Zustand mit Übergang in den Endszustand

3.2 Startzustand

Der Startzustand einer Anwendung lässt sich aus der `web.xml` ermitteln. Der Übergang zum Folgezustand erfolgt ohne auslösendes Ereignis.

web.xml

```
  <welcome-file-list>
    <welcome-file>index.jsp</welcome-file>
  </welcome-file-list>
```

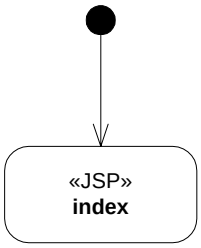


Abb. 6: Startzustand mündet in JSP

3.3 Eingehende Transitionen, Aktionen und Parameter

Die Datei `struts-config.xml` bildet die zentrale Konfigurationseinheit für eine Struts-Anwendung. Sie verknüpft die von den JSPs ausgehenden Ereignisse mit Aktionen und Parametern.

Parameter werden durch das Element `form-bean` deklariert. Das Attribut `type` gibt die Datenklasse an, die von `org.apache.struts.action.ActionForm` abgeleitet wird. Innerhalb der Konfiguration wird der Parameter durch den Wert des `name` Attributes angesprochen.

Definition eines Parameters in der struts-config.xml:

```
<form-beans>
    <form-bean name="employeeForm" type="net.reumann.EmployeeForm"/>
</form-beans>
```

Das `action` Element bezeichnet eine Transition, die mit einer Aktion verknüpft ist. Die Transition führt in eine dynamische Verzweigung mit einer oder mehreren bedingten ausgehenden Transitionen. Die Auswertung der Bedingung findet in der Aktion statt.

Definition von Aktion und dynamischen Übergang in struts-config.xml:

```
<action-mappings>
    <action path="/insertEmployee"
            type="net.reumann.InsertEmployeeAction"
            name="employeeForm"
            scope="request"
            validate="true"
            input="/employeeForm.jsp"
            <forward
                    name="success"
                    path="/confirmation.jsp"/>
        </action>
```

Die weiterführenden Transitionen werden in dem `action` Element durch `forward` Elemente deklariert. Ein `forward` Element kann in einen Zustand oder einen weiteren Übergang führen. Das `name` Attribute in dem `action` Element referenziert den Parameter, der an die Aktion übergeben wird. Das Attribut `type` deklariert die auszuführende Aktion als voll qualifizierten Klassennamen. Sie erweitert die Klasse `org.apache.struts.action.Action`. In der `execute` Methode der Action-Klasse wird das spezifische Verhalten der Aktion implementiert. Sie wird von dem Framework aufgerufen. Als Rückgabe wird eine Transition in Form eines `org.apache.struts.action.ActionForward` Objektes geliefert. Um festzustellen welche Bedingung über die zurückgegebene Transition entscheidet, wird der Quellcode der `execute` Methode herangezogen.

Bedingung des nächsten Überganges in InsertEmployeeAction.java:

```
public final class InsertEmployeeAction extends Action {
    public ActionForward execute(ActionMapping mapping,
                                 ActionForm form,
                                 HttpServletRequest request,
                                 HttpServletResponse response)
        throws Exception {
         //Objekt Instanzierungen.
        try {
            //Zugriff auf die Datenbank.
            return (mapping.findForward("success"));
        }
        catch( DatabaseException de ) {
            //Erzeugen von Fehlermeldungen.
            return (mapping.findForward("error"));
        }
    }
}
```

Verläuft der Zugriff auf die Datenbank erfolgreich, wird die Transition mit dem Namen `success` zurückgegeben, welche in den Zustand `confirmation.jsp` übergeht. Tritt hingegen eine `DatabaseException` auf, wird die Transition `error` zurückgegeben. Die Transition `error` wird in diesem Fall nicht innerhalb des `action` Elements, sondern in dem Element `global-forwards` deklariert.

Eine globale Weiterleitung in struts-config.xml:

```
<global-forwards>
    <forward name="error" path="/error.jsp"/>
</global-forwards>
```

Seit der Struts-Version 1.1 ist es möglich Ausnahmen deklarativ[18] zu behandeln.

[18] Siehe dazu [CAV02, S.44ff]

Deklarative Ausnahmebehandlung:

```
<global-exceptions>
    <exception
          key="FehlermeldungFuerDatabaseException"
          type="net.reumann.DatabaseException"
          path="/error.jsp"></exception>
</global-exceptions>
```

Alle in der Action-Klasse nicht abgefangenen Ausnahmen werden an das Framework weitergereicht, so dass anhand des `type` Attribute die Transition für die Ausnahme bestimmt wird. Der Try-Catch-Block in der `execute` Methode fällt dadurch weg. Eine Ausnahme die weder programmatisch noch deklarativ abgefangen wird führt zu einem Abbruch der Programmausführung.

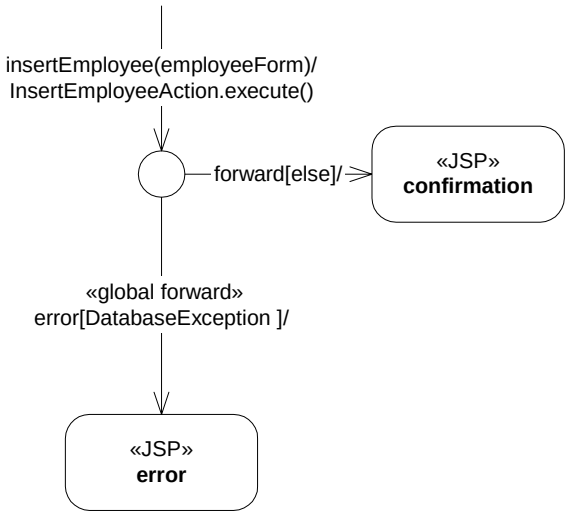


Abb. 7: Dynamische Verzweigung durch die Aktion InsertEmployeeAction

Die Erläuterung für die Attribute `validate` und `input` ist noch offen: Unterklassen von `org.apache.struts.action.ActionForm` können die Methode `validate` überschreiben, um eine Kontrolle der eingehenden Daten durchzuführen. Die Art und der Umfang der Validierung ist spezifisch zum Anwendungsfall, möglich ist z.B. das prüfen auf die Existenz und den Typen einer Variable. Tritt ein Fehler auf, wird eine Instanz der Klasse `org.apache.struts.action.ActionErrors` erzeugt, die eine Fehlermeldung für den Anwender aufnimmt. Enthält das Attribut `validate` den Wert `true`, wird vor dem Ausführen der Aktion von dem Framework die `validate` Methode aufgerufen. Das Attribute `input` gibt die JSP an, die im Fehlerfall angezeigt wird. Ihr wird das `ActionError` Objekt übergeben, dessen Inhalt mittels eines Struts-Tags in der JSP angezeigt werden kann.

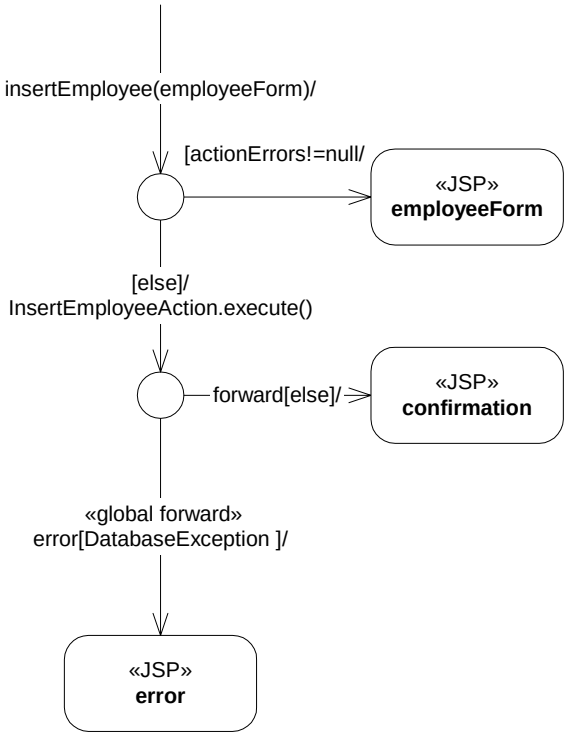


Abb. 8: Validieren des Parameters vor Ausführen der Aktion

Die zweite Aktivität gestaltet sich simpler. Es erfolgt keine Validierung der eingehenden `ActionForm`. In der Action-Klasse befindet sich keine Bedingung, die bestimmt welcher Übergang gewählt wird, noch wird eine Methode aufgerufen, die eine Ausnahme werfen könnte.

Definition von Aktion und dynamischen Übergang in struts-config.xml:

```
    <action path="/setUpEmployeeForm"
            type="net.reumann.SetUpEmployeeAction"
            name="employeeForm"
            scope="request"
            validate="false">
            <forward
                    name="continue"
                    path="/employeeForm.jsp"/>
        </action>
    </action-mappings>
```

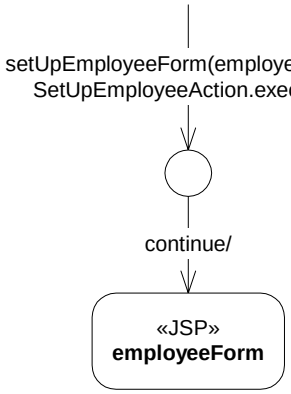


Abb. 9: Aktion ohne Validierung der ActionForm

Nachdem alle Bestandteile identifiziert wurden, besteht der letzte Schritt darin diese zusammenzufügen.

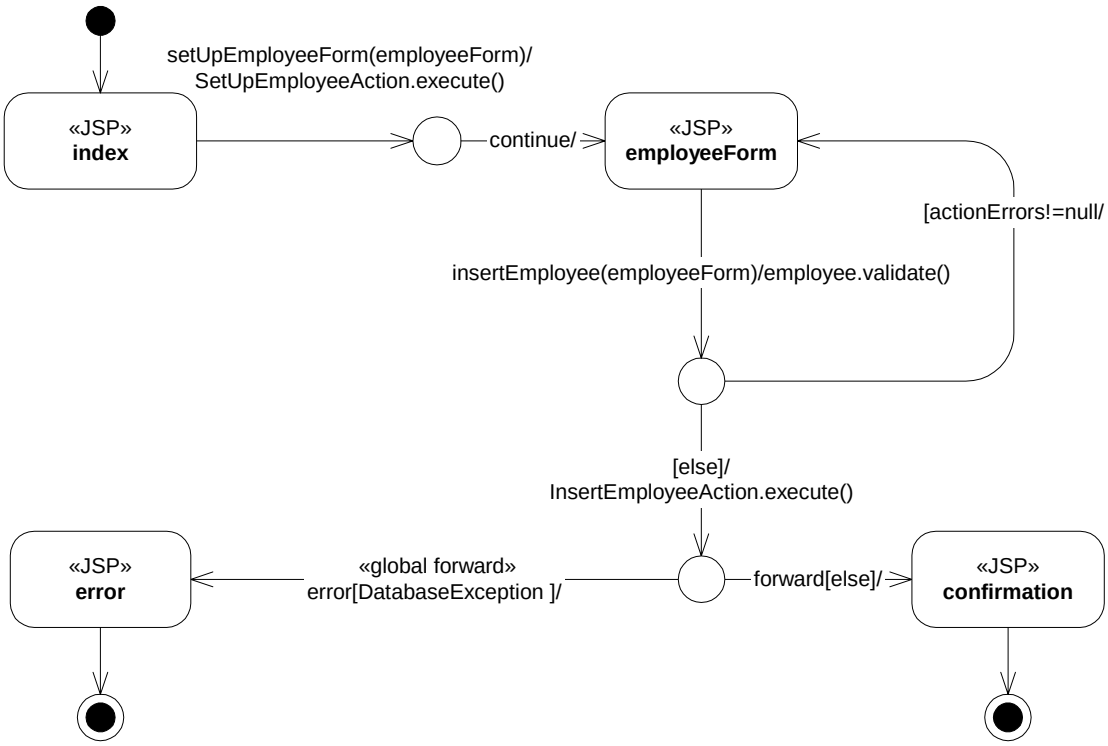


Abb. 10: Der komplette Zustandsautomat

4.0 Beschreibung der Anforderungen

Der Benutzerantrag[19] ist ein klassisches Papierformular, das von dem Antragsteller auszufüllen ist. Es erfasst u.a. Name, Niederlassung, Gebäude und Raumnummer des Beschäftigten, sowie dessen Zugehörigkeit zu einer Organisationseinheit. Ferner die vom Antragsteller gewünschten Zugriffsberechtigungen. Gleichzeitig beinhaltet das Formular eine Verpflichtung (Policy), die die Regeln bezüglich der Nutzung der Rechnerinfrastruktur festlegt.

Der Administrator der hierarchischen Datenbank (Verzeichnis-Administrator) erzeugt anhand der Angaben des Benutzerantrages einen Personeneintrag in der Datenbank. Der Personeneintrag enthält sämtliche Daten des Benutzerantrages und wird u.a. durch Emailadresse und Mailserver ergänzt. Die Personeneinträge werden von den Mail-Servern des AWI benutzt und stellen gleichzeitig ein elektronisches Telefonbuch dar.

Die Administratoren der Dienste für Email, Unix und Windows erstellen die Zugriffsberechtigungen und füllen ein Kontoinformationsblatt[19] aus. Es enthält Benutzername, vorläufiges Passwort und eventuelle Konfigurationsparameter, die der Antragsteller benötigt, um die Dienste in Anspruch zu nehmen.

[19] Siehe dazu Anhang 1

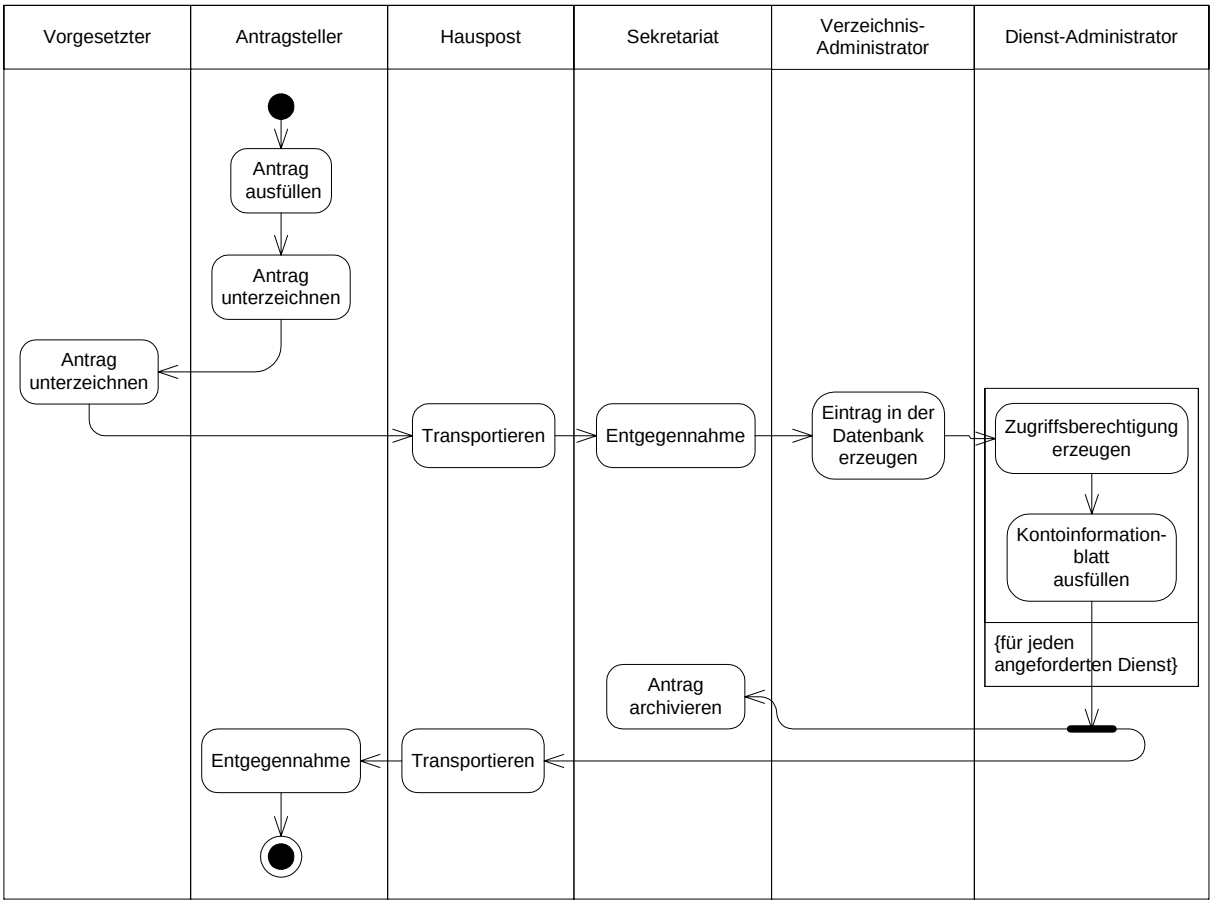


Abb. 11: Arbeitsablauf zur Bearbeitung eines Benutzerantrages

Der Arbeitsablauf ist in einer zeitlichen Abfolge verschiedener Tätigkeiten gegliedert. Der erste Vorgang besteht in dem Ausfüllen des Antrages seitens des Antragstellers und die Anerkennung der Policy durch Unterschrift. Ebenso muss der Antrag von einem unterschriftsberechtigten Vorgesetzten gegengezeichnet werden, wodurch bestätigt wird, dass der Antragsteller dem AWI angehört. Der ausgefüllte Antrag wird durch die Hauspost zum Sekretariat des Rechenzentrums transportiert und an den Administrator der Verzeichnisdatenbank weitergeleitet. Dieser überprüft den Antrag und erzeugt einen Personeneintrag in die Datenbank. Als Umlauf erhalten die Administratoren der speziellen Dienste den Antrag und richten Zugriffsberechtigungen ein. Auf einem Kontoinformationsblatt werden Benutzername und vorläufiges Passwort eingetragen und dem Antragsteller per Hauspost zugesandt. Letzter Schritt ist die Archivierung des Antrages durch das Sekretariat.

4.1 Varianten des Benutzerantrages

Es gibt unterschiedliche Situationen die das Ausfüllen eines Benutzerantrages erfordern.

- Erstellen – Der Antragsteller verfügt noch nicht über ein Benutzerkonto und möchte die Dienste des RZ in Anspruch nehmen.
- Verlängern – Der Antragsteller verfügt über ein Benutzerkonto. Die Laufzeit ist bereits abgelaufen oder wird in Kürze beendet sein.
- Ändern – Der Antragsteller verfügt über ein Benutzerkonto. Dieser will Änderungen vornehmen, die eine Unterschrift des Vorgesetzten benötigen.
- Löschen – Der Antragsteller verfügt über ein Benutzerkonto. Dieses soll vor Beenden der normalen Laufzeit gelöscht werden.

4.2 Personen und Rollen

Name	Antragsteller
Beschreibung	Er unterschreibt den Benutzerantrag und erkennt damit die Bedingungen zur Nutzung der Rechnerinfrastruktur an. Er bekommt ein Kontoinformationsblatt.

Name	Vorgesetzter
Beschreibung	Der Vorgesetzte erhält einen korrekt ausgefüllten Benutzerantrag, der vom Antragsteller unterschrieben ist und von ihm unterschrieben wird.

Name	Verzeichnis-Administrator
Beschreibung	Der Verzeichnis-Administrator erwartet ein korrekt ausgefüllten Benutzerantrag, der von Antragsteller und Vorgesetzten unterschrieben wurde. Er erstellt einen Personeneintrag im Personenverzeichnis.

Name	Dienst-Administrator
Beschreibung	Der Dienst-Administrator bekommt ein Kontoinformationsblatt. Er richtet die vom Antragsteller gewünschten Dienste ein. Er trägt die Zugangsinformationen auf dem Kontoinformationsblatt ein und reicht das Blatt an den nächsten Dienst-Administrator oder über die Hauspost an den Antragsteller weiter.

Name	Hauspost
Beschreibung	Die Hauspost transportiert Benutzerantrag und Kontoinformationsblatt.

Name	Sekretariat
Beschreibung	Das Sekretariat leitet den Benutzerantrag an den Verzeichnis-Administrator weiter und archiviert den Benutzerantrag als abschließenden Vorgang.

Tab. 1: Personen und Rollen

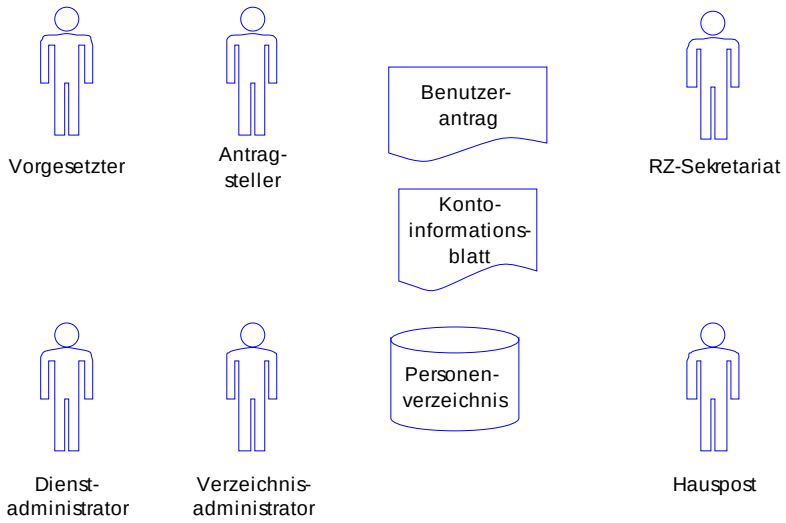


Abb. 12: An dem Arbeitsablauf beteiligte Rollen und Dokumente

4.3 Daten und Dokumente

Name	Benutzerantrag
Beschreibung	Der Benutzerantrag enthält personenbezogene Daten und Angaben über die geforderten Dienste. Er wird vom Antragsteller und Vorgesetzten unterschrieben. Der Benutzerantrag stellt ein rechtsverbindliches Dokument dar. Der Antragsteller verpflichtet sich dazu die Auflagen der Policy zu beachten.

Name	Personeneintrag
Beschreibung	Der Personeneintrag enthält Angaben über die Person, Standort, Organisationszugehörigkeit, Emailadresse, Mailhost etc. Er wird in einer Verzeichnisdatenbank gespeichert.

Name	Kontoinformationsblatt
Beschreibung	Das Kontoinformationsblatt enthält Angaben über Dienste, Benutzername und vorläufige Passwörter sowie Konfigurationsparameter. Er wird von dem Dienst-Administrator ausgefüllt und dem Antragsteller übermittelt.

Tab. 2: Daten und Dokumente

4.4 Verwendung des elektronischen Personeneintrages

Personeneinträge werden in einer hierarchischen Datenbank gespeichert. Hiefür wird ein Verzeichnis-Server benutzt, der das Lightweigt Directory Access Protocol (LDAP) verwendet. Die Verzeichnisdatenbank beinhaltet momentan ca. 1200 Personeneinträge.

Das Email-System des AWI-Rechenzentrums arbeitet eng mit der Verzeichnisdatenbank zusammen. Der externe Mail-Server vergleicht die Empfängeradresse der eingehenden Email mit den Adressen aus der Verzeichnisdatenbank und entscheidet ob die Email angenommen wird. Nachdem eine Email angenommen wurde, gibt der Personeneintrag Auskunft an welchen internen Mail-Server die Nachricht weitergeleitet werden soll. Der interne Mail-Server verwendet die Einträge um den Empfänger einer Email zu authentifizieren.

Ein Anwendungsbereich der weniger spezifisch ist, dafür aber einen umfassenden Benutzerkreis anspricht, besteht in der Verwendung der Verzeichnisdatenbank als elektronisches Telefonbuch. Die im Auftrag des AWI Institutes entwickelte Software eDirectory ermöglicht die Suche und Darstellung von Personeneinträgen mittels eines Webbrowsers. Der Mozilla-Mail-Client bietet die Möglichkeit eine Verzeichnisdatenbank einzubinden. Personeneinträge werden dadurch in das Adressbuch integriert und eine automatische Vervollständigung bei Eingabe einer Email-Adresse ermöglicht.

4.5 Beurteilung der Situation

Nachteile:

- Da der Benutzerantrag und das Kontoinformationsblatt mit der Hauspost transportiert werden, kann ein Vorgang mehrere Tage dauern. Befindet sich der Antrag an einem entfernten Standort, wie etwa Berlin oder Helgoland, werden die Dokumente per Fax übermittelt.
- Ein Benutzerantrag wird handschriftlich ausgefüllt. Einige Handschriften lassen sich nur schwer oder überhaupt nicht entziffern. Eine zeitintensive telefonische Rückfrage ist erforderlich.
- Während die Daten von dem Benutzerantrag in die EDV eingegeben werden, können Übertragungsfehler entstehen.

- Das Erzeugen eines Personeneintrages ist für den Verzeichnis-Administrator zeitaufwendig.

Vorteile:

- Der Arbeitsablauf ist, mit Ausnahme der neu angestellten Mitarbeiter allen Beschäftigten vertraut.

4.6 Anforderungen an das System

Aufgaben, die das System erfüllen soll:

- Die zeitliche Verzögerung durch den Postweg soll reduziert werden.
- Der Zeitaufwand der durch Abtippen, Nachfragen und Korrigieren entsteht soll reduziert werden.
- Übertragungsfehler sollen durch direkte Eingabe der Daten seitens des Antragstellers weitestgehend vermieden werden.

Abgrenzung der Anforderungen:

- Es wird keine digitale Signatur eingesetzt. Der Benutzerantrag bleibt daher als Papierdokument bestehen.
- Die Administratoren erstellen die Zugriffsberechtigungen der gewünschten Dienste weiterhin manuell.

4.7 Darstellung der Applikation

Um einen Überblick über die Funktionalität der Anwendung zu vermitteln, werde ich die Applikation aus der Sicht des Anwenders beschreiben. In der Abbildung Abb. 13: werden alle JSP und Übergänge schematisch dargestellt. Anschließend werden die Aufgaben der Zustände in tabellarischer Form beschrieben. Bildschirmfotos der in einem Browser dargestellten JSP befinden sich im Anhang.

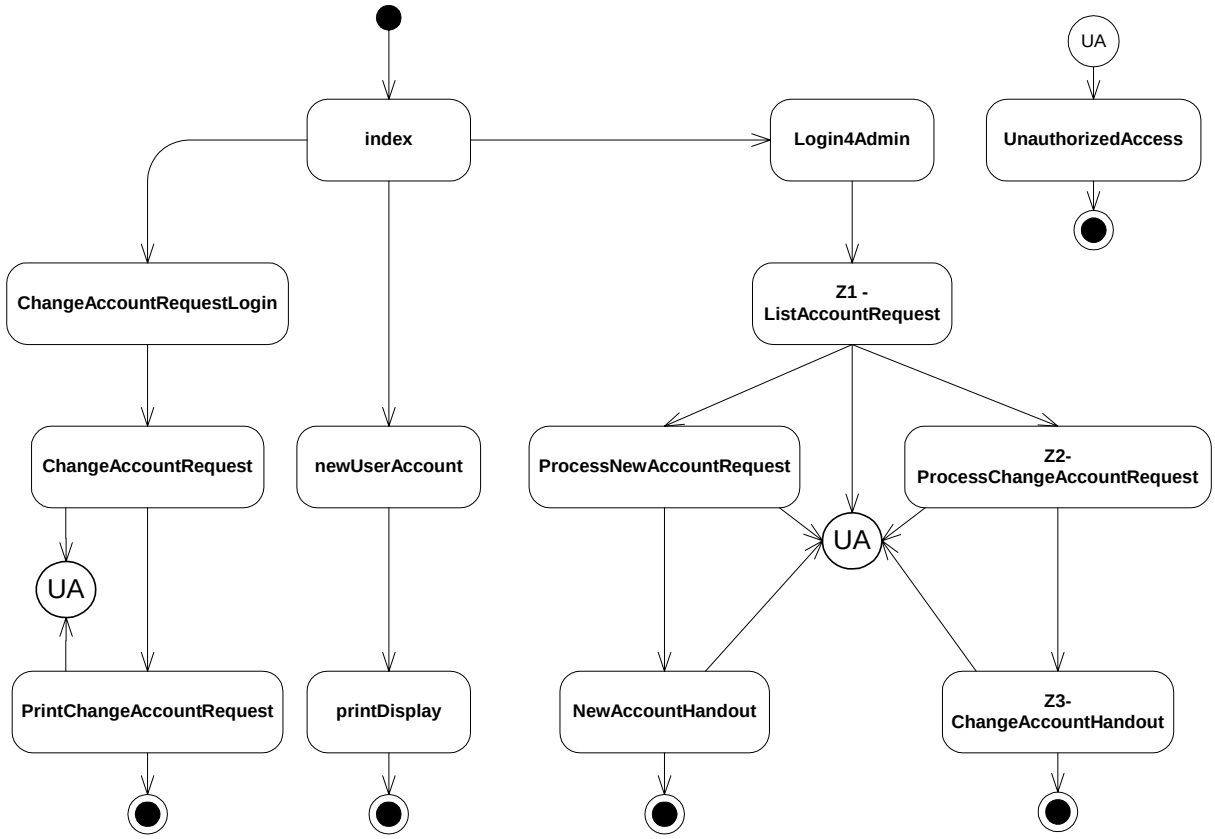


Abb. 13: Alle JSPs der Anwendung im Überblick

Zustand	index
Benutzergruppe	Alle Benutzergruppen
Beschreibung	Die Startseite der Anwendung stellt Links zu den verschiedenen Funktionen der Anwendung dar und gibt dem Antragsteller Hinweise zur Bedienung.

Zustand	newUserAccount
Benutzergruppe	Antragsteller ohne Benutzerkonto
Beschreibung	Diese Seite stellt ein Formular dar. Sie wird von dem Antragsteller benutzt, der noch nicht über ein Benutzerkonto verfügt. Dies werden in der Regel Personen sein, die erst seit kurzem für das Institut tätig sind.

Zustand	printDisplay
Benutzergruppe	Antragsteller ohne Benutzerkonto
Beschreibung	Die in `newUserAccount` eingegebenen Daten, sowie die Richtlinie über den rechtmäßigen Gebrauch des Kontos werden als Druckansicht dargestellt.

Zustand	ChangeAccountRequestLogin
Benutzergruppe	Antragsteller mit Benutzerkonto
Beschreibung	Eine Loginseite die Benutzerkennung und Passwort abfragt.

Zustand	ChangeAccountRequest
Benutzergruppe	Antragsteller mit Benutzerkonto
Beschreibung	Ein Formular, das es einem Antragsteller ermöglicht Änderungen an seinem Benutzerkonto zu beantragen.

Zustand	PrintChangeAccountRequest
Benutzergruppe	Antragsteller mit Benutzerkonto
Beschreibung	Die in `ChangeAccountRequest` geänderten Daten, sowie die Richtlinie über den rechtmäßigen Gebrauch des Kontos, werden als Druckansicht dargestellt.

Zustand	Login4Admin
Benutzergruppe	Administrator
Beschreibung	Eine Loginseite die Benutzerkennung und Passwort abfragt.

Zustand	ListAccountRequest
Benutzergruppe	Administrator
Beschreibung	Es wird eine Liste der vorliegenden Anträge dargestellt, wobei für jeden Antrag die Möglichkeit besteht diesen zu löschen oder zu bearbeiten.

Zustand	ProcessNewAccountRequest und ProcessChangeAccountRequest
Benutzergruppe	Administrator
Beschreibung	Ein Formular, das den durch technische und organisatorische Details ergänzten Antrag darstellt. Der Administrator überprüft und korrigiert die automatisch erzeugten Daten.

Zustand	PrintNewAccountHandout und PrintChangeAccountHandout
Benutzergruppe	Administrator
Beschreibung	Die Druckansicht eines Handzettels, der als Umlauf an die Dienst-Administratoren gesandt wird. Der Zettel beinhaltet die Informationen über den Antragsteller, die benötigt werden um einen Dienst einzurichten.

Zustand	UnauthorizedAccess
Benutzergruppe	Alle Benutzergruppen
Beschreibung	Eine Fehlermeldung die angezeigt wird, falls ein Anwender Zugriff auf einen Bereich ausüben will für den er keine Berechtigung besitzt.

Tab. 3: Beschreibung der JSP

5.0 Entwicklung der Domänensteuerungsschicht

5.1 Problembeschreibung

Die Aufgaben der Action-Klassen sollen sich darauf beschränken die dynamischen Inhalte für die JSPs bereitzustellen und die Benutzereingaben entgegenzunehmen und weiterzuleiten. Beide Aufgaben werden in Zusammenarbeit mit dem Model durchgeführt.

Der Aufruf des Models ist in der bestehenden Anwendung zu umständlich.

Um den dynamischen Inhalt für den Zustand Z2 `ProccessChangeAccountRequest` zu erstellen sind elf Aufrufe in der Action-Klasse nötig:

1. Ein Objekt der Klasse `AccountRequestRepositoryDAO`[20] aus dem `ServletContext` abrufen; ein Objekt das die Benutzeranträge lesen kann.
2. In der vorangegangenen Aktion wurde ein Benutzerantrag ausgewählt, um ihn zu bearbeiten. Um die Bearbeitung fortzusetzen wird die Identifikation des Antrages aus der Session gelesen.
3. In der Zeit zwischen der Darstellung der Liste der Benutzeranträge und dem Auswählen eines Antrages zur Bearbeitung, kann ein anderer Administrator den Antrag bereits verändert haben. Deshalb wird an das Objekt der Klasse `AccountRequestRepositoryDAO` die Anfrage gestellt ob für die gegebene Identifikation ein Antrag existiert.
4. Der Benutzerantrag wird durch das Objekt der Klasse `AccountRequestRepositoryDAO` gelesen.
5. Es wird geprüft ob der Antrag gelesen wurde.
6. Das Objekt für die Transformationsregeln ist zu initialisieren.
7. Die Transformation wird durchgeführt.
8. Die transformierten Daten werden in eine ActionForm kopiert.
9. Den unveränderten Personeneintrag aus der Verzeichnisdatenbank lesen.
10. Die Daten in eine JavaBean kopieren.
11. Daten für die graphischen Interaktionselemente vorbereiten.

[20] data access object (DAO) (engl.), dt.: Datenzugriffsobjekt

Um die Operationen durchführen zu können ist die Aktion von vierzehn Klassen abhängig. Bei den übrigen Aktionen des administrativen Bereiches verhält es sich ähnlich. Abstrahiert man die Zusammenhänge der Klassen auf die Schichtenebene stellt sich die Abhängigkeit wie in Abbildung 14 dar.

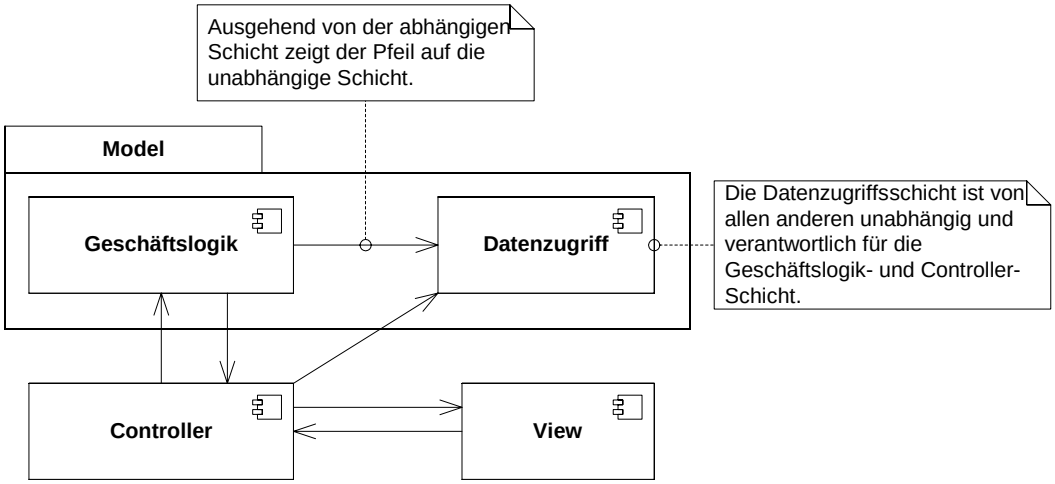


Abb. 14: Abhängigkeiten der Schichten

Die Action-Klassen sollen von den aufwendigen und komplizierten Aufrufen an das Model befreit werden. Treten Änderungen in dem Model auf, so sollen die Action-Klassen davon unbetroffen bleiben. Um die Abhängigkeit der Controller-Schicht von der Geschäftslogik- und Datenzugriffsschicht zu lösen wird eine zusätzliche Schicht eingefügt.

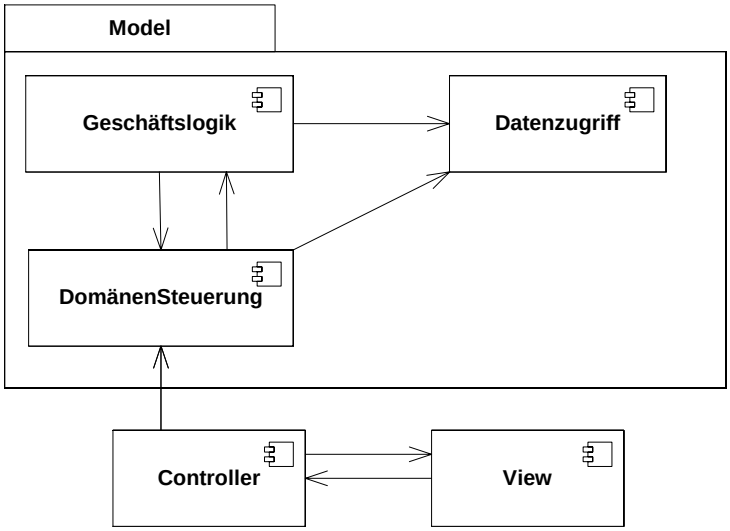


Abb. 15: Abhängigkeiten mit einer zusätzlich eingefügten Schicht

Die Aufrufe der Controller-Schicht an das Model werden auf die Domänensteuerungsschicht verschoben. Die Controller-Schicht ist dadurch nur noch von zwei Schichten abhängig.

5.2 Struktureller Aufbau

Im folgenden zeige ich wie die Domänensteuerung entwickelt wird. Hierbei werden die Zustände

Z1 (`ListAccountRequest`),

Z2 (`ProcessChangeAccountRequest`) und

Z3 (`ChangeAccountHandout`)

betrachtet. Die Zustände `ProcessNewAccountRequest` und `NewAccountRequest` gehören ebenso in den Einflussbereich der Domänensteuerungsschicht. Sie verhalten sich ähnlich zu Z2 und Z3, deshalb werden sie wegen der Übersichtlichkeit des Beispiels ausgelassen. Die restlichen Zustände der Anwendung erfordern nur unkomplizierte Operationen mit dem Model, die nicht in einer zusätzlichen Schicht gekapselt werden müssen.

Kern der Domänensteuerungsschicht ist die Klasse DomainStateController. Die Action-Klassen sprechen das Model ausschließlich über die Instanz dieser Klasse an. Die Klasse wird als Zustandsautomat entworfen, die den Zugriff auf das Model kontrolliert. Die `DomainStateController`-Instanz hält Referenzen auf die Klassen `AccountManager` und `EntryRule` der Geschäftslogikschicht. Diese wiederum benutzen Instanzen der Klasse `AccountDAO` und `EntryDAO` der Datenzugriffsschicht. Tatsächlich werden mehr als vier Klassen benötigt um die Aufgaben der Schichten zu erfüllen. Um die Domänenzugriffsschicht zu entwickeln ist eine detailliertere Darstellung nicht erforderlich.

5.2.1 Referenzierte Klassen

Die Klasse `AccountDAO` liest, schreibt und löscht Benutzeranträge. Diese werden in dem Dateisystem als serialisierte Objekte gespeichert. Die Klasse `AccountManager` verwaltet die Benutzeranträge und hat als Einzige Zugriff auf die Klasse `AccountDAO`. Ein Benutzerantrag darf zu jedem Zeitpunkt nur von einem Administrator bearbeitet werden. Er kann genau einmal bearbeitet oder gelöscht werden.

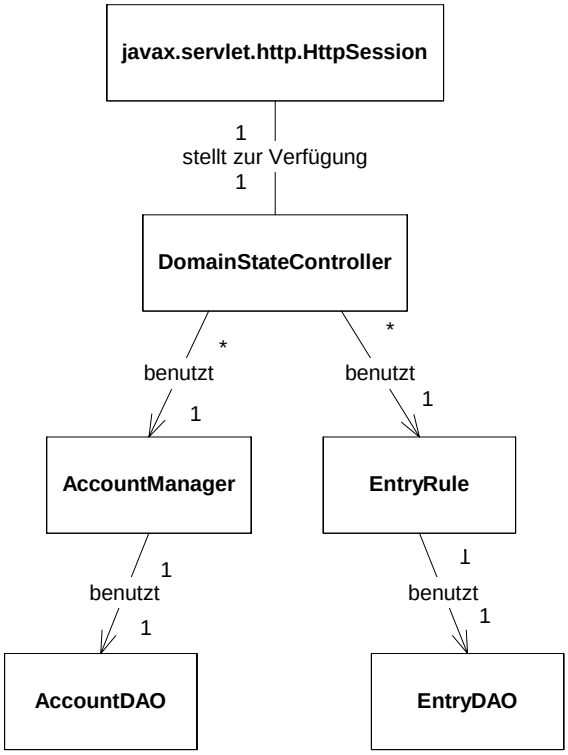


Abb. 16: Referenzen des DomainStateController

Die Klasse `EntryDAO` liest, schreibt, löscht und ändert Personeneinträge, die in der Verzeichnisdatenbank gespeichert sind. Sie wird von der Klasse `EntryRule` benutzt. Diese beinhaltet das Expertenwissen um aus ein Benutzerantrag einen Personeneintrag zu erzeugen oder zu ändern. Als Beispiel hierfür sei die Regel genannt, die der Erstellung einer Benutzeridentifikation aus Vor- und Nachnamen des Antragsteller dient, oder um die Zuordnung eines Standortes mit einem Mailserver herzustellen. Sie ist weiterhin dafür zuständig, das nur gültige Datensätze gespeichert werden.

Von einem Personeneintrag gibt es zwei unterschiedliche Datensichten die von dieser Klasse geliefert werden: Die Korrektursicht wird verwendet solange sich ein Antrag in Bearbeitung befindet und das System mit dem Anwender kommuniziert. Die Druckansicht stellt die Daten bereit die für das Kontoinformationblatt nötig sind, nachdem die Bearbeitung abgeschlossen wurde.

5.2.2 Initialisierung

Jedem als Administrator authentisierten Anwender wird eine Instanz der Klasse `DomainStateController` zu Verfügung gestellt. Würde mehr als eine Instanz verwendet, könnten sich diese unabhängig voneinander in unterschiedlichen Zuständen befinden. Die Kontrolle über die Reihenfolge der Zugriffe auf das Model

ginge verloren. Der `DomainStateController` darf daher keine öffentlichen Konstruktoren besitzen. Die Erweiterung durch andere Klassen muss ausgeschlossen werden.[21] Instanzen werden ausschließlich von dem Erbauer[22] `DomainStateControllerFactory` durch Aufruf der Methode `createInstanceOfDSC` erzeugt. Das Objekt wird in dem Parameter `HttpSession`[23] gespeichert und steht damit den Action-Klassen zu Verfügung. Der Erbauer hält während der gesamten Laufzeit der Anwendung Referenzen auf die Klassen der Geschäfts- und Datenzugriffsschicht. Diese werden jedem neu erzeugtem `DomainStateController` Objekt als Argumente übergeben. Sie werden in dem Erbauer mit dem Schlüsselwort `final` als unveränderbar deklariert, damit sie nicht versehentlich durch den `DomainStateController` geändert werden können. Da beliebig viele `DomainStateController` Objekte konkurrierenden Zugriff auf das Model haben, werden die öffentlichen Methoden gegebenenfalls mit den Schlüsselwort `synchronized` geschützt.

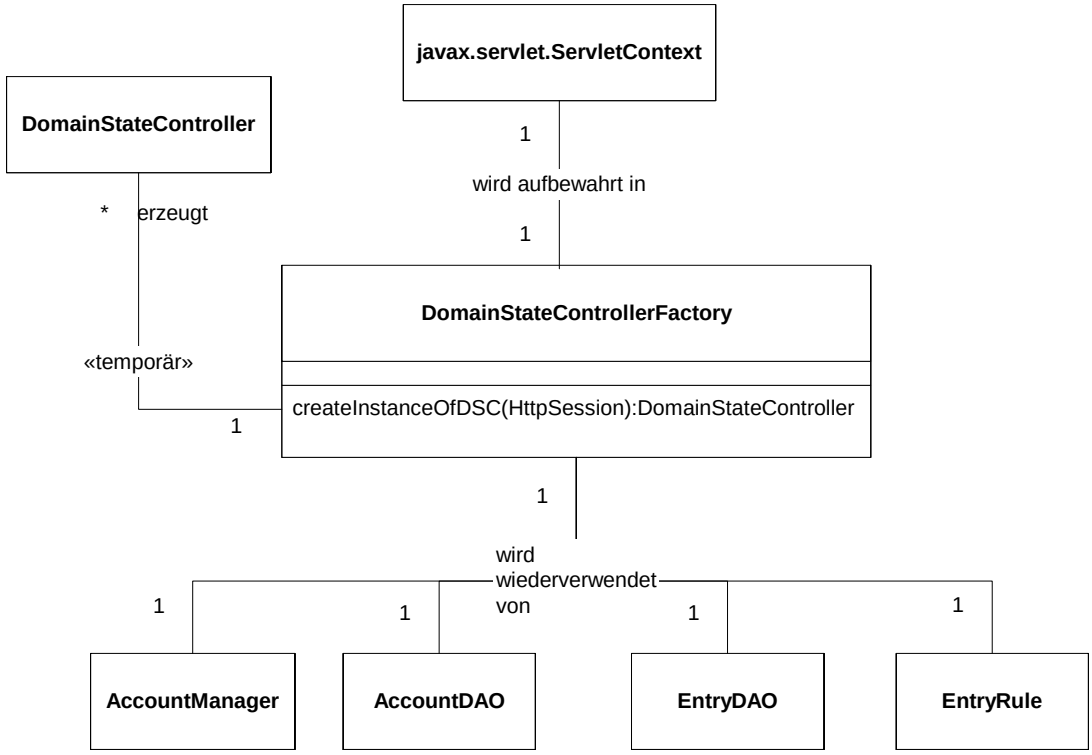


Abb. 17: Erzeugen des DomainStateController

[21] Siehe dazu [Blo01, S. 89ff]

[22] Entwurfsmuster Erbauer: Siehe dazu [Gam96, S. 119ff]

[23] Vereinfacht ausgedrückt: Ein assoziatives Array, das während der Interaktion mit den Anwender besteht.

5.2.3 Zugriff der Aktionen

Alle Action-Klassen des administrativen Bereiches der Anwendung werden von der Klasse `AdministrationBaseAction` abgeleitet. Sie definiert die Methode `getMyInstanceOfDSC` und liefert die zuständige Instanz des `DomainStateController`. Die Action-Klassen erhalten durch den Aufruf dieser Methode ihren zuständigen `DomainStateController`. Die Oberklasse kennt den Schlüssel unter der das Objekt abgelegt ist, prüft ob das Objekt nicht Null ist und führt eine Typkonvertierung durch.

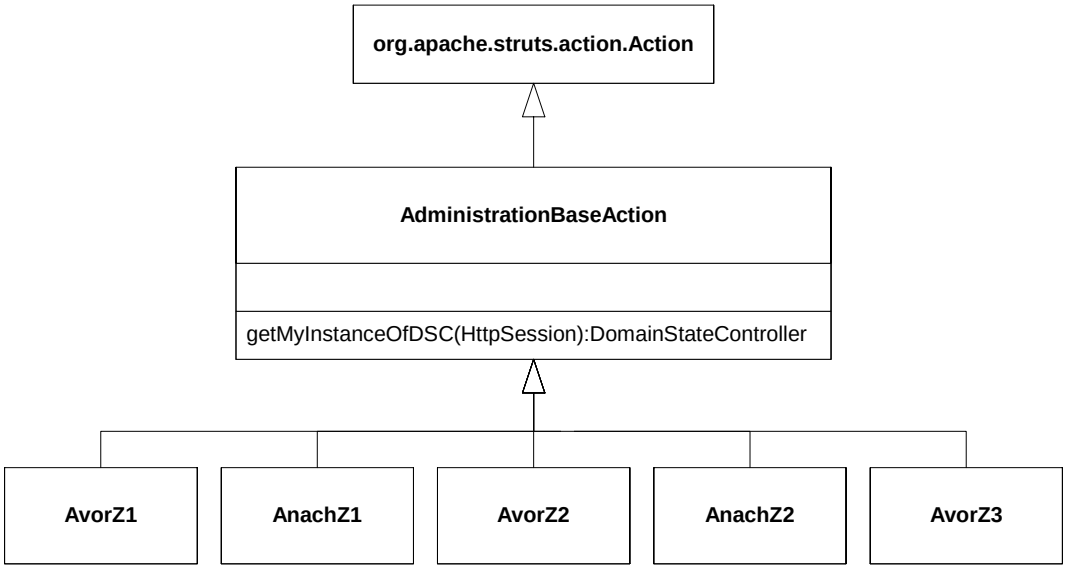


Abb. 18: Action-Klassen erhalten Zugriff auf den DomainStateController

5.3 Iterative Entwicklung des Zustandsautomaten

Der Zustandsautomat wird in mehreren Schritten entwickelt. Zuerst wird der Gutfall dargestellt, dann die erwarteten Abweichungen und als letztes die Fehlerbehandlung. Die verschiedenen Szenarien werden als Sequenzdiagramme dargestellt, aus denen der Automat abgeleitet wird.

5.3.1 Erste Iteration

Ausgangspunkt des ersten Schrittes ist der Zustandsautomat der Controller-Schicht, der in der `struts-config.xml` und den JSPs implementiert wird. Die JSP `ListAccountRequest` und `ProccessChangeAccountRequest` werden als Zustand Z1 und Z2 dargestellt. Beide JSP zeigen dynamischen Inhalt an, der in den Aktionen AvorZ1 (sprich: Aktion vor Zustand) und AvorZ2 erzeugt wird. Beide

Zustände nehmen Eingaben des Benutzers entgegen, die in den Aktionen AnachZ1 und AnachZ2 ausgewertet werden. Die JSP `ChangeAccountHandout` stellt einen dynamischen Inhalt dar, der in der Aktion AvorZ3 erzeugt wird. Sie nimmt keine Benutzereingabe entgegen und benötigt daher keine nachfolgende Aktion.

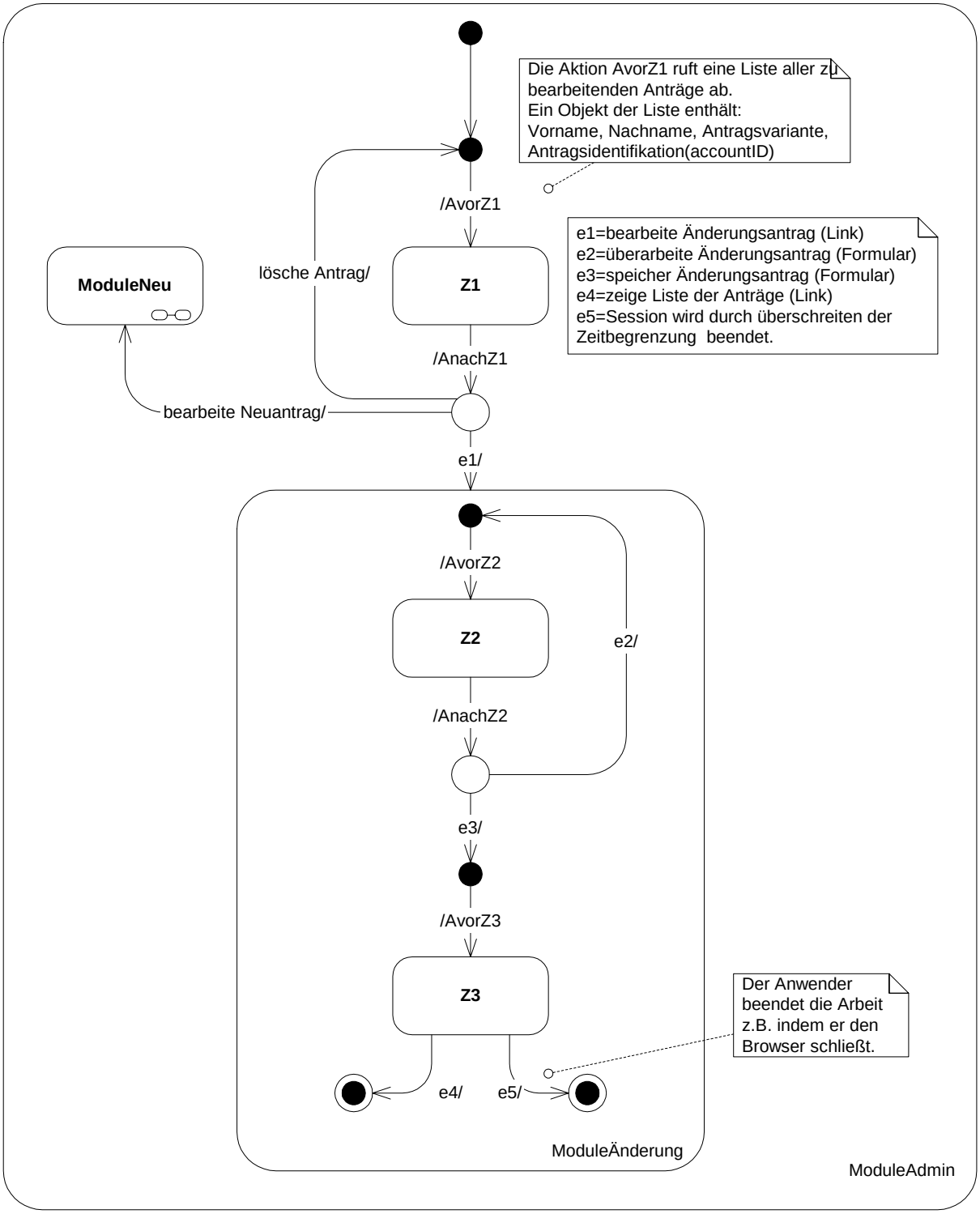


Abb. 19: Zustandsautomat der Controller-Schicht

Aus dem Diagramm lassen sich die Struts-Konfigurationsdateien ableiten. Statt die Konfiguration in einer großen Datei zu erstellen, wird sie in mehrere kleine, sogenannte Module aufgeteilt. Die Zusammengehörigkeit der Aktionen wird dadurch deutlich und die Übersicht und Änderbarkeit erhöht.

struts-config-moduleAdmin.xml:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE struts-config PUBLIC "-//Apache Software
    Foundation//DTD Struts Configuration 1.1//EN"
    "http://jakarta.apache.org/struts/dtds/struts-config_1_1.dtd">
<struts-config>
    <action-mappings>
          <action path="/vorZ1"
                type="de.awibremerhaven.action.admin.AvorZ1">
                <forward path="/admin/Z1.jsp"
                      name="darstellen"/>
          </action>
          <action path="/nachZ1"
                type="de.awibremerhaven.action.admin.AnachZ1">
                <forward path="/moduleAenderung/vorZ2"
                      name="e1-1" contextRelative="true">
                </forward>
                <!--Sowie ein forward Element für das
                     Lösche eines Antrages und dem
                     Bearbeiten eines Neuantrages.-->
          </action>
    </action-mappings>
</struts-config>
```

struts-config-moduleAenderung.xml:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE struts-config PUBLIC "-//Apache Software
    Foundation//DTD Struts Configuration 1.1//EN"
    "http://jakarta.apache.org/struts/dtds/struts-config_1_1.dtd">
<struts-config>
    <action-mappings>
          <action path="/vorZ2"
                type="de.awibremerhaven.action.admin.aendern.AvorZ2">
                <forward path="/admin/aendern/Z2.jsp"
                      name="darstellen"/>
          </action>
          <action path="/nachZ2"

    type="de.awibremerhaven.action.admin.aendern.AnachZ2">
                <forward path="/vorZ2" name="e2-1"/>
                <forward path="/vorZ3" name="e3-1"/>
          </action>
```

```
            <action path="/vorZ3"
                  type="de.awibremerhaven.action.admin.aendern.AvorZ3">
                  <forward path="/admin/aendern/Z3.jsp"
                        name="darstellen"/>
            </action>
      </action-mappings>
</struts-config>
```

Das Sequenzdiagramm der Abbildung 20 zeigt wie die Action-Klassen den DomainStateController aufrufen und dieser die Aufgaben an die Klassen des Models delegiert. Die Action-Klassen sind in der linken Spalte des Diagrams dargestellt. Die Abfolge der Aktionen lässt sich von dem Zustandsautomaten der Controller-Schicht (Abbildung 19) ablesen. Der Detaillierungsgrad der Darstellung ist auf der linken Seite am höchsten und nimmt nach rechts immer weiter ab. Der Nachrichtenaustausch mit der Datenzugriffsschicht wird nicht mehr dargestellt.

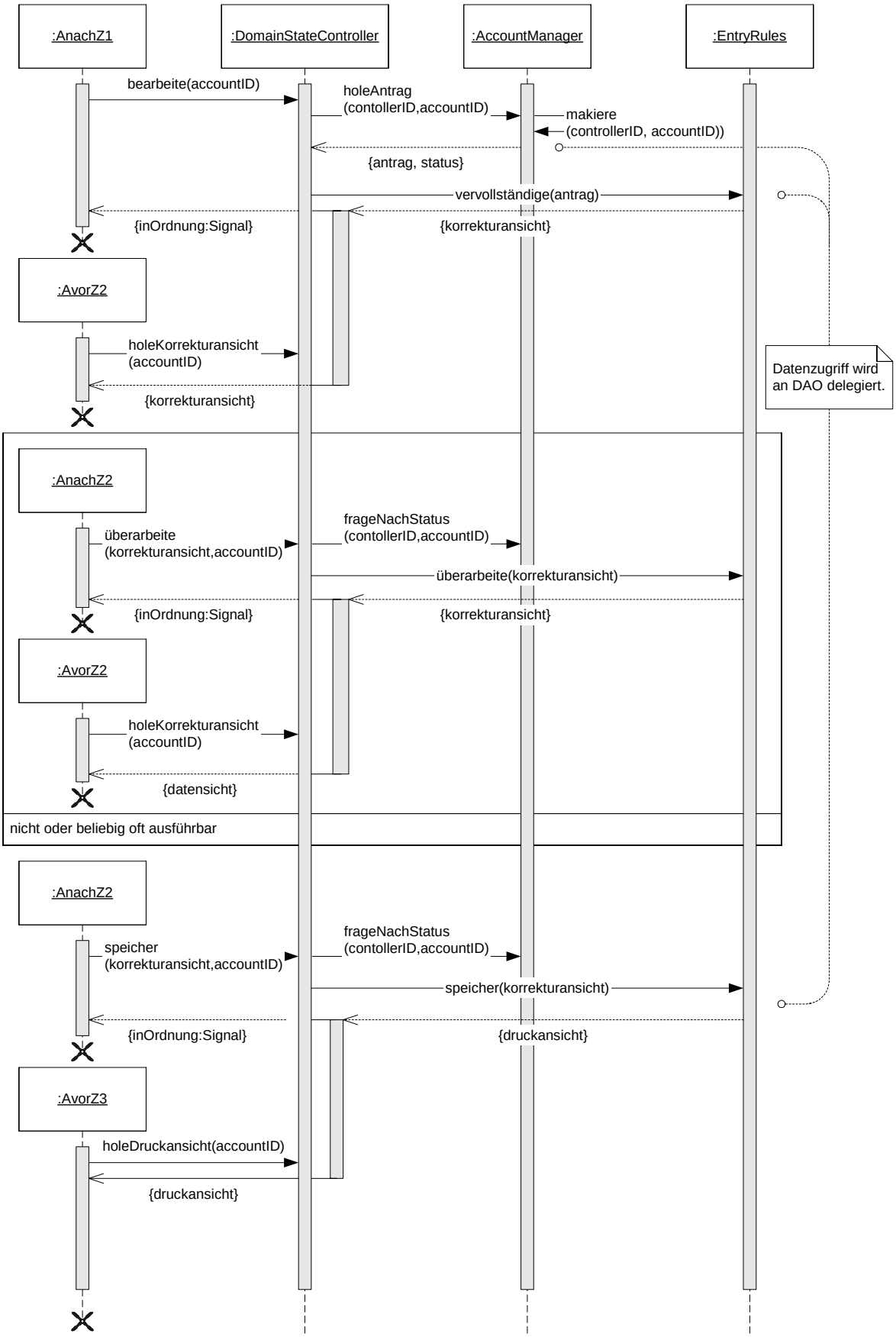


Abb. 20: Interaktion des DomainStateController mit der Controller- und Model-Schicht

Die Aktion AnachZ1 sendet als Reaktion auf das Ereignis e1 die Nachricht bearbeite mit der Antragsidentifikation als Parameter an den DomainStateController. Der DomainStateController erhält durch Senden der Nachricht holeAntrag von dem AccountManager den Benutzerantrag. Der Benutzerantrag wird an die EntryRule weitergegeben. Diese veranlasst das Lesen des dazugehörigen Personeneintrages. Auf den Daten des Personeneintrages und des Benutzerantrages werden die Transformationsregeln angewandt, die als Ergebnis die Korrekturansicht[24] liefert. Sie wird an den DomainStateController zurückgegeben und dort zwischengespeichert. Als Antwort auf die Nachricht bearbeite erhält die Aktion das Signal inOrdnung um anzuzeigen, dass der Vorgang erfolgreich durchgeführt wurde. Dies ist die Bedingung um die Bearbeitung an die Aktion AvorZ2 weiterzugeben.

Die Aktion AvorZ2 bereitet für den Zustand Z2 den dynamischen Inhalt vor. Sie sendet die Nachricht holeKorrekturansicht an den DomainStateController und erhält die zwischengespeicherte Korrekturansicht. Die Daten werden in einer Unterklasse von ActionForm kopiert, an die JSP des Zustandes Z2 weitergegeben und dargestellt. Der Administrator hat in diesem Zustand die Möglichkeit die Daten zu ändern (Ereignis e2) oder den Vorgang der Bearbeitung durch das Speichern (Ereignis e3) abzuschließen.

Die Aktion AnachZ2 beginnt mit einer Kontrollstruktur um in Abhängigkeit der Ereignisse den Programmablauf zu steuern. Empfängt sie das Ereignis e2, sendet es die Nachricht überarbeite mit der geänderten Korrekturansicht an den DomainStateController. Die Daten werden an die EntryRule weitergegeben. Sie führt entsprechend den Transformationsregeln bedingte Änderungen automatisch durch. (z.B. Ändern des Mailservers in Abhängigkeit des Beschäftigungsortes.). Die Korrekturansicht wird an den DomainStateController zurückgegeben und zwischengespeichert. Die Aktion AnachZ2 erhält das Signal über die positive Bearbeitung des Vorganges. Es wird die Aktion AvorZ2 aufgerufen, die wie in oben beschriebener Weise die Daten für die Darstellung entgegennimmt und anzeigen lässt. Der Anwender hat im Zustand Z2 wiederholt die Möglichkeit die Daten der Korrektursicht zu ändern oder zu speichern.

[24] Der Benutzerantrag ist eine echte Teilmenge der Korrekturansicht. Diese ist eine Teilmenge des Personeneintrages.

Empfängt die Aktion `AnachZ2` das Ereignis `e3`, sendet sie die Nachricht `speichere` an den `DomainStateController`. Die `EntryRule` prüft ob der Datensatz gültig ist und lässt die Änderungen des Personeneintrages durch die Instanz der Klasse `EntryDAO` speichern. Der Benutzerantrag wird von dem `AccountManager` als „Bearbeitet“ markiert. Die Druckansicht wird dem `DomainStateController` zurückgegeben und zwischengespeichert. Die Aktion `AnachZ2` erhält ein positives Signal und gibt daraufhin die Verarbeitung an die Aktion `AvorZ3` weiter.
Die Aktion holt sich von dem `DomainStateController` die Druckansicht. Die Bearbeitung wird an den Zustand `Z3` weitergegeben. Die JSP stellt die Daten der Druckansicht als Kontoinformationsblatt dar.

Aus dem Sequenzdiagramm wird der Zustandsautomat für die Domänensteuerungsschicht erstellt. Jede Nachricht einer Aktion an den DomainStateController wird als Übergang in einen Zustand dargestellt. Das Model wird während der Zustandsübergänge von dem DomainStateController aufgerufen. Der Zustandsautomat ist somit ein Mealy-Automat[25].

[25] Siehe dazu [Bal96, S. 275]

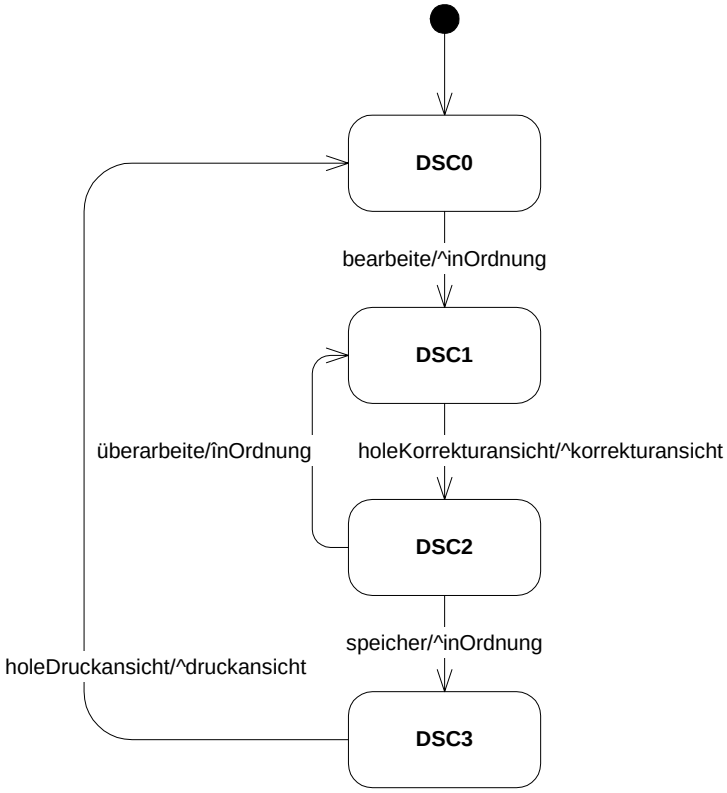


Abb. 21: Zustandsautomat des Gutfalles

5.3.2 Zweite Iteration

Der zweite Schritt beschreibt die geplanten Abweichungen vom Gutfall.

Ein Benutzerantrag kann zu jedem Zeitpunkt von nur jeweils einem Administrator bearbeitet werden. Das Signal `istGesperrt` wird zurückgegeben falls ein weiterer Administrator versucht den Antrag zu bearbeiten. Dem `AccountManager` muss deshalb bekannt sein von welchem `DomainStateController` aus ein Antrag bearbeitet wird. Der `DomainStateController` identifiziert sich durch einen eindeutigen String. Wurde ein Antrag einmal bearbeitet, wird der Versuch eines erneuten Bearbeiten mit dem Signal `istBearbeitet` quittiert. Wenn ein Antrag zum wiederholten Male gelöscht werden soll, wird das Signal `existiertNicht` zurückgegeben.

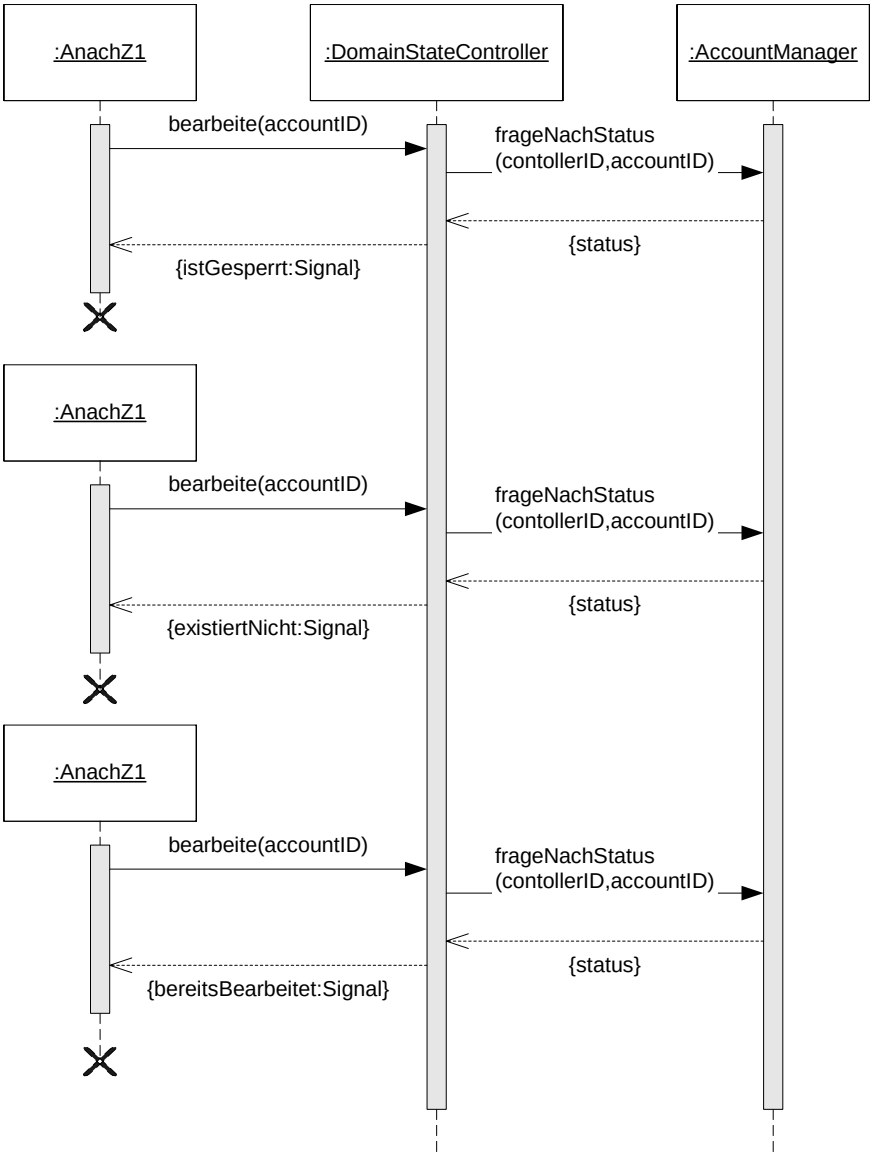


Abb. 22: Sequenzdiagramm der geplanten Abweichungen

Die Nachrichten führen zu drei neuen Übergängen, die keinen Zustandswechsel hervorrufen. Die Entscheidung welcher Zweig gewählt wird, kann erst dann getroffen werden, nachdem der `DomainStateController` den `AccountManager` aufgerufen hat. Das Ereignis `bearbeite` führt daher in eine dynamische Verzweigung, von der vier bedingte Übergänge ausgehen.

Ein Mealy-Automat sieht keine bedingten Übergänge vor. Der vorliegende Automat kann nunmehr mit der Notation nach David Harel[26] beschrieben werden.

[26] Siehe dazu [Bal96, S. 275]

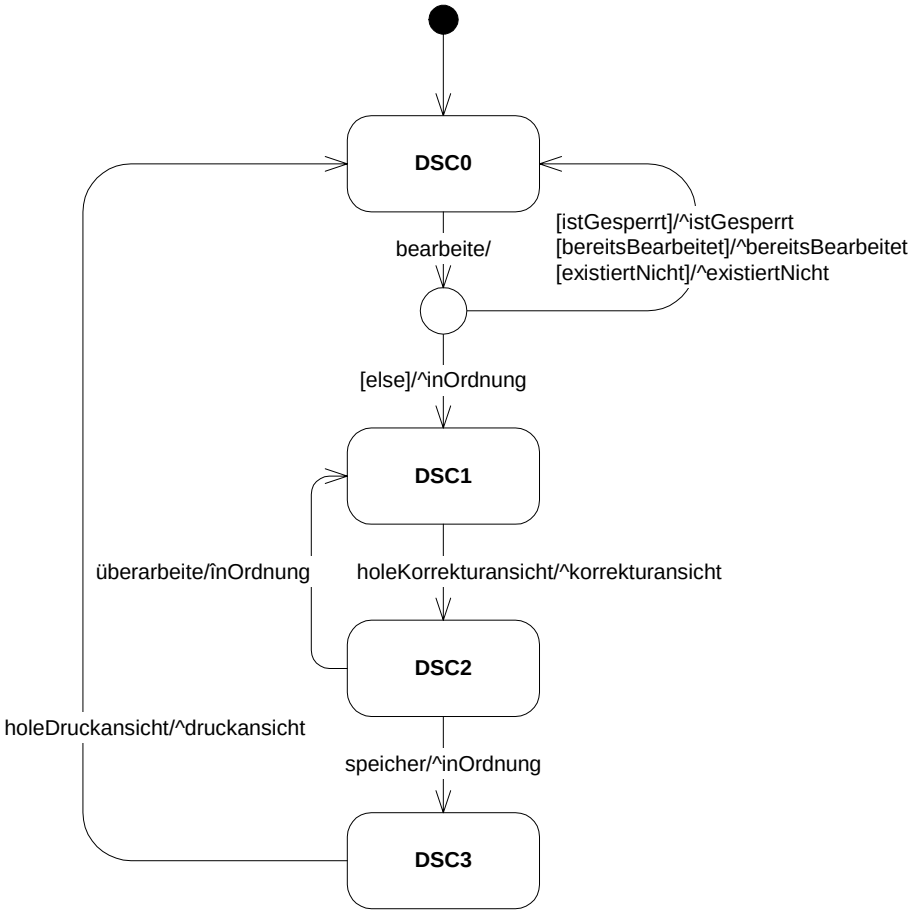


Abb. 23: Zustandsautomat der geplanten Abweichung

Die Kontrollstruktur in der Aktion `AnachZ1` wird erweitert um die Bearbeitung für die drei neuen Fälle an die Aktion `AvorZ1` weiterzuleiten. Der Anwender wird über die Ursache der abgewiesenen Bearbeitung informiert. Als Nachrichtenbehälter wird eine Instanz der Klasse `ActionMessage` verwendet. In der Struts-Konfiguration wird ein zusätzliches `forward` Element eingefügt.

5.3.3 Dritte Iteration

In dem dritten Schritt wird die Domänensteuerungsschicht um eine Ausnahmebehandlung erweitert um Fehler abzufangen, die von dem Model ausgehen können.

Die Nachrichten `bearbeite`, `überarbeite` und `speichern` veranlassen den `DomainStateController` dazu die Datenzugriffsklassen aufzurufen. Diese können eine Ausnahme werfen, wenn z.B. der Zugriff auf die Verzeichnisdatenbank oder dem Dateisystem gestört ist. Der `DomainStateController` fängt diese

Ausnahmen ab, geht daraufhin in den Fehlermodus und wirft eine `DomainException` an die aufrufende Aktion.

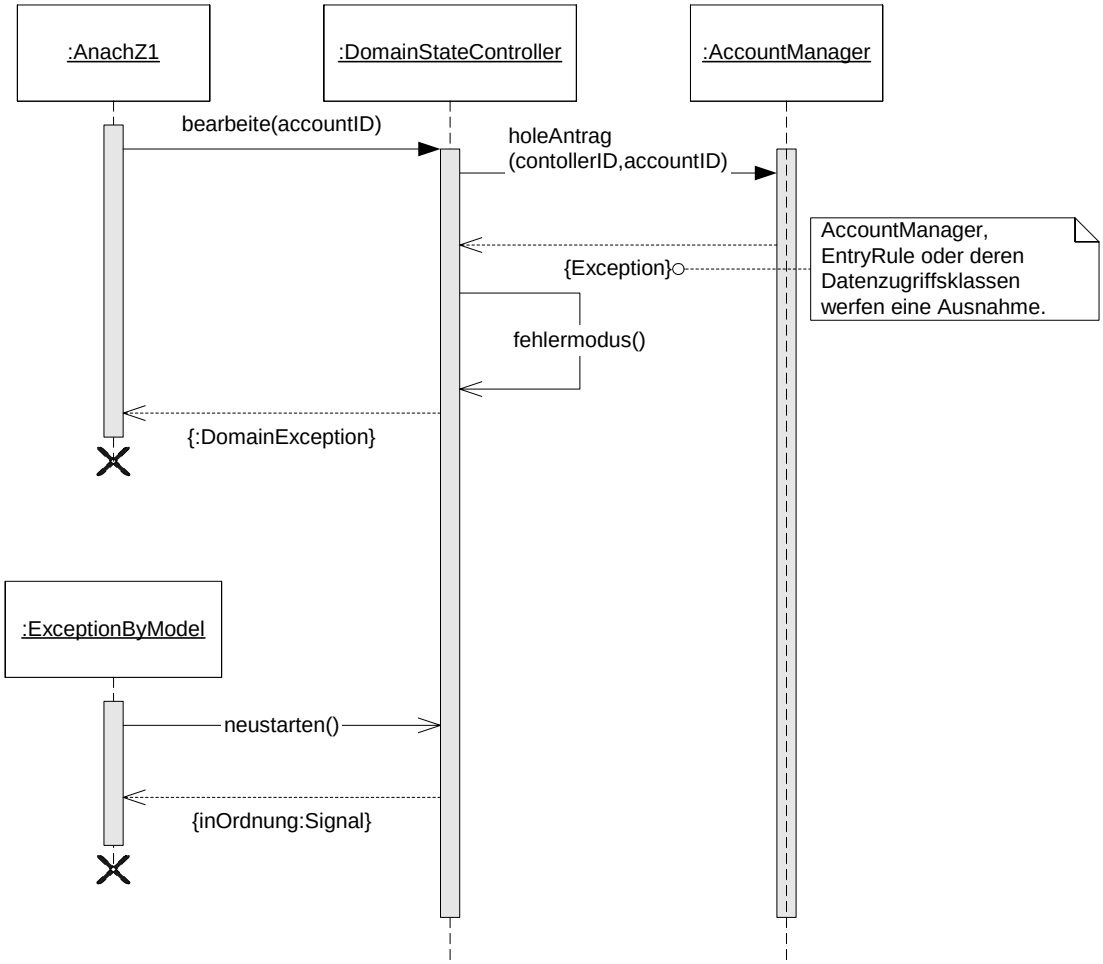


Abb. 24: Das Model erzeugt eine Ausnahme

Das Verarbeiten der Ausnahme erfordert eine neue Action-Klasse und einem zusätzliche Eintrag in die Struts-Konfiguration.

Deklarative Ausnahmebehandlung:

```
<global-exceptions>
    <exception key="FehlerImModell"
          type="de.awibremerhaven.eAccount.ModelException"
          path="/exceptionByModel">
    </exception>
</global-exceptions>
<action-mappings>
    <action path="/exceptionByModel"
      type="de.awibremerhaven.eAccount.action.admin.ExceptionByModel">
    </action>
</action-mappings>
```

Die Action-Klasse `ExceptionByModel` erzeugt für den Anwender eine Fehlermeldung. Sie sendet die Nachricht `neustarten` an den

`DomainStateController` der darauf den Fehlerzustand `DSC4` verlässt. Die Bearbeitung wird an die Aktion `AvorZ1` weitergegeben.

Für umfangreichere Systeme können an dieser Stelle Diagnose- und Test-Funktionen eingerichtet werden. Bei einer Anwendung dieser Größe wäre das Implementieren von Diagnose- und Test-Funktionen aufgrund des Umfanges und der Komplexität nicht gerechtfertigt.

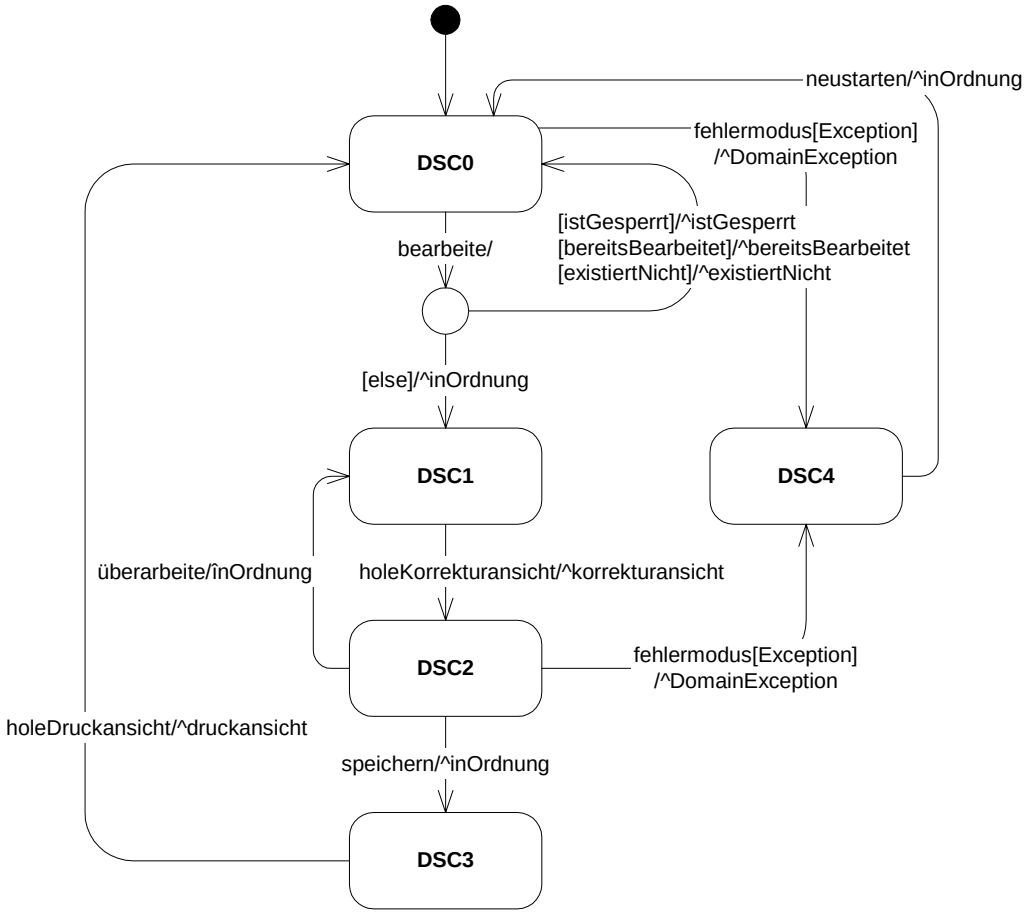


Abb. 25: Erweiterung des Automaten um einen Fehlerzustand

5.3.4 Vierte Iteration

Der letzte Schritt der Entwicklung besteht darin das Verhalten festzulegen, das auftreten soll falls der Automat ein Signal erhält das von dem gegenwärtigen Zustand nicht bearbeitet werden kann. Als Reaktion auf ein falsches Signal wird eine `DomainStateException` an die aufrufende Aktion geworfen. Der Zustand des Automaten ändert sich indes nicht. Für die Controller-Schicht besteht nun die Möglichkeit ein Ereignis zu senden das von dem Automaten angenommen wird oder

den Automaten durch Senden des Ereignisses `bearbeitungAbbrechen` in den Ausgangszustand zu versetzen. Das eingeführte Ereignis kann von allen Zuständen aufgerufen werden.

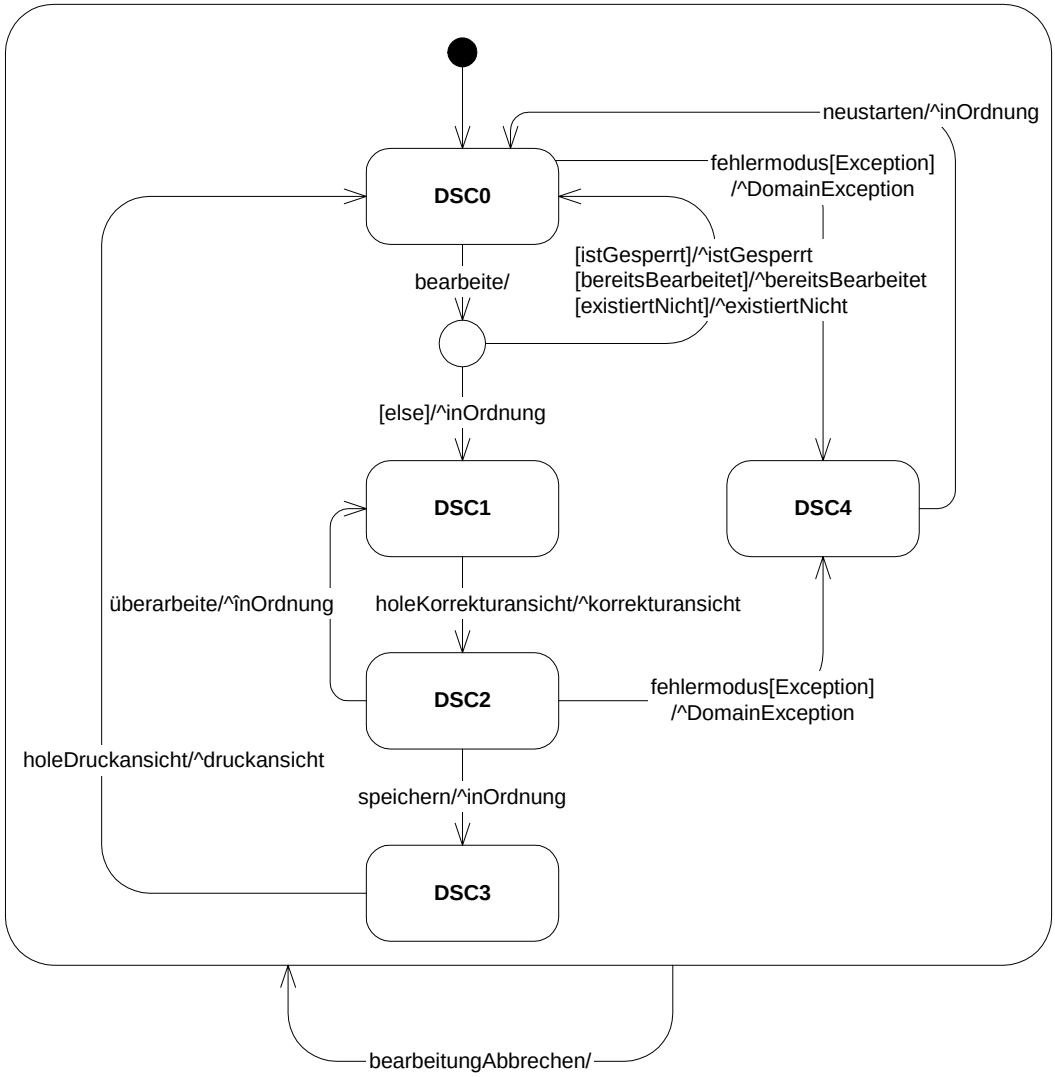


Abb. 26: Abschließende Erweiterung des Automaten

5.4 Implementierung

Der nun entwickelte Zustandsautomat wird durch das Entwurfsmuster Zustand[27] implementiert.

Der `DomainStateController` stellt für jeden Übergang eine öffentliche Methode bereit. Wenn das Objekt von einer Action-Klasse aufgerufen wird, hängt das

[27] Siehe dazu [Gam98, S. 398ff] und [Co98, S. 185ff]

Verhalten von dem aktuellen Zustand ab. Die abstrakte Klasse `DomainState` implementiert ebenfalls für alle Übergänge eine Methode. Diese sind in dem Paket sichtbar und können nur von dem `DomainStateController` aufgerufen werden. Alle Methoden werfen eine `DomainStateException`, solange sie nicht von den Unterklassen überschrieben werden.

Das zustandsspezifische Verhalten wird in den konkreten Unterklassen von `DomainState` implementiert. Die Klassen `DSC0` bis `DSC5` implementieren die durch den Zustand bedingten Übergänge, indem die entsprechenden Methoden der Oberklasse durch Aufrufe an das Model überschrieben werden. Übergänge die der Zustand nicht ausführen soll, werden nicht überschrieben und werfen weiterhin eine `DomainStateException`.

Das Objekt der Klasse `DomainStateController` hält für jeden Zustand eine Referenz auf ein Objekt von der Unterklasse `DomainState` bereit. Jeweils eine Instanz der Unterklasse ist das aktuelle Zustandsobjekt. Die Anfrage einer Action-Klasse wird in den öffentlichen Methoden des `DomainStateController` an dieses Objekt delegiert, um die zustandsspezifischen Aufrufe durchzuführen. Kommt es daraufhin zu einem Zustandswechsel, wird das aktuelle Zustandsobjekt gegen ein anderes Objekt ausgetauscht. Nach außen hin ändert sich dadurch das Verhalten der Klasse `DomainStateController`.[28]

[28] Der Quellcode für die Klassen DomainStateController, DomainState und DSC0 findet sich in Anhang 3.

Abb. 27: Implementierung des Zustandsautomaten

Wird für einen Zustand ein anderes internes Verhalten gefordert, wird die konkrete Unterklasse durch eine andere Implementierung ersetzt. Der Konstruktor der Klasse DomainStateController nimmt für jeden Zustand ein Objekt entgegen, das dem Typ DomainState entspricht. Änderungen in der Implementierung der Unterklassen ziehen dadurch keine Änderungen in dem DomainStateController oder einer Action-Klasse nach sich. Gründe für eine Änderung der Unterklasse wären z.B.:

- Änderung der Schnittstellen in den Klassen der Geschäftslogik- oder Datenzugriffsklassen.
- Das Austauschen der Model-Schicht gegen eine Attrappe, um die Controller-Schicht zu prüfen.

Hingegen stellt das Entfernen oder Zufügen von Übergängen einen schwerwiegenden Eingriff dar, weil die Schnittstellen der Klassen DomainStateController und DomainState dadurch geändert werden. Die Änderung der Schnittstelle hat Einfluss auf die abhängigen Action-Klassen, die ebenfalls angepasst werden müssen.

6.0 Zusammenfassung

Das Problem der Anwendung besteht in der Abhängigkeit der Controller- von der Model-Schicht. Diese wird durch eine Abfolge von Aufrufen an verschiedene Klassen der Model-Schicht erzeugt.

Die Lösung ist hier in einer zusätzlichen, trennenden Schicht zu sehen. Diese übernimmt die Aufgabe die Model-Schicht aufzurufen. Gleichzeitig schützt sie die Integrität der Daten vor falschen Aufrufen seitens der Controller-Schicht.

Die zusätzliche Schicht wird durch einen Zustandsautomaten realisiert. Es wurde von mir veranschaulicht wie sich dieser Automat aus einer Reihe von Sequenzdiagrammen entwickelte.

Abschließend wurde in dem Abschnitt 5.4 die Implementierung des Zustandsautomaten mit Hilfe eines Entwurfsmusters veranschaulicht.

7.0 Ausblick

Das zustandslose HTTP ermöglicht, dass nach einer Anfrage jede beliebige weitere Anfrage folgen kann. Es ist nicht möglich sich darauf zu verlassen, dass der Anwender die vordefinierte Reihenfolge der Aktionen einhält.

Die Reload-, Back- und Stop-Schaltflächen des Browsers ermöglichen es dem Anwender aus einer Dialogreihenfolge auszubrechen. Somit kann es vorkommen, dass Formulardaten einer Seite mehrfach an den Server gesandt werden oder ein Datensatz wiederholt in der Datenbank gespeichert wird. Der Client befindet sich in einem Zustand der nicht dem der serverseitigen Anwendung entspricht.

Mittels JavaScript besteht die Möglichkeit auf der Client-Seite Einfluss auf die Schaltflächen zu nehmen. Da der Anwender die Verwendung von JavaScript ausschalten kann, ist diese Methode unzuverlässig. Die Dialogfolge clientseitig durchzusetzen ist somit ausgeschlossen.

Das mehrfache Absenden von Formulardaten kann also nur serverseitig kontrolliert werden. Der erste erforderliche Schritt hierzu besteht zunächst in der Feststellung dieser Mehrfachsendung. Hierfür wird von dem Framework Struts das Entwurfsmuster Synchronizer Token implementiert. Die Aktion vor einem Zustand erzeugt durch Aufruf der Methode `saveToken(HttpRequest)` ein eindeutiges Zeichen. Das Token wird in der `HttpSession` des Benutzers gespeichert und der nachfolgenden JSP als verstecktes Formularfeld übergeben. Nach Betätigen der Submit-Schaltfläche werden die Formulardaten mit dem Token an die verarbeitende Aktion abgeschickt. Die Aktion stellt durch die Methode `isTokenValid(HttpRequest):boolean` fest, ob das in dem Formular übergebene Token mit dem in der `HttpSession` übereinstimmt. In diesem Fall ist sichergestellt, dass sich Client und Server im gleichen Zustand befinden. Das Token wird durch Aufruf der `Methode reset(HttpRequest)` zurückgesetzt. Der Aufruf an das Model erfolgt anschließend. Werden die Formulardaten durch betätigen der Back- und Submit-Schaltfläche wiederholt übermittelt, liefert die Methode `isTokenValid` den Wert `false`.

Das Framework bietet die Möglichkeit der Feststellung eine Abweichung von der Dialogfolge festzustellen. Es liegt in der Verantwortung des Anwendungsentwicklers die Zustände des Clients und des Servers zu synchronisieren.

Der Artikel „Protect Web application control flow“[29] beschreibt ein Verfahren um voneinander abweichende Zustände zu synchronisieren. Zusätzlich wird die letzte gültige Benutzereingabe zwischengespeichert. Weicht der Anwender von der vorgegebenen Dialogfolge ab, wird die letzte gültige Seite mit der gespeicherten Benutzereingabe erneut dargestellt, so dass der Anwender die Bearbeitung fortführen kann.

Erreicht wird dies durch die abstrakte Klasse `SynchroAction`, die eine Unterklasse von `org.apache.struts.action.Action` darstellt. Sie überschreibt die `execute` Methode und stellt ihrerseits die abstrakte Methode `executeSynchro` zur Verfügung. In der `execute` Methode der Klasse `SynchroAction` wird das Synchronized Pattern angewandt und die Benutzereingabe zwischengespeichert. Im Fall einer Dialogabweichung wird die Benutzereingabe an die JSP weitergegeben, die für die Zustandssynchronisierung in der `struts-config.xml` angegeben wurde. Die anwendungsspezifischen Aktionen werden von der Klasse `SynchroAction` abgeleitet. Das Model wird durch überschreiben der abstrakten Methode `executeSynchro` implementiert.

Die abstrakte Methode `executeSynchro` wird überschrieben um das Model zu implementieren. Die Methode wird von der `excecute` Methode der abstrakten Unterklasse aufgerufen.

Das oben angeführte Verfahren ist nach meiner Beurteilung für kleine bis mittlere Anwendungen einsetzbar.

Eine fertige Lösung bietet die Erweiterung „Struts Workflow-Extension“.[30]

Der Struts Controller, der alle Http-Anfragen entgegennimmt wird um die Klasse `WorkflowRequestProcessor` ergänzt. Die Action Klassen des Struts Frameworks werden bei dieser Lösung nicht durch eine andere Klasse erweitert. Die Workflow-Extension bietet prinzipiell den gleichen Nutzen wie das bereits dargestellte „Protect Web application control flow“ Verfahren.

Die Dialogfolge einer Anwendung kann als Zustandsautomat der Controller-Schicht dargestellt werden. Eine Dialogfolge wird in der Erweiterung als Workflow bezeichnet und in der Struts-Konfiguration deklariert. Das meines Erachtens bemerkenswerte dieser Erweiterung besteht in der Fähigkeit einen entworfenen

[29] Siehe dazu [Gua03]

[30] Siehe dazu [Bau02]

Automaten durch die vorgegebenen Schlüsselwörter direkt auf die Implementierung abzubilden.
Nach Angaben der Entwickler der Fa. Livinglogic New Media Solutions ist die Erweiterung hinreichend sicher genug um für e-Commerce- und elektronische Bankanwendungen eingesetzt zu werden.

Das Projekt „Struts Flow“[31] liegt seit dem Juni 2004 als Version 0.1 vor. Die Steuerung der Dialogfolge wird durch die Programmiersprache JavaScript implementiert. Die Steuerdateien werden von dem Struts Framework aufgerufen. Dieser Ansatz bietet von allen bis dato dargestellten Verfahren die höchste Flexibilität. Die Erweiterung ist zwar für den Einsatz mit Struts entwickelt worden, kann aber auch für andere Anwendungsbereiche wie etwa Web Services oder Portlets verwendet werden.
Da das Projekt derzeit noch den Status eines Prototypen besitzt, sind für den wirtschaftlichen Einsatz noch Verbesserungen im Bereich der programmiersprachenübergreifenden Schnittstelle notwendig.

[31] Siehe dazu [Bro04]

8.0 Anlagenverzeichnis

Anlage 1: Formulare

An
Sekretariat
AWI-Rechenzentrum

Benutzerantrag für das AWI-Rechnernetz

Name: _____ **Vorname:** _____

Ort : _____ **Gebäude/Raum:** _____

Fachbereich: _____ **Telefon:** _____

Sektion/Gruppe: _____

☐ **Beantragung** ☐ **Änderung/Verlängerung** ☐ **Löschung** (bitte ankreuzen)

O E-Mail-Account

O Windows-Account für File- und Printservices (auch File-Services für Macs)

O VPN-Zugang (Windows-Account ist erforderlich)

O Unix-Account für das Workstation-Cluster (zutreffendes Cluster unterstreichen):

BAT / BIO / CHE / EDV / GLA / GPH / MET / MOB / MPH / SAT / TPH

Dauer (max. 3 Jahre): _____

Status am AWI? (bitte ankreuzen:)

O Mitarbeiter/in (Anstellung mit Vertrag, z.B. Angestellte, Arbeiter, Beamte, Studenten, FÖJ)

O Gast (z.B. Gäste ohne Vergütung, Praktikant)

Verantwortlicher Betreuer: _____

Voraussichtliches Ende der Aufgabe (unbedingt eintragen): _____

Der/die Antragsteller/in bestätigt hiermit, dass er/sie zur Kenntnis nimmt, dass

1. es verboten ist, Zugangsberechtigungen (Passwörter) weiterzugeben,
2. es verboten ist, Ressourcen unberechtigt in Anspruch zu nehmen (dazu gehört auch der Missbrauch lizenzierter Software und copyright-geschützter Daten),
3. es verboten ist, von AWI-Rechnern aus Versuche zu starten, in andere Rechnernetze einzubrechen,
4. alle Dokumente, die für Empfänger außerhalb des AWI bestimmt sind (mail, ftp, WWW), der allgemeinen Verschwiegenheitspflicht aller AWI-Angestellten unterliegen.

_____ _____

Datum, Unterschrift des Antragstellers

Unterschrift des Fachbereichs- oder Sektionsleiters, bzw. eines offiziellen Vertreters

Bitte füllen Sie den Antrag vollständig aus!
Damit werden Verzögerungen bei der Bearbeitung vermieden.

papierener Benutzerantrag

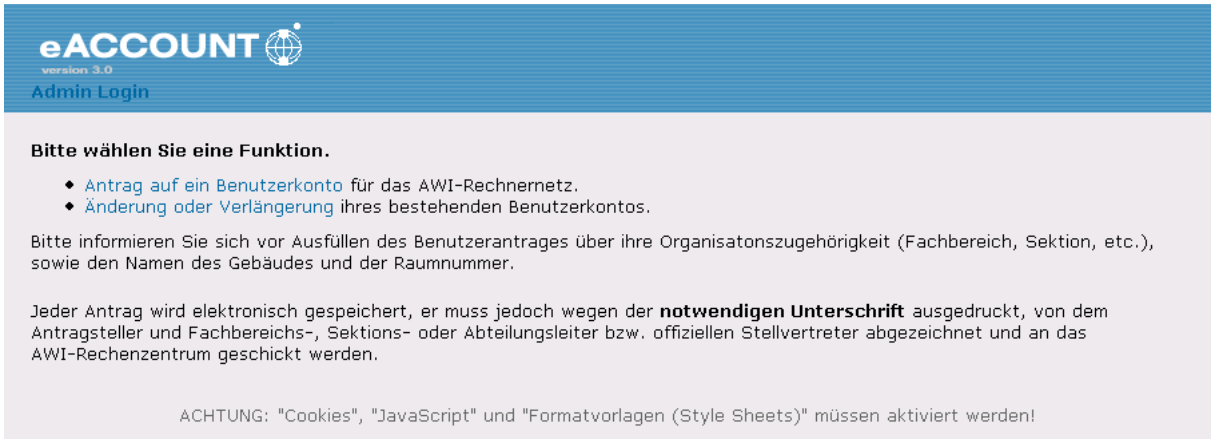
An
Sekretariat
AWI-Rechenzentrum

Wird vom AWI-RZ ausgefüllt:

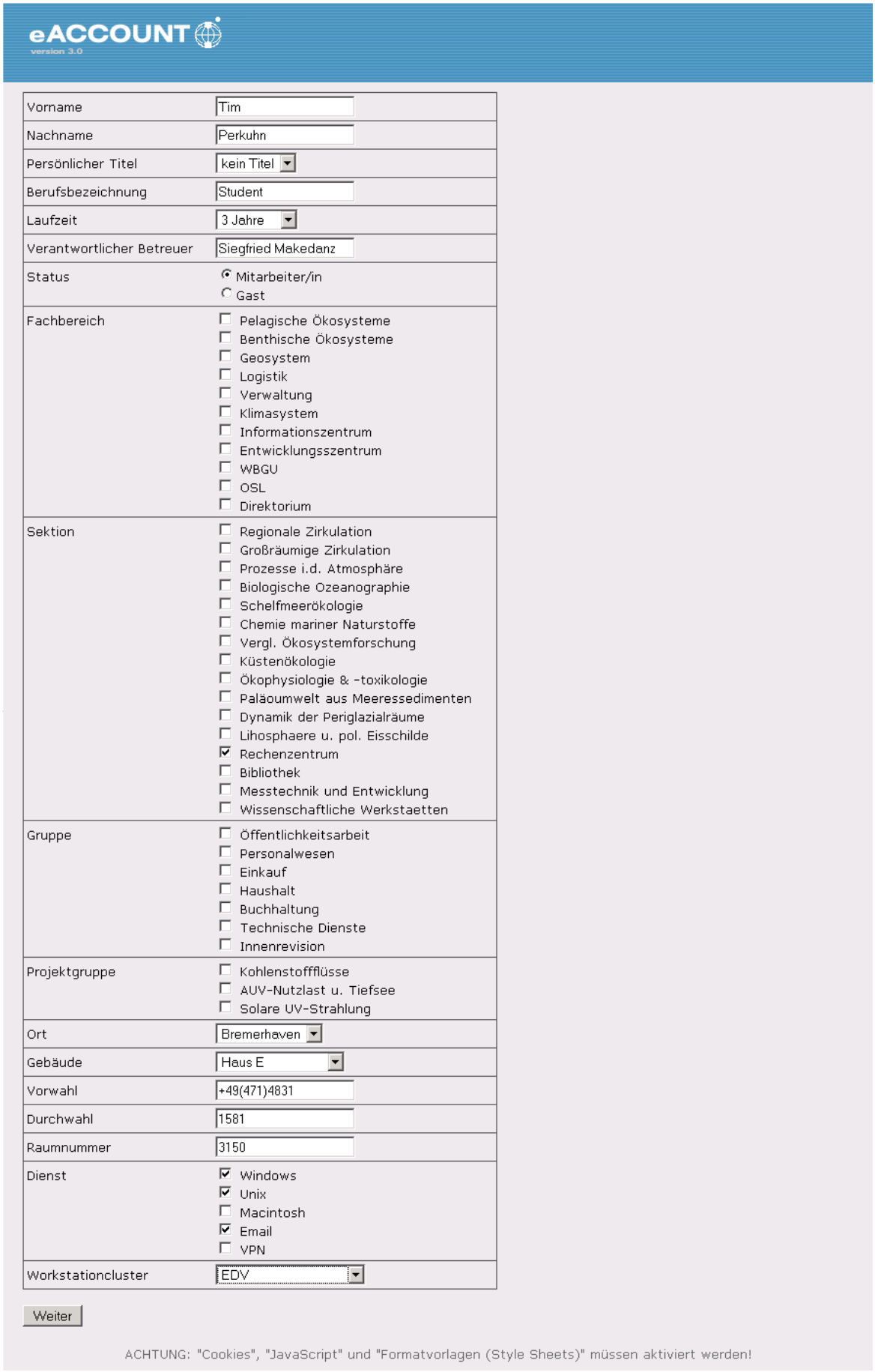
Mail:	
E-mail-Adresse	
Username	
Password	
eingerichtet am:	
PC:	
Username	
Password	
Arbeitsgruppe	
eingerichtet am:	
VPN:	
.....Gruppen-Account:	
.....Passwort dazu:	
.....eingerichtet am:	
.....informiert am:	
Unix:	
Account /UID	
Password	
home-directory	
Quote	
Gruppe/GID	
Netgroup	
eingerichtet am:	
Auslaufdatum:	
Benutzer benachrichtigt:	

papierenes Kontoinformationsblatt

Anlage 2: Bildschirmfotos



index.jsp



newUserAccount.jsp

eACCOUNT
version 3.0

- Bitte **drucken Sie diese Seite** zum Unterschreiben aus.
- Benutzen Sie dafür die Druckfunktion Ihres Browsers.

An das
Sekretariat
des
AWI-Rechenzentrum

Mon Jul 19 18:41

Benutzerantrag für das AWI-Rechnernetz

Beantragung

Vorname	Tim
Nachname	Perkuhn
Berufsbezeichnung	Student
Laufzeit	36
Ort	Bremerhaven
Gebäude	E
Raumnummer	3150
Telefonnummer	+49(471)4831-1581
Sektion	Rechenzentrum
Dienst	Windows Unix Email
Workstationcluster	EDV

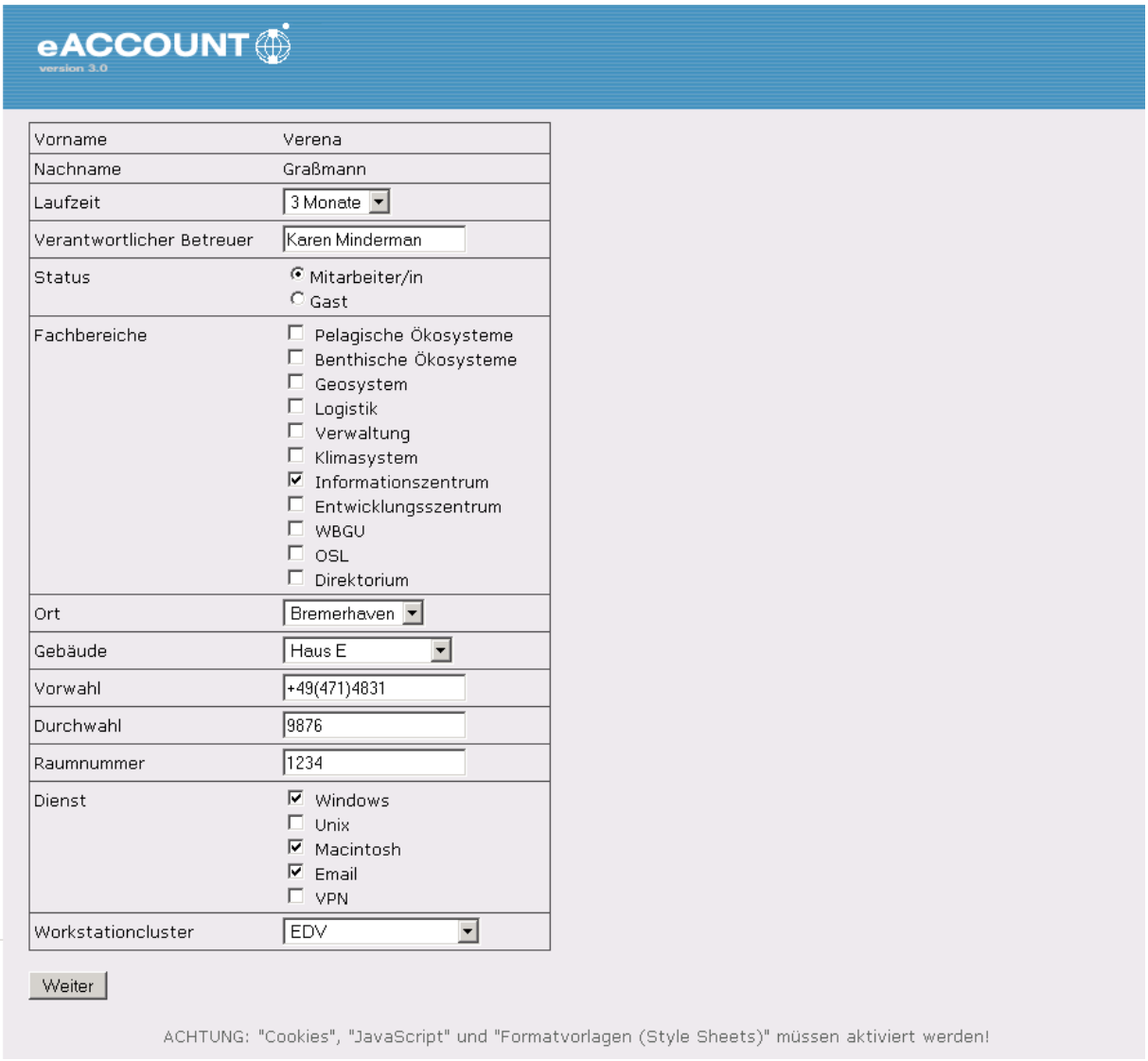
Der(Die) Antragsteller(in) bestätigt hiermit, daß ihm(ihr) bekannt ist, daß

1. es verboten ist, Zugangsberechtigungen (passwords) weiterzugeben,
2. es verboten ist, Ressourcen unberechtigt in Anspruch zu nehmen (dazu gehört auch der Mißbrauch lizensierter Software und copyright-geschützter Daten),

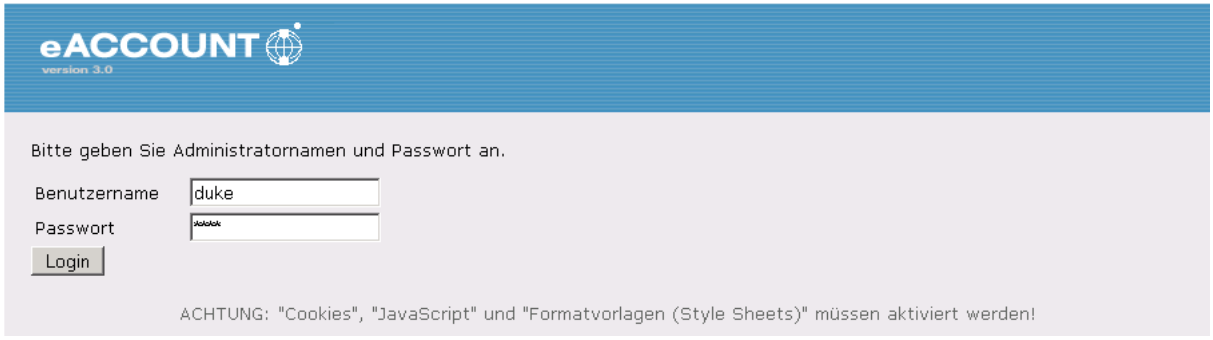
Datum, Unterschrift des(der) Antragsteller(in)

Unterschrift des(der)
Fachbereich-/Sektions-/Abteilungsleiter(in)
bzw. dessen Stellvertreter(in)

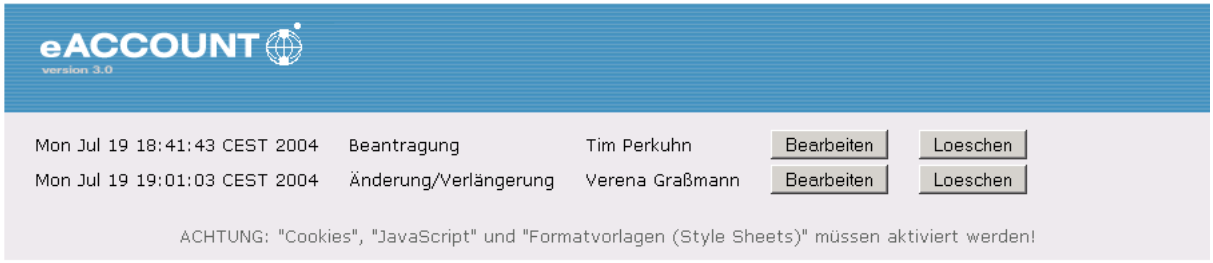
printDisplay.jsp



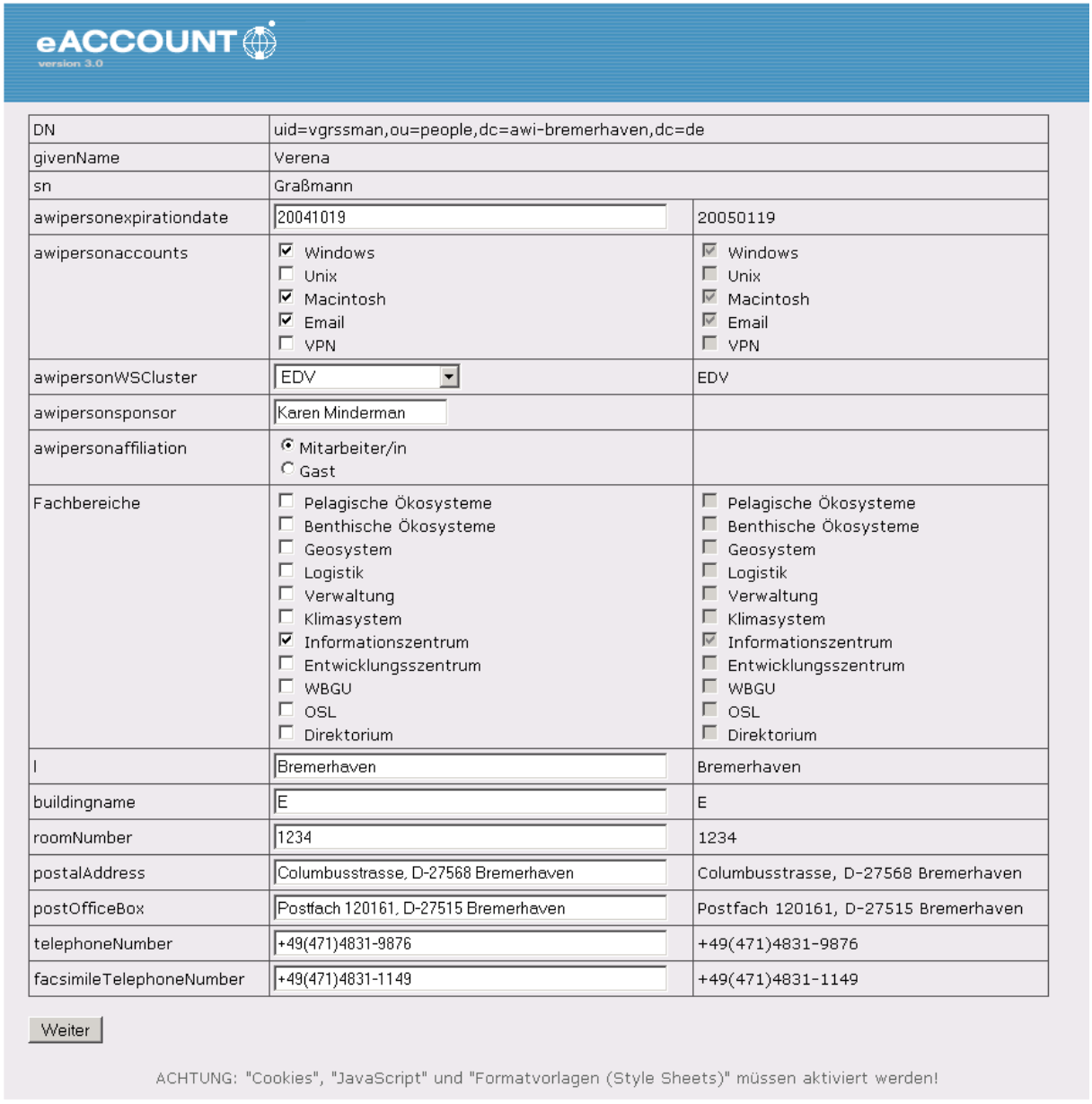
ChangeAccountRequest.jsp



Login4Admin.jsp



Z1 - ListAccountRequest.jsp



Z2 - ProcessChangeAccountRequest.jsp

eACCOUNT
version 3.0

- Bitte **drucken Sie diese Seite** aus.
- Benutzen Sie dafür die Druckfunktion Ihres Browsers.

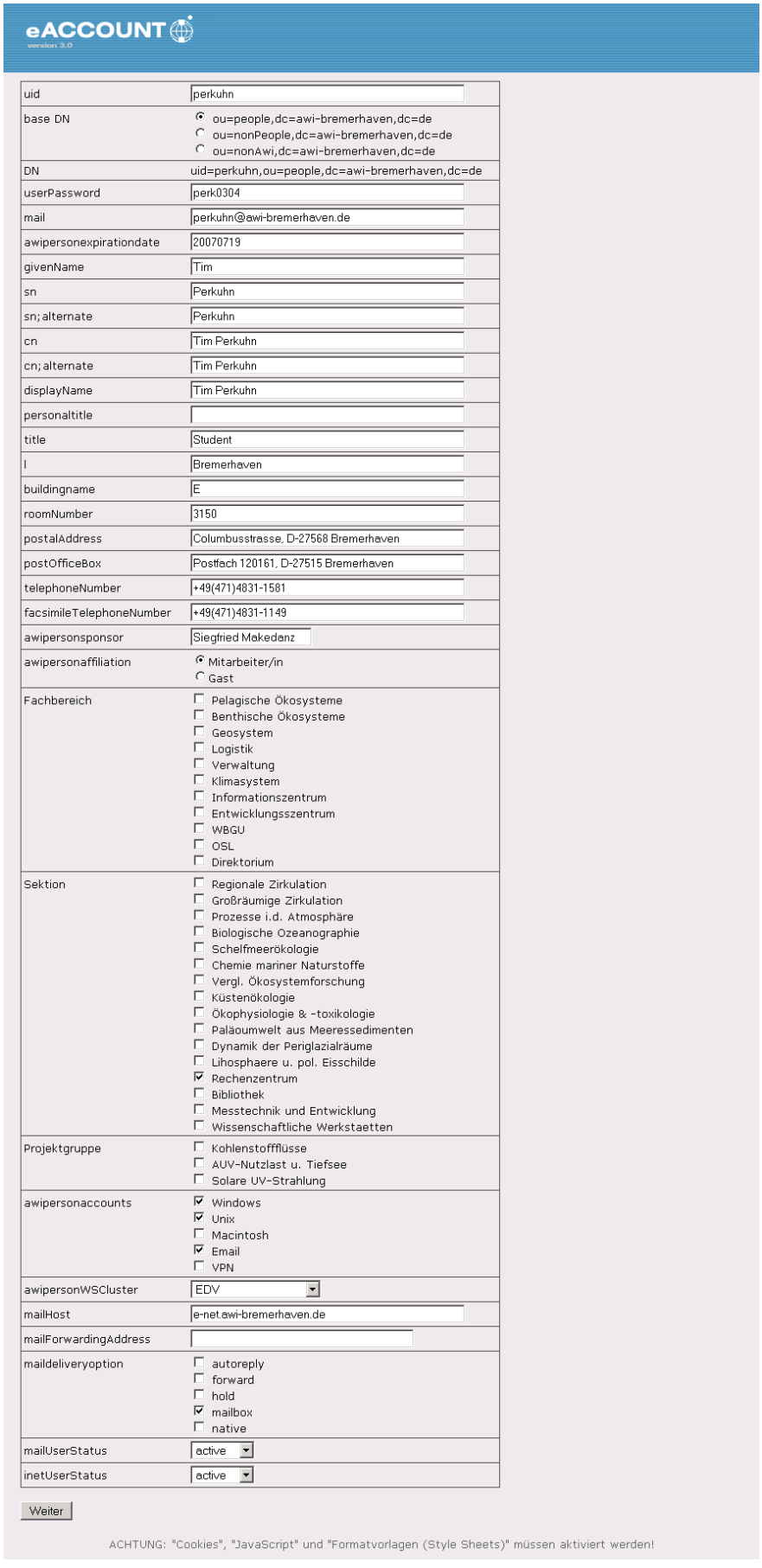
An
Sekretariat
AWI-Rechenzentrum

Wird vom AWI-RZ ausgefüllt:

Mail:	
E-mail-Adresse	
Username	
Password	
eingerichtet am	19.07.2004
Ansprechpartner	Jörg Kosinski, kosinski@awi-bremerhaven.de, +49(471)4831-1577 Siegfried Makedanz, smakedanz@awi-bremerhaven.de, +49(471)4831-1250
PC:	
Username	
Password	
Arbeitsgruppe	
eingerichtet am	
Ansprechpartner	Dietmar Schulze,.dschulze@awi-bremerhaven.de, +49(471)4831-1403

Benutzer benachrichtigt:

Z3 - ChangeAccountHandout.jsp



ProcessNewAccount.jsp

eACCOUNT
version 3.0

- Bitte **drucken Sie diese Seite** aus.
- Benutzen Sie dafür die Druckfunktion Ihres Browsers.

An
Sekretariat
AWI-Rechenzentrum

Wird vom AWI-RZ ausgefüllt:

Mail:	
E-mail-Adresse	perkuhn@awi-bremerhaven.de
Username	perkuhn
Password	perk0304
eingerichtet am	19.07.2004
Ansprechpartner	Jörg Kosinski, kosinski@awi-bremerhaven.de, +49(471)4831-1577 Siegfried Makedanz, smakedanz@awi-bremerhaven.de, +49(471)4831-1250
PC:	
Username	perkuhn
Password	perk0304
Arbeitsgruppe	
eingerichtet am	
Ansprechpartner	Dietmar Schulze,.dschulze@awi-bremerhaven.de, +49(471)4831-1403
Unix:	
Account/UID	perkuhn
Password	perk0304
home-directory	
Quote	
Gruppe/GID	
Netgroup	
eingerichtet am	
Auslaufdatum	
Ansprechpartner	Herbert Liegmahl-Pieper, hliegmahl@awi-bremerhaven.de, +49(471)4831-1269

Benutzer benachrichtigt:

NewAccountHandout.jsp

Anlage 3: Quellcode des Zustandsautomaten

DomainStateController.java

```
package de.awibremerhaven.eAccount.DomainController;
import de.awibremerhaven.eAccount.Geschaeftslogik.Druckansicht;
import de.awibremerhaven.eAccount.Geschaeftslogik.Korrekturansicht;
import de.awibremerhaven.eAccount.action.admin.ModelException;
public final class DomainStateController {
    DomainStateController(
        String id,
        DomainState dsc0,
        DomainState dsc1,
        DomainState dsc2,
        DomainState dsc3,
        DomainState dsc4) {
        this.CONTROLLER_ID = id;
        this.dsc0 = dsc0;
        this.dsc1 = dsc1;
        this.dsc2 = dsc2;
        this.dsc3 = dsc3;
        this.dsc4 = dsc4;
        aktuellerZustand = this.dsc0;
    }
    final String CONTROLLER_ID;
    private final DomainState dsc0, dsc1, dsc2, dsc3, dsc4;
    DomainState aktuellerZustand;
    public Signal bearbeite(String accountID)
        throws ModelException, DomainStateException {
        return this.aktuellerZustand.bearbeite(accountID);
    }
    public Signal ueberarbeite(String accountID)
        throws ModelException, DomainStateException {
        return this.aktuellerZustand.ueberarbeite(accountID);
    }
    public Signal speichere(String accountID)
        throws ModelException, DomainStateException {
        return this.aktuellerZustand.speichere(accountID);
    }
    public Korrekturansicht holeKorrekturansicht(String accountID)
        throws DomainStateException {
        return this.aktuellerZustand.holeKorrekturansicht(accountID);
    }
    public Druckansicht holeDruckansicht(String accountID)
        throws DomainStateException {
        return this.aktuellerZustand.holeDruckansicht(accountID);
    }
    public Signal neustarten()
        throws ModelException, DomainStateException {
        return this.aktuellerZustand.neustarten();
    }
    public Signal bearbeitungAbbrechen()
        throws ModelException, DomainStateException {
```

```
        return this.aktuellerZustand.bearbeitungAbbrechen();
    }
    void geheInZustand_DSC0() {
        aktuellerZustand = dsc0;
    }
    void geheInZustand_DSC1() {
        aktuellerZustand = dsc1;
    }
    void geheInZustand_DSC2() {
        aktuellerZustand = dsc2;
    }
    void geheInZustand_DSC3() {
        aktuellerZustand = dsc3;
    }
    void geheInZustand_DSC4() {
        aktuellerZustand = dsc4;
    }
}
```

DomainState.java

```
package de.awibremerhaven.eAccount.DomainController;
import de.awibremerhaven.eAccount.Geschaeftslogik.Druckansicht;
import de.awibremerhaven.eAccount.Geschaeftslogik.Korrekturansicht;
import de.awibremerhaven.eAccount.action.admin.ModelException;
public abstract class DomainState {
    public DomainState(DomainStateController besitzer) {
        this.besitzer = besitzer;
    }
    DomainStateController besitzer;
    String name;
    public Signal bearbeite(String accountID)
        throws ModelException, DomainStateException {
        throw new DomainStateException(this.name);
    }
    public Signal ueberarbeite(String accountID)
        throws ModelException, DomainStateException {
        throw new DomainStateException(this.name);
    }
    public Signal speichere(String accountID)
        throws ModelException, DomainStateException {
        throw new DomainStateException(this.name);
    }
    public Korrekturansicht holeKorrekturansicht(String accountID)
        throws DomainStateException {
        throw new DomainStateException(this.name);
    }
    public Druckansicht holeDruckansicht(String accountID)
        throws DomainStateException {
        throw new DomainStateException(this.name);
    }
    public Signal neustarten()
        throws ModelException, DomainStateException {
        throw new DomainStateException(this.name);
    }
    public Signal bearbeitungAbbrechen()
        throws ModelException, DomainStateException {
        throw new DomainStateException(this.name);
    }
}
```

DSC0.java

```
package de.awibremerhaven.eAccount.DomainController;
import de.awibremerhaven.eAccount.Geschaeftslogik.AccountManager;
import de.awibremerhaven.eAccount.Geschaeftslogik.DomainException;
import de.awibremerhaven.eAccount.Geschaeftslogik.EntryRule;
import de.awibremerhaven.eAccount.Geschaeftslogik.Status;
import de.awibremerhaven.eAccount.action.admin.ModelException;
public class DSC0 extends DomainState {
    private AccountManager accountManager;
    private EntryRule entryRule;
    public DSC0(
        DomainStateController owner,
        AccountManager am,
        EntryRule er) {
        super(owner);
        this.accountManager = am;
        this.entryRule = er;
    }
    public Signal bearbeite(String accountID)
        throws ModelException, DomainStateException {
        Status antwortStatus =
            this.accountManager.frageNachStatus(
                besitzer.CONTROLLER_ID,
                accountID);
        try {
            if (Status.IN_ORDNUNG.equals(antwortStatus)) {
                accountManager.leseAntrag(accountID);
                besitzer.geheInZustand_DSC1();
                return Signal.IN_ORDNUNG;
            }
            if (Status.IST_GESPERRT.equals(antwortStatus)) {
                return Signal.IST_GESPERRT;
            }
            if (Status.EXISTIERT_NICHT.equals(antwortStatus)) {
                return Signal.EXISTIERT_NICHT;
            }
            if (Status.BEREITS_BEARBEITET.equals(antwortStatus)) {
                return Signal.BEREITS_BEARBEITET;
            }
            besitzer.geheInZustand_DSC4();
            throw new DomainException("Unbekanter Status");
        }
        catch (DomainException e) {
            throw new DomainStateException(e);
        }
    }
    public Signal bearbeitungAbbrechen() {
        return Signal.IN_ORDNUNG;
    }
}
```

AnachZ1.java

```
package de.awibremerhaven.eAccount.action.admin;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import org.apache.struts.action.ActionForm;
import org.apache.struts.action.ActionForward;
import org.apache.struts.action.ActionMapping;
import org.apache.struts.action.ActionMessage;
import org.apache.struts.action.ActionMessages;
import de
    .awibremerhaven
    .eAccount
    .DomainController
    .DomainStateController;
import de.awibremerhaven.eAccount.DomainController.Signal;
public class AnachZ1 extends AdministrationBaseAction {
    public ActionForward execute(
        ActionMapping mapping,
        ActionForm form,
        HttpServletRequest request,
        HttpServletResponse response)
        throws Exception {
        Ereignis ereignis = (Ereignis) form;
        DomainStateController dsc =
            getMyInstanceOfDSC(request.getSession());
        //Änderungsantrag bearbeiten
        if (EreignisZ1.e1 == ereignis.getName()) {
            Signal antwort = dsc.bearbeite(ereignis.getAccountID());
            if (Signal.IN_ORDNUNG.equals(antwort)) {
                return mapping.findForward("/moduleAenderung/vorZ2");
            }
            //Die Bearbeitung wird abgewiesen.
            if (Signal.IST_GESPERRT.equals(antwort)
                || Signal.EXISTIERT_NICHT.equals(antwort)
                || Signal.BEREITS_BEARBEITET.equals(antwort)) {
                //Ein Nachrichtencontainer
                ActionMessages nachrichtUberAbgewieseneBearbeitung =
                    new ActionMessages();
                //Die Nachricht wird erzeugt.
                nachrichtUberAbgewieseneBearbeitung.add(
                    ActionMessages.GLOBAL_MESSAGE,
                    new ActionMessage(antwort.nachrichtenSchluessel));
                /*Der Nachrichtencontainer wird der Anfrage
uebergeben.
                 * wodurch sie von der nächsten JSP dargestellt werden
kann.
                 */
                saveMessages(
                    request,
                    nachrichtUberAbgewieseneBearbeitung);
                return mapping.findForward("/vorZ1");
            }
```

```
            throw new Exception(
                "Ein unbekantes Signal: " + antwort.toString());
        }
        //Neuantrag bearbeiten
        if (EreignisZ1.e2 == ereignis.getName()) { //@TODO
        }
        //Antrag löschen
        if (EreignisZ1.e3 == ereignis.getName()) { //@TODO
        }
        throw new Exception(
            "Ein unbekanntes Ereignis: " + ereignis.toString());
    }
}
```

Signal.java

```
package de.awibremerhaven.eAccount.DomainController;
public class Signal {
    public final String nachrichtenSchluessel;
    private Signal(String nachrichtenSchluessel) {
        this.nachrichtenSchluessel = nachrichtenSchluessel;
    }
    public static final Signal IN_ORDNUNG = new Signal("inOrdnung");
    public static final Signal IST_GESPERRT =
        new Signal("istGesperrt");
    public static final Signal EXISTIERT_NICHT =
        new Signal("existiertNicht");
    public static final Signal BEREITS_BEARBEITET =
        new Signal("bereitsBearbeitet");
}
```

9.0 Abkürzungsverzeichnis

DAO	Data Access Objects
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
JSP	JavaServer Pages
LDAP	Lightweight Directory Access Protocol
MVC	Model View Controller
UML	Unified Modeling Language
XML	Extensible Markup Language

10.0 Literaturverzeichnis

Bal96 Balzert, Helmut: *Lehrbuch der Software-Technik - Software-Entwicklung*. Spektrum Akademischer Verlag, Heidelberg, 1996 .

Bau02 Bauer, Matthias-Livinglogic New Media Solutions (Hrsg.): *Struts Workflow Extension*. 2002, http://www.livinglogic.de/Struts/index.html, letzter Zugriff am: 27.07.04 .

RFC2396 Berners-Lee, T. u.a. : *Uniform Resource Identifiers - Generic Syntax*. 1998, http://www.ietf.org/rfc/rfc2396.txt, letzter Zugriff am: 21.07.04 .

Bien02 Bien, Adam: *J2EE Patterns - Entwurfsmuster für die J2EE*. Addison-Wesley, München, 2002 .

Blo01 Bloch, Joshua: *Effektiv Java programmieren*. Addision-Wesely, München, 2001 .

Bro04 Brown, Don-The Apache Software Foundation (Hrsg.): *Struts Flow*. 2004, http://struts.sourceforge.net/struts-flow/index.html, letzter Zugriff am: 27.07.04 .

Cav02 Cavaness, Chuck: *Programming Jakarta Struts - Building Web Applications with Servlets and JSPs*. O'Reilly, Sebastopol, 2002 .

CaKe03 Cavaness, Chuck; Keeton, Brian: *Jakarta Struts - kurz und gut*. O'Reilly, Köln, 2004 .

Co98 Cooper, James W.: *The Design Patterns - Java Companion*. 1998, http://www.patterndepot.com/put/8/JavaPatterns.htm, letzter Zugriff am: 24.07.04 .

ServletSpec Davidson, James Duncan; Coward, Danny: *Java Servlet Specification - Version 2.2*. 1999, http://java.sun.com/products/servlet/reference/api/index.html, letzter Zugriff am: 14.06.04 .

RFC2616 Fielding, R. u.a.: *Hypertext Transfer Protocol -- HTTP/1.1 - RFC 2616*. 1999, http://www.rfc-editor.org/rfc/rfc2616.txt, letzter Zugriff am: 14.06.04 .

Gam96 Gamma, Erich u.a : *Entwurfsmuster - Elemente wiederverwendbarer objektorientierter Software*. 5. Auflage, Addison Wesley, München, 1996 .

Gua03 Guay, Romain-JavaWorld.com (Hrsg.): *Protect Web application control flow - Java Tip 136*. 2003, http://www.javaworld.com/javatips/jw-javatip136_p.html, letzter Zugriff am: 27.07.04 .

Hoff87 Hoffmann, Hans-Jürgen: *Smalltalk - verstehen und anwenden*. Hanser, München, 1987 .

RFC2109 Kristol, D.; Montulli, L.: *HTTP State Management Mechanism - RFC 2109*.
1997, http://www.rfc-editor.org/rfc/rfc2109.txt, letzter Zugriff am: 14.06.04 .

Mak90 Makedanz, Siegfried: *Programmieren in Smalltalk - Portierung einer Benutzeroberfläche für TeX*.
Hochschule Bremerhaven, Bremerhaven, 1990 .

Mue01 Münz, Stefan: *SELFHTML - HTML-Dateien selbst erstellen*.
2001, http://de.selfhtml.org/, letzter Zugriff am: 14.06.04 .

JspSpec Pelegrí-Llopart, Eduardo; Cable, Larry: *JavaServer Pages Specification - Version 1.1*.
1999, http://java.sun.com/products/jsp/reference/api/index.html, letzter Zugriff am: 14.06.04 .

Reu Reumann, Rick: *Struttin' with Struts*.
o.J, http://www.reumann.net/do/struts/main, letzter Zugriff am: 21.07.04 .

Verm00 Vermeulen, Allan u.a : *The Elements of Java Style*.
Cambridge University Press/SIGS Books, Cambridge, 2000 .

Mag99 o.V.-MageLang Institute (Hrsg.): *Fundamentals of JFC/Swing - Part 2*.
1999, http://java.sun.com/developer/onlineTraining/GUI/Swing2/index.html, letzter Zugriff am: 24.07.04 .

UMLSpec o.V.-Object Management Group (Hrsg.): *OMG Unified Modeling Language Specification - Version 1.5*.
2001, http://www.omg.org/technology/documents/formal/uml.htm, letzter Zugriff am: 21.07.04 .

Struts01 o.V.-The Apache Software Foundation (Hrsg.): *Struts Homepage - Welcome*.
2004, http://jakarta.apache.org/struts/index.html, letzter Zugriff am: 17.06.04 .

Versicherung

Hiermit versichere ich, dass ich die vorliegende Arbeit selbständig und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt habe. Alle Stellen, die wörtlich oder sinngemäß aus veröffentlichten und nicht veröffentlichten Schriften entnommen sind, sind als solche kenntlich gemacht. Die Arbeit hat in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegen.