Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Bachelorarbeit

Roman Geez

Entwicklung einer multikanalfähigen Auftragsverwaltung am Beispiel von Android und Java Server Faces

Fakultät Technik und Informatik
Department Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Roman Geez

Entwicklung einer multikanalfähigen Auftragsverwaltung am Beispiel von Android und Java Server Faces

Bachelorarbeit eingereicht im Rahmen Bachelorprüfung

im Studiengang Angewandte Informatik
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer : Prof. Dr. Ing. Birgit Wendholt
Zweitgutachter : Prof. Dr. sc. pol. Gerken, Wolfgang

Abgegeben am 30.08.2012

Roman Geez

Thema der Bachelorarbeit

Entwicklung einer multikanalfähigen Auftragsverwaltung am Beispiel von Android und Java Server Faces

Stichworte

Android, Java Server Faces, Zeiterfassung, Auftragsverwaltung

Kurzzusammenfassung

Im 21. Jahrhundert werden Menschen immer mobiler. Es kommt öfter vor, dass Selbstständige an einem Tag bei mehreren Kunden arbeiten. In dieser Arbeit wird ein Auftragsverwaltungssystem konzipiert, mit dem man eigene Aufträge und Leistungen verwalten kann. Einige Funktionen wie z.B. Zeiterfassen sollten auf mobilen Geräten verwendbar sein, man sollte auch von Zuhause aus einige Sachen komfortabel erledigen können. Dafür wird das System auch stationär angeboten. Somit wird in dieser Arbeit ein System modelliert, was auf mobilen Geräten und auch auf jedem PC laufen soll. Die Hauptaufgabe dieser Arbeit ist, eine multikanalfähige Anwendung zu konzipieren. Das heißt, dass GUIs von unterschiedlichen Plattformen auf die gleiche Implementierung zugreifen können sollen.

Roman Geez

Title of the paper

Development of a multichannel job management with Android and Java Server Faces

Keywords

Android, Java Server Faces, time tracking, job management

Abstract

In the 21 Century, people are increasingly mobile. It often happens that self-employed persons are working for several customers in a day. In this work, a job management system will be developed, that allows management of jobs and services on your own. Some functions such as Time tracking should be used on mobile devices, you also should be able to do it comfortably from your home. Therefore, the system is offered stationary. Therefor the system in this work is developed to work on mobile devices, as well as on PCs. The main task of this work is to develop a multi-channel-capable application. This means that GUIs from different platforms can access the same implementation.

Inhaltsverzeichnis

Inhaltsverzeichnis 4

1 Einleitung 7

1.1 Motivation 7

1.2 Ziel dieser Arbeit 8

1.3 Aufbau dieser Arbeit 9

2 Grundlagen 10

2.1 Android 10

2.1.1 Lebenszyklus einer Activity 10

2.1.2 View (Android Layout) 11

2.1.3 Listener 12

2.1.4 GUI-Steuerung 14

2.2 Java Server Faces (JSF) 15

2.2.1 Lebenszyklus einer Java Server Faces-Seite 15

2.2.2 Managed-Beans 16

2.2.3 GUI-Steuerung 16

2.2.4 Listener 17

2.2.5 View (JSF-Seite) 18

2.3 Vergleich 19

2.3.1 Lifecycle Steuerung, die unterschiedlichen Zwecken dienen 19

2.3.2 Navigationsregelung (GUI-Steuerung) 19

2.3.3 Views 19

2.3.4 Listener 19

2.4 Zusammenfassung 20
3 Analyse 21
3.1 Einführung in die Anwendungsfälle 21
3.2 Use-Case 23
3.2.1 Liste der AG 23
3.2.2 Neuen AG registrieren 25
3.2.3 AG bearbeiten 26
3.2.4 Rechnungsübersicht 27
3.2.5 Auftragsliste 28
3.2.6 Auftragsdetails 29
3.2.7 Auftrag stornieren 30
3.2.8 Auftrag abschließen 31
3.2.9 Auftrag erstellen 32
3.2.10 Leistungsübersicht 33
3.2.11 Leistung hinzufügen 34
3.2.12 Leistungsdetails 35
3.2.13 Leistung bearbeiten 36
3.2.14 Rechnung erstellen 37
3.2.15 Arbeitszeit erfassen 38
3.2.16 Leistung stornieren 39
3.3 Zusammenfassung 39
4 Design 40
4.1 System Architektur 40
4.2 Designentscheidungen 41
4.2.1 MVC-Konzept 41
4.2.2 Schnittstellen zur Persistenzschicht 42
4.2.3 Factory-Pattern 43
4.2.4 Transportobjekte 43
4.2.5 Parametrisierte Methoden 43
4.2.6 Stateless Services 44
4.3 Softwarearchitektur 44
4.3.1 Logik-Komponente im Detail 44

4.3.1.1. Zustandsdiagramme 46
4.3.1.2. Sequenzdiagramme 47
4.3.2 DB-Adapter im Detail 51
4.3.3 Persistenzschicht. Realisierung für Android 53
4.3.4 Klassendiagramm der mobilen Anwendung 54
4.4 Entwurfsentscheidungen für eine multikanalfähige Anwendung 56
4.4.1 Plausibilitäts- und Vollständigkeitsprüfung auf Benutzereingaben 56
4.4.2 Model-Objekte als Transportobjekte für Benutzereingaben 56
4.4.3 DB-Adapter 56
4.5 Zusammenfassung 57
5 Zusammenfassung und Ausblick 58
5.1 Zusammenfassung 58
5.2 Ausblick 59
6 Literatur 60

1 Einleitung

1.1 Motivation

Wir werden immer mobiler und sind oft unterwegs und beschäftigt. Es ist noch gar nicht lange her, dass man sehen konnte, wie Menschen in öffentlichen Verkehrsmitteln Arbeitsunterlagen handschriftlich bearbeitet haben. Heute sind die meisten Menschen mit einem mobilen Gerät ausgestattet und viele Aufgaben werden mit deren Hilfe erledigt.
Mit zunehmender Mobilität wächst auch die Nachfrage nach mobilen Geräten wie nie zuvor. Gleichzeitig werden immer mehr Funktionen für Smartphones angeboten und genutzt. Außerdem werden derzeit relativ günstige Flatrates angeboten, die eine andauernde Verbindung zum Internet erlauben. Deswegen entscheiden sich die meisten Käufer heutzutage bereits für ein Smartphone, welches mit seiner enormen Verbreitung die klassischen Handys mittlerweile abgelöst hat. Durch die große Auswahl an verschiedensten Smartphones, findet jeder ein für seine Bedürfnisse passendes Gerät.
Viele Aufgaben, für die man noch vor wenigen Jahren einen Rechner brauchte, können heute mit einem mobilen Gerät erledigt werden.
Da die Daten der meisten Applikationen (APPs) lokal auf dem Gerät gespeichert werden, wird die Zusammenarbeit mehrerer Clients mit denselben Daten weitreichend und stark beeinträchtigt. Aus diesem Grund findet ein deutlich zunehmendes Wachstum für internetbasierte Anwendungen statt.
Google hat eine internationale Statistik (siehe Abb. 1.1) über die Nutzung von verschiedenen Devices erhoben. Für diese Statistik wurden jeweils 2000 Menschen aus den jeweiligen Ländern befragt. In Deutschland werden Smartphones von 73%-76% benutzt, 49%-51% nutzen einen PC und 46%-49% arbeiten mit einem Laptop.
Wie man aus der Statistik entnehmen kann, arbeiten immer noch recht viele an einem Rechner oder einem Notebook. Bevorzugt werden dabei eine bequeme Tastatur und ein größerer Bildschirm, denn beides bietet einen höheren Bedienkomfort als heutige Smartphones. Ein anderer Vorteil der Rechner und Laptops ist das problemlose Sharing von Daten untereinander.
Ebenso gibt es Anwendungen mit höherem Performance-Bedarf. Für diesen Fall wäre es nicht sinnvoll sie auf einem Smartphone laufen zu lassen.

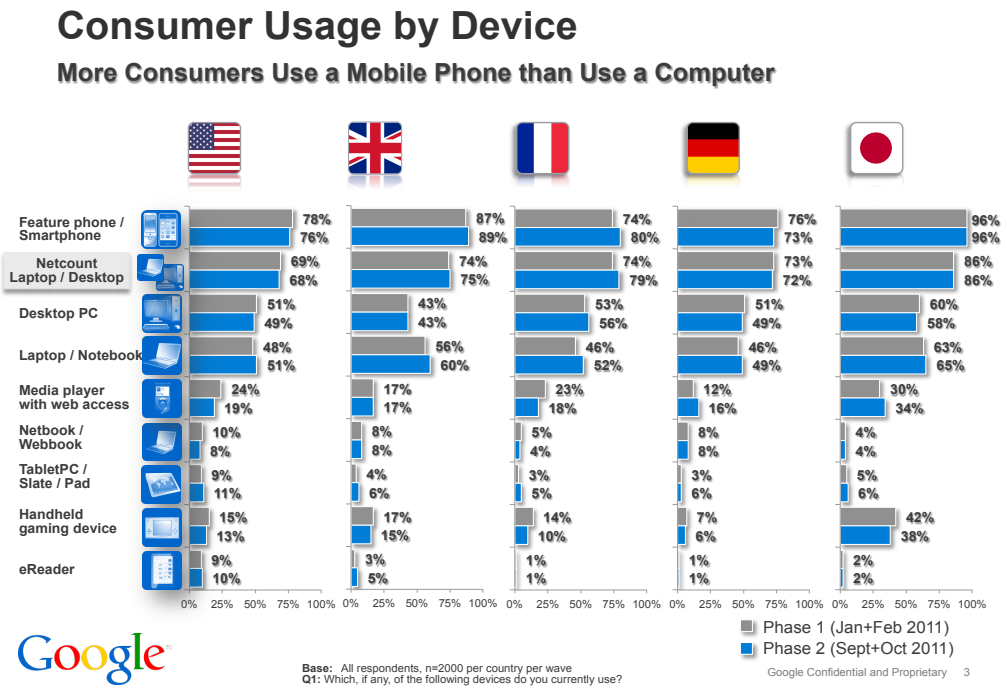

Abbildung 1.1: Google Statistik [GS 2011]

Aus den genannten Statistiken lässt sich entnehmen, dass sowohl Smartphones als auch Notebooks und Rechner ihre eigenen Vorteile bieten. Man benutzt das Gerät, mit dem es gerade bequemer zu arbeiten ist und erwartet dabei in allen Umgebungen gleiche Funktionalität.
Infolgedessen werden immer mehr Anwendungen entwickelt, die auf mehreren Plattformen laufen können und gemeinsame Daten synchronisieren. Das bietet dem Benutzer sehr viel Flexibilität und spart so einige Arbeitswege.

1.2 Ziel dieser Arbeit

In dieser Arbeit soll ein Konzept für eine auftragsverwaltende Anwendung entwickelt werden, welche Selbständige, kleine und mittlere Firmen unterstützen kann. Es ermöglicht einem Auftragsnehmer neue Aufträge oder Leistungen ins System einzutragen, Arbeitszeit zu erfassen, Rechnungen für einen Auftrag oder einzelne Leistungen zu erstellen und diese per E-Mail an den Auftragsgeber zu versenden. Je nach Tätigkeit ist es oft bequemer die Arbeitszeit mit einem Smartphone zu erfassen, dennoch wird ein größerer Bildschirm bei

einer Rechnungserstellung bevorzugt. Um das zu erreichen wird die Anwendung so konzipiert, dass sie auf mehreren Plattformen lauffähig ist, sodass man unterwegs auf einem Smartphone mit dem System arbeiten kann, bzw. an einem Arbeitsplatz über einen Browser auf die Daten zugreifen kann.

In dieser Arbeit wird ein Entwurf präsentiert, welcher so generisch ist, dass er auf mehreren Plattformen abbildbar ist (Android, Web Applikation).
Es wird viel Wert auf die Wiederverwendbarkeit von Hauptkomponenten gelegt.
Des Weiteren soll auch möglich sein, mit für die jeweiligen Plattformen typischen Datenbanken zu arbeiten, sowohl lokal als auch entfernt. Zum Beispiel wird für Android die integrierte SQLite Datenbank eingesetzt. Für die Webapplikation kann man z.B. eine Oracle Datenbank verwenden. Natürlich muss man sich dabei um die Synchronisierung der Datenbanken kümmern, dies ist aber kein Bestandteil dieser Arbeit.

1.3 Aufbau dieser Arbeit

Im Kapitel 2 werden Grundlagen von Android und Java Server Faces vorgestellt. Dabei werden wesentliche Komponenten von diesen beiden Welten beschrieben. Darüberhinaus werden auch Komponenten und Funktionen miteinander verglichen, die ähnliche Aufgaben haben.
Das Kapitel 3 stellt die ersten Schritte der Softwareentwicklung vor. Dazu wird ein Use-Case Diagramm erstellt, das die einzelne Anwendungsfälle und deren Zusammenhang verdeutlicht. Die Anwendungsfälle werden in Tabellenform beschrieben.
Im Kapitel 4 wird basierend auf die Anwendungsfälle das Design entwickelt. Dabei werden technische Aspekte von Android und JSF vorgestellt und verglichen. Es werden auch Komponenten und Funktionen detailliert beschrieben, die nur eine der beiden Plattformen anbietet. Außerdem wird auch beschrieben, wie fehlende Komponenten und Funktionen für die andere Plattform beschafft werden.
Eine kurze Zusammenfassung dieser Arbeit mit einem Ausblick und möglichen Erweiterungen ist abschließend im Kapitel 5 zu lesen.

2 Grundlagen

In diesem Kapitel werden die beiden Programmiermodelle, auf die das Design abgestimmt wird, vorgestellt. Dabei wird auf Konzepte, die im Design eingeführt werden, im Vorfeld in Bezug genommen.

2.1 Android

Android ist ein Open-Source Betriebssystem von Google für mobile Geräte wie Smartphones oder Tabs. Es basiert auf Linux. Die Plattform ist frei verwendbar und änderbar und es gibt schon viele Millionen Geräte, die auf Android basieren. Entwickler können neue Apps programmieren und über „Android Market“ veröffentlichen.
Google stellt ein Android Plugin für Eclipse (ADT Plugin) bereit. Eine detaillierte Beschreibung zur Installation findet man auf Android Developer Seite [ADG 2012].
Die Android Plattform hat eine integrierte Datenbank SQLite. Dadurch wird ermöglicht, Daten nicht nur in Datei-Form zu speichern, sondern auch bequem in die Datenbank zu schreiben, ohne dass die Entwickler eine Datenbank installieren und einrichten müssen.
Ein Benutzer kann ein Android-Gerät mittels Touchscreen und entsprechenden Hardwaretasten steuern.

2.1.1 Lebenszyklus einer Activity

In Abbildung 2.1.1 ist ein Lebenszyklus einer Activity dargestellt. Eine Activity hat 3 mögliche unterschiedliche Zustände. Wenn eine Activity gestartet und im Vordergrund ist, ist sie im Zustand *running*. Wird eine weitere Activity nicht im Vollbildmodus gestartet, ist sie im Zustand *pausiert*.
Der Zustand einer Activity wird in den Zustand *pausiert* gesetzt, sobald eine weitere Activity nicht im Vollbildmodus gestartet wird. Das hat zur Folge, dass die erste Activity nicht mehr im Vordergrund aber immer noch sichtbar ist. Wenn eine Activity nicht mehr sichtbar ist, wird sie *gestoppt*.

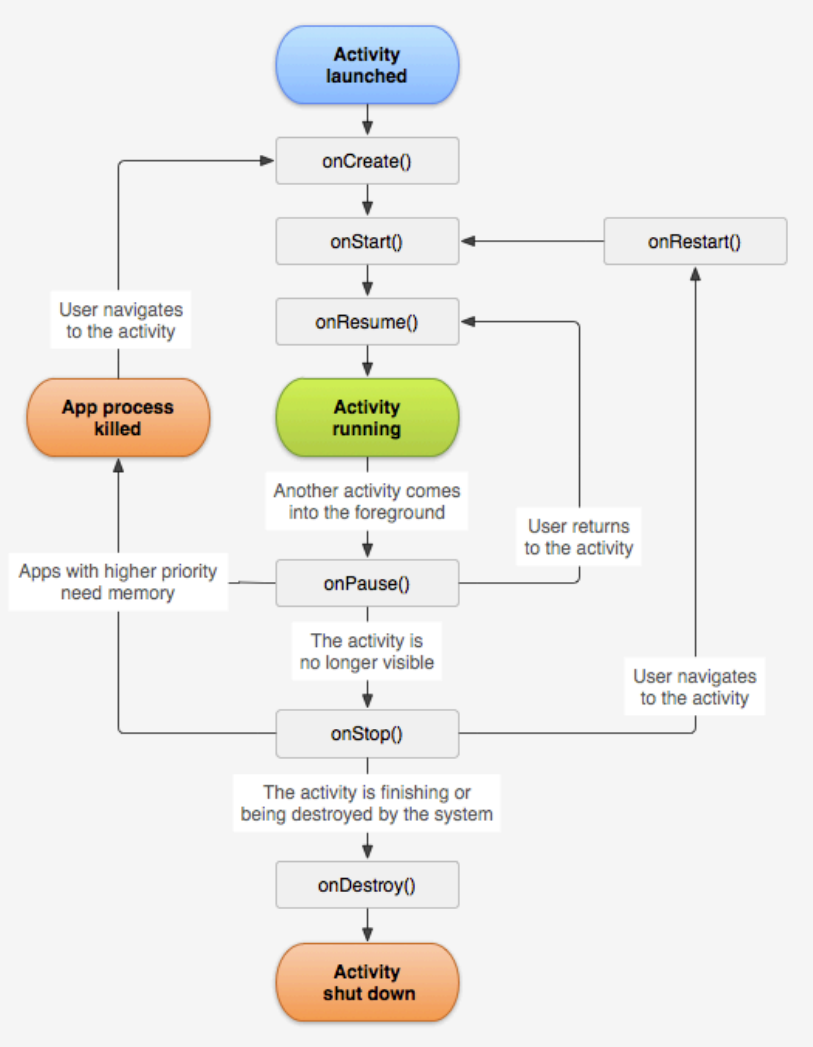

Abbildung 2.1.1 Lebenszyklus einer Activity [ADG 2012]

2.1.2 View (Android Layout)

Jedes an der Oberfläche sichtbare Element ist von der Klasse *android.view.View* abgeleitet. Views in Android sind Eingabefelder, Schaltflächen, Auswahllisten usw. Layouts fassen die Views zusammen. Die Layouts sind in Android vorgegeben, z.B. lineare, tabellarische usw. Wie man aus den Namen erkennen kann, fasst ein lineares Layout die Views linear (horizontal oder vertikal), ein tabellarisches in eine Tabelle usw. zusammen. Layouts können ineinander geschachtelt werden.

```
<TableLayout
          xmlns:android="http://schemas.android.com/apk/res/android"
          android:layout_width="match_parent"
          android:layout_height="wrap_content" >
      <LinearLayout
           android:layout_width="wrap_content"
           android:layout_height="wrap_content"
           android:orientation="vertical" >
           <TextView
               android:id="@+id/textView234"
               android:layout_width="wrap_content"
               android:layout_height="wrap_content"
               android:text="@string/Strasse" />
           ...
       </LinearLayout>
       ...
</TableLayout>
```

Jedes Layout und jede View muss die Attribute `android:layout_width` und `android:layout_height` besitzen. Aus Performance-Gründen soll man möglichst geschachtelte Views vermeiden. Jede View hat eine ID, über diese ID werden Views in einer Activity angesprochen. Bei Bedarf kann man über IDs in die Views Werte eintragen lassen, bzw. wiederabrufen. Alle Layouts werden in Form einer XML-Datei im Ordner */res/layout* abgelegt.

2.1.3 Listener

Wie gerade beschrieben wurde, bestehen Android Bildschirmseiten aus View-Elementen. Interaktionen des Benutzers mit Views lösen Ereignisse auf. Dafür wird ein Code geschrieben, der beim Klicken auf View (z.B. Button) ausgeführt werden soll. Jetzt gibt es zwei Möglichkeiten vorzugehen:

1. Es wird eine Methode (z.B. *onButtonClick*) in der Activity-Klasse definiert:

```
public void onButtonClick(View view){
         switch (view.getId()){
         case R.id.agAendern_AGspeichern:
               // Unser Code für speichern
         case R.id.agAendern_AGändern:
               // Unser Code für ändern
         }
}
```

   Die Methode bekommt eine View als Parameter übergeben. Mit Hilfe von Switch-Case Block und Views-ID kann man festlegen, welches Ereignis beim Klicken ausgelöst werden soll. Jetzt muss die Methode nur bei Views registriert werden. Das wird mit dem Attribut `android:onClick` gemacht:

```
<Button
            android:id="@+id/agAendern_AGspeichern"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:onClick="onButtonClick"
            android:text="@string/Speichern" />
```

Beim Klicken auf den Button *speichern* wird die Methode *onButtonClick* aufgerufen. Als Parameter wird eine View mit ID ="@+id/agAendern_AGspeichern" übergegeben. In dieser Methode wird der Case-Block ausgeführt.

2. Es wird eine Klasse erzeugt, die ein Listener-Interface (z.B. *OnClickListener*) implementiert. In dieser Klasse muss eine Methode *void onClick(View v)* definiert sein:

```
class AGSpeichern implements OnClickListener{
      public void onClick(View v){
            //Unser Code
      }
}
```

Meistens wird diese Klasse als eine innere Klasse der Activity-Klasse definiert. Als nächstes muss man ein Objekt der Listener-Klasse erstellen und bei View registrieren. Das passiert in der *onCreate()* Methode der Activity-Klasse:

```
AGSpeichern ags = new AGSpeichern();
Button agspeichern = (Button) findViewById(R.id.
                                        agAendern_AGspeichern);
agspeichern.setOnClickListener(ags);
```

Das ags ist hier das Listener-Klasse-Objekt. Mit Hilfe von `findViewById` schafft man eine Referenz auf das Button-Element. Zuletzt registriert man das Listener-Objekt.

Neben dem gerade beschrieben *„onCklick()“*-Ereignis gibt es viele andere Ereignisse, welche mit jeweiliger Beschreibung in der Abbildung 2.1.3 vorgestellt werden.

Ereignismethode	Beschreibung
`void onClick(View v)`	Wird bei Klickoperationen aufgerufen. Interface: `OnClickListener` Registriermethode: `setOnClickListener()`
`boolean onDrag(View v, DragEvent e)`	Wird bei Drag&Drop-Operationen aufgerufen. Interface: `OnDragListener` Registriermethode: `setOnDragListener()` Mithilfe der `getAction()`-Methode des `DragEvent`-Parameters können Sie abfragen, um welche Drag&Drop-Aktion es sich handelt (also beispielsweise den Beginn einer Zieh-operation oder das Ablegen des gezogenen Objekts).

Ereignismethode	Beschreibung
`boolean onLongClick(View v)`	Wird aufgerufen, wenn ein View-Element gedrückt gehalten wird. Interface: `OnLongClickListener` Registriermethode: `setOnLongClick-Listener()`
`void onFocusChange(View v, boolean f)`	Wird bei Änderung des Fokus aufgerufen. Interface: `OnFocusChangeListener` Registriermethode: `setOnFocusChange-Listener()` Der Parameter `f` gibt den neuen Fokuszustand an (`true` bedeutet, das View-Element hat den Fokus erhalten).
`boolean onKey(View v, int taste KeyEvent e)`	Wird bei Drücken einer Taste aufgerufen, vorausgesetzt das betreffende UI-Element besitzt den Tastaturfokus. Interface: `OnKeyListener` Registriermethode: `setOnKeyListener()` Der Parameter `taste` gibt den Tastencode der gedrückten Taste an und hilft bei deren Identifizierung. Mithilfe des `KeyEvent`-Parameters können Sie detailliertere Informationen über den Tastendruck abfragen (siehe weiter unten den Abschnitt »Auf Tipp- und Wischereignisse reagieren«).
`boolean onTouch(View v, MotionEvent e)`	Wird bei Tipp-Operationen aufgerufen. Interface: `OnTouchListener` Registriermethode: `setOnTouchListener()` Im `MotionEvent`-Parameter sind ausführliche Informationen über die Tipp-Operation gespeichert (siehe unten).
`onCreateContextMenu()`	Zur Erstellung von Kontextmenüs (siehe *Kapitel 10.1.4*).

Abbildung 2.1.3 Android-Listener [Andn2011] S.183

2.1.4 GUI-Steuerung

Wie die Abbildung 2.1.1 zeigt, wird beim Start einer Activity die Methode *onCreate()* aufgerufen. In dieser Methode wird ein Layout ausgewählt, das auf dem Bildschirm dargestellt wird. Beim Auslösen eines Ereignisses, welches zu einer neuen Activity führen soll, muss die Methode *startActivity()* oder *startActivityForResult()* in dem zugehörigen Eventhandler aufgerufen werden. Dabei wird die aktuelle Activity pausiert. Der Unterschied zwischen diesen beiden Methoden ist, dass die *startActivityForResult()* einen Rückgabewert liefert. Beiden Methoden haben als Parameter *Intent*. Wenn ein Datenaustausch zwischen

zwei Activities stattfinden soll, passiert das über dieses *Intent*: *intent.putExtra(„String“, Daten).* Die Funktionsweise ist gleich der bei Java-Maps. Als String wird ein Key übergeben, über den die aufgerufene Activity auf *Data* zugreifen kann: *int neuInteger = extras.getInt("String");*

2.2 Java Server Faces (JSF)

Seit mehreren Jahren gibt es webbasierte Anwendungen. Java Server Faces ist ein Framework für die Serverseitige Entwicklung von Java-Web-Anwendungen. Bei der Entwicklung von GUIs hat sich das MVC-Entwurfsmuster durchgesetzt, welches noch im Kapitel 4.2.1 detailliert beschrieben wird. JSF erbt die Eigenschaften des HTTP-Protokolls. Hierbei wird viel Wert darauf gelegt, das Request-Response-Model besser nutzen zu können. Unter anderem wird die Zustandslosigkeit von HTTP mit Hilfe von dynamischen JSF-Seiten versteckt.

2.2.1 Lebenszyklus einer Java Server Faces-Seite

Nach [JSF 2004] hat Andry Bosch den Lebenszyklus einer JSF-Seite beschrieben. Eine JSF-Seite ist zunächst eine JSP-Seite. Somit kann man sagen, dass eine JSF-Seite, auch wie eine JSP-Seite, sieben Phasen hat:

1. Seite übersetzen
2. Seite kompilieren
3. Seite laden
4. Instanz erzeugen
5. Seite initialisieren
6. Service-Routine
7. Servlet entfernen

Allerdings bietet die JSF-Seite viel mehr Funktionen an. Es gibt 2 Arten von Requests und 2 Arten von Responses:

- Faces Requests
- Non-Faces Requests
- Faces Response
- Non-Faces Response

Ein Faces Request heißt, dass die Anfrage von einer JSF-Seite kam. Non-Faces Requests heißt, dass die Anfrage nicht von einer JSF-Seite kam (bspw. von einer JSP-Seite oder einer statischen HTML-Seite). Ähnlich sieht es mit Responses aus. Wenn der Server eine Seite generiert, die JSF-Funktionalität benötigt, dann ist das eine Faces Response, ansonsten eine Non-Faces Response. In den meisten Fällen löst ein Faces Request eine Faces Response aus. Die Abbildung 2.2.1 stellt den Standard-Lebenszyklus eines Requests dar. Mit den gestrichelten Linien sind mögliche Alternativen des Ablaufes dargestellt.

In der Phase *Restore View* wird das View Objekt aus FacesContext wiederhergestellt. Bei *Apply Request Values* werden Werte aus dem Request rausgeholt und die entsprechende Datenkonvertierung durchgeführt. Danach kommt die *Phase Process Validation*. Dabei werden die Werte auf Gültigkeit geprüft. In der *Phase Update Model Values* werden die

Werte in das Modellobjekt geschrieben. In der nächsten *Phase Invoke Application* passiert die Weiterleitung auf die nächste (aufgerufene) Seite. Die letzte Phase ist die *Render Response Phase*. Da findet das Rendering einzelner Komponenten statt. Aus diesen Komponenten wird eine neue Seite erzeugt.

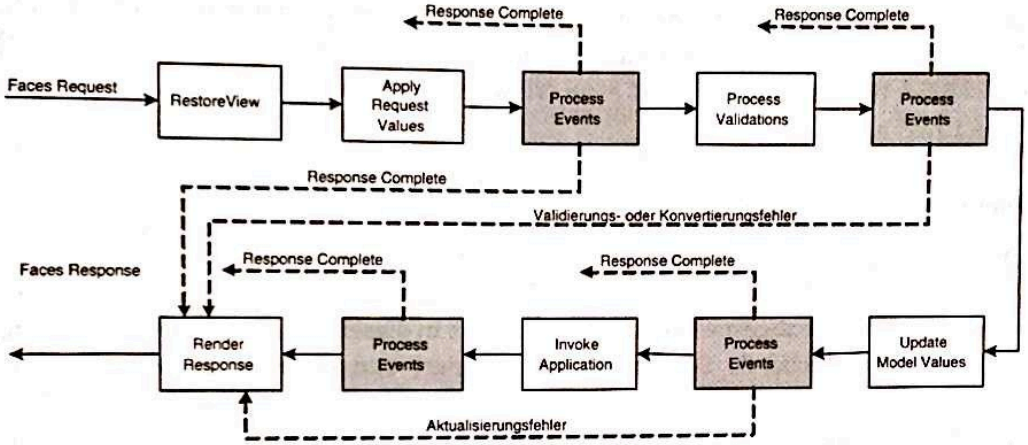

Abbildung 2.2.1: Lebenszyklus eines Requests [JSF 2004] Seite 69.

2.2.2 Managed-Beans

Wie gerade erwähnt, werden in der *Phase Update Model Values* Werte in das Modellobjekt geschrieben. Als Modellobjekte dienen hier die Java-Beans, die in JSF als Managed-Beans deklariert werden. Dafür müssen sie in die Anwendungskonfigurationsdatei *faces-config.xml* hinterlegt werden. Die Datei wird beim Start des Applikationsservers eingelesen. Danach können JSF-Seiten auf diese Beans zugreifen. JSF kümmert sich um die rechtzeitige Bereitstellung der Objekte.

Eine Deklaration in dem Application Configuration File kann z.B. so aussehen:

```
<managed-bean>
        <managed-bean-name>Name</managed-bean-name>
        <managed-bean-class>
                de.romangeez.auftragsverwaltung.Auftraggeber
        </managed-bean-class>
        <managed-bean-scope>session</managed-bean-scope>
</managed-bean>
```

Das deklarierte Bean wird innerhalb einer Session zur Verfügung gestellt. In JSF-Seiten ist das Bean unter den Namen „Auftraggeber“ (was zwischen `<managed-bean-name>` und `</managed-bean-name>` vergeben wurde) zugreifbar. Sollte eine Session ungültig werden, wird eine neue Instanz erstellt. Somit wird nie ins „Leere“ zugegriffen.

2.2.3 GUI-Steuerung

Navigation in JSF basiert auf Actions. Beim Klicken auf einen Button oder einen Link werden Actions ausgelöst. Es gibt 3 Faktoren, die auf die Navigation Einfluss haben:

1. Aktuelle Seite
2. Aktion, die durch Anklicken ausgelöst wurde
3. Der Rückgabewert von der Actionmethode.

Man kann in der Anwendungskonfigurationsdatei *faces-config.xml* alle Regeln definieren. Sie beschreibt unter anderem, zu welcher Seite navigiert werden soll, ausgehend davon von welcher Seite der Aufruf kommt und je nach dem, was für einen Rückgabewert die Actionmethode liefert. Man kann aber auch in den jeweiligen „JSF-Seiten“ die Regeln definieren. Dadurch wird aber die Erweiterbarkeit dieser Anwendung aufwändiger.
Eine einfache Navigationsregelung in der Konfigurationsdatei ist so aufgebaut:

```
<navigation-rule>
	<from-view-id>/seite1.jsp</from-view-id>
	<navigation-case>
		<from-outcome>success</from-outcome>
		<to-view-id>/seite2.jsp</to-view-id>
	</navigation-case>
</navigation-rule>
```

Hier sieht man, dass sobald auf der „seite1.jsf“ beim Klicken auf den Button die Action-Methode den Rückgabewert „success“ liefert, wird das JSF zur „seite2.jsf“ navigieren.

2.2.4 Listener

Listener im JSF haben den gleichen Zweck wie Listener in Android, sind aber ganz anders aufgebaut. Es gibt zwei Arten von Action-Listenern. Eine wurde bereits im Kapitel 2.2.3 vorgestellt. Eine Art der Action-Methoden hat einen Rückgabewert und steuert die Navigation. Die andere Art der Action-Methoden hat keinen Rückgabewert und wird für den Zugriff auf den Action-Event und auf die vom Action-Event erzeugte Komponente verwendet, hat aber keine Auswirkung auf die Navigation.
Für alle Eingabekomponenten wird ein Value-Change-Event erzeugt. Mit Hilfe von Value-Change-Events lässt sich prüfen, ob sich bei den Eingaben was geändert hat. Wenn die Eingaben sich nicht geändert haben, kann man sich zum Beispiel den Schreibzugriff auf die Datenbank sparen.

2.2.5 View (JSF-Seite)

Views in JSF sind die JSF-Seiten. Diese werden von Entwicklern einzeln programmiert. Um besser zu verstehen, wie eine JSF-Seite aufgebaut ist, wird in der Abbildung 2.2.5 ein Beispiel dargestellt.

```
<html>
  <head>
    <title>Hallo Welt</title>
  </head>

  <body>
  <h3>Hallo-Welt-Beispiel</h3>
  <i>Dieses Beispiel zeigt ein erstes kurzes JSF-Beispiel</i>
  <br><br>
  Bitte geben Sie hier Ihre Daten ein:
  <f:view>
    <h:form>
      Vorname: <h:inputText value="#{Visitor.firstname}" /><br>
      Nachname: <h:inputText value="#{Visitor.lastname}" /><br>
      Geburtsdatum:
        <h:inputText value="#{Visitor.birth}">
          <f:convertDateTime dateStyle="short" type="date" />
        </h:inputText>
      <br><br>
      <h:commandButton action="success" value="Submit" />
    </h:form>
  </f:view>
</body>
</html>
```

Abbildung 2.2.5 JSF-Seite Beispiel. [JSF 2004] S.86

Auf dieser Seite wird ein Formular dargestellt, welches man mit Vor- und Nachnamen sowie dem Geburtsdatum ausfüllen kann. Hier sieht man auch, wie eine JSF-Seite auf Managed-Beans zugreift.

Mit <h:inputText> wird ein Eingabefeld definiert. Der Value-Wert beschreibt, von welchem Modelobjekt der Wert für dieses Feld genommen werden soll. Beim Klicken auf den Button werden die eingegebenen/geänderten Werte in das Modellobjekt geschrieben. Wie man sieht, wird nirgendwo auf der JSF-Seite ein Objekt „Visitor“ erstellt. Wie schon im Kapitel 2.2.2 beschrieben wurde, werden Managed-Beans in der Konfigurations-Datei deklariert. Somit sorgt das JSF, dass die Beans bereitgestellt werden.

2.3 Vergleich

2.3.1 Lifecycle Steuerung, die unterschiedlichen Zwecken dienen

JSF hat die Aufgaben

1. das Zustandslose HTTP Protokoll zu „verstecken“.
2. Benutzereingaben weit möglichst über eine Session zu erhalten.

Android hat im Gegensatz dazu die Aufgaben

1. die Kontrolle über die Aktivierung von Komponenten und Applikationen zu übernehmen, um priorisierte Funktionalitäten (z.B. Anrufe) immer sicher zu stellen.
2. Zustandsübergänge von Android zu erzwingen: Wenn Aktivitäten den Zustand behalten wollen, dann müssen sie diesen in geeigneten Methoden `onSaveInstanceState(Bundle savedInstanceState)` aktiv schreiben.

Der Zustand einer JSF-Seite und Managed-Beans ist, so lange die Anwendung läuft, sichergestellt. Bei Android ist es genau umgekehrt. Man muss immer damit rechnen, dass eine Aktivität oder eine Komponente nicht genügend Ressourcen hat. Insbesondere bedeutet die Auswirkung des Lifecycle-Modells in Android mehr Entwicklungsaufwand um den Zustand einer Applikation zu retten.

2.3.2 Navigationsregelung (GUI-Steuerung)

Die Navigation auf beiden Plattformen ist sehr ähnlich aufgebaut. Beim Aufruf einer neuen Seite bei JSF wird eine Action ausgelöst, die zu einer neuen JSF-Seite führt. Beim Android wird eine neue Activity aufgerufen, die eine neue View darstellt. Aber in JSF gibt es eine deklarative Formulierung der Navigation über Navigationsregeln. In Android wird dieses programmatisch als Implementierung von Listenern gelöst.

2.3.3 Views

Obwohl die GUI-Steuerung in beiden Welten sehr ähnlich ist, ist die Implementierung der GUI sehr unterschiedlich. Daraus ergibt sich, da für beide Plattformen GUIs-Seiten programmiert werden müssen, ein erhöhter Entwicklungsaufwand.

2.3.4 Listener

Die Funktionsweise der Listener ist in beiden Welten sehr ähnlich aufgebaut, sie müssen aber genauso wie Views für die beiden Plattformen implementiert werden. Views und Listener gehören beide zur GUI-Komponente. Listener von beiden Plattformen sollen so implementiert werden, dass sie mit der Logik-Komponente interagieren können.

2.4 Zusammenfassung

Für die Multikanalfähigkeit ist daher im Designkapitel eine geeignete Abstraktion beim Entwurf einer Logikkomponente zu finden. Des weiteren sollten Transportobjekte zwischen Komponenteninhalten und der Logikkomponente zur Verfügung stehen, die dem Konzept der Managed-Beans in JSF entsprechen.
Die Implementierung der GUI-Komponente sowie der Navigationsteuerung muss für beide Welten parallel erfolgen, da die technische Lösung sehr unterschiedlich ist.

3 Analyse

In diesem Kapitel werden die Anwendungsfälle beschrieben. Dabei werden die Anforderungen an das System Auftragsverwaltung analysiert und spezifiziert. Dabei wird ein Use-Case Diagramm mit Hilfe der standardisierten Modellierungssprache für Software (UML) modelliert, welche die Anforderungen, Beziehungen und Zusammenhänge sehr gut darstellt und verdeutlicht.
Um die gesamte Problemstellung besser zu erfassen, wird die Komplexität so gering wie möglich gehalten. Dafür wird versucht, technische Aspekte zu vermeiden. Diese werden im nächsten Kapitel ausführlich beschrieben.

3.1 Einführung in die Anwendungsfälle

Zum besseren Verständnis der im Folgenden vorgestellten Anwendungsfälle werden im Vorfeld die wesentlichen Begriffe eingeführt.

Rollen
Es gibt 2 Arten von Benutzern:
Auftragsnehmer, im Folgenden AN
Auftragsgeber, im Folgenden AG

Zugangsvoraussetzungen
Jeder Benutzer muss im System registriert sein und seine Zugangsdaten kennen.
Nur ein AN kann einen neuen AG registrieren und den Zugang zum System freischalten.
Ein Benutzer kann das System nur benutzen, nachdem er sich im System mit seinen Benutzernamen und Passwort angemeldet hat.

Entitäten
Auftrag kann vier unterschiedliche Status haben

Status	**Beschreibung**
Offen	Der Auftrag ist mit dem AG abgesprochen
Storniert	Wenn der AN den Auftrag storniert hat
Abgeschlossen	Alle Leistungen wurden erbracht
Bezahlt	Alle Leistungen des Auftrags sind bezahlt

Leistung ist im Rahmen eines Auftrags erbrachte Arbeit, verbunden mit einem Zeitraum. Leistungen haben 5 unterschiedliche Status:

Status	**Beschreibung**
Offen	Die Leistung und der Termin sind mit dem AG abgesprochen.
Erbracht	Sobald die Arbeitszeit erfasst wurde, wird der Status automatisch auf „Erbracht“ geändert: Die Leistung ist bereits erbracht, es wurde aber noch keine Rechnung dafür erstellt
In Rechnung	Für die Leistung wurde eine Rechnung erstellt
Bezahlt	wurde vom AG bezahlt
Storniert	wenn der AN die Leistung storniert hat.

Rechnung kann 2 Status haben

Status	**Beschreibung**
Offen	Die Rechnung wurde erstellt, ist aber noch nicht bezahlt
Bezahlt	Die Rechnung wurde bezahlt

Filterfunktion

Um dem Benutzer einen besseren Bedienkomfort zu ermöglichen, bietet das System für die Leistungsübersicht, Auftragsliste und Rechnungsübersicht eine Filterfunktion an.

nach Status	
nach AG	
nach Zeit	Tage
	seit
	von bis (Intervall)
	Monat
	Jahr

3.2 Use-Case

Abbildung 3.2 gibt einen Überblick über alle für die Applikation relevanten Use-Cases, die nachfolgend in Details erläutert werden.

3.2.1 Liste der AG

Akteur:
AN
Ziel:
Eine Übersicht von AG bekommen, evtl. einen AG bearbeiten oder einen neuer AG registrieren
Auslöser:
AN
Vorbedingungen:
Mit dem APP (mobiles Gerät) und über Webseite möglich
Nachbedingungen:
Der AN hat Übersicht von AG bekommen
Erfolgsszenario:
1. AN geht auf Liste der AG 2. Das System zeigt dem AN eine Liste von AG 3. Der AN kann auf „Bearbeiten“ gehen (Anwendungsfall AG bearbeiten) 4. Der AN kann einen neuen AG registrieren (Anwendungsfall neuen AG registrieren)

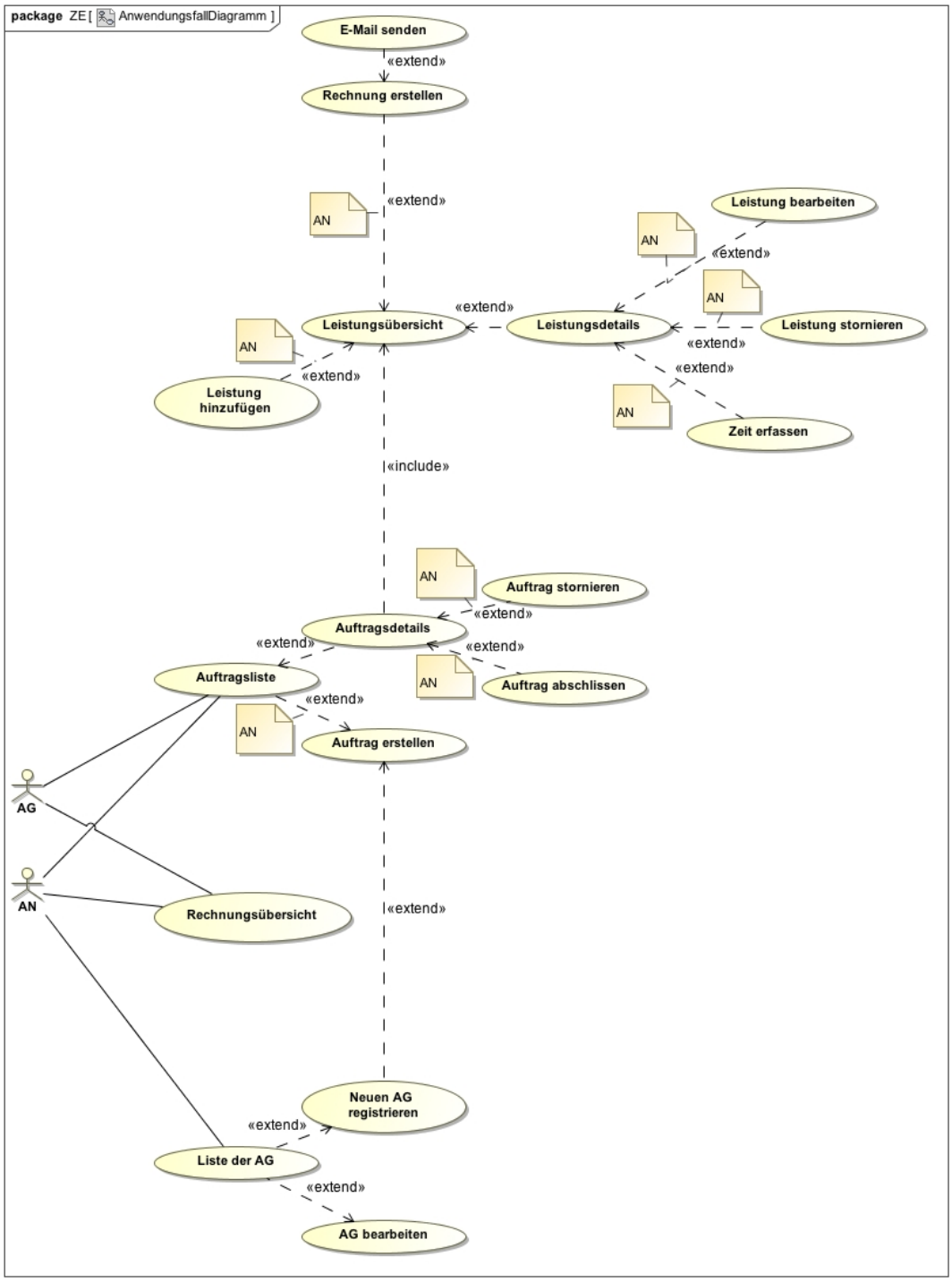

Abbildung 3.2: Use-Case Diagramm

3.2.2 Neuen AG registrieren

Akteur:
AN
Ziel:
Einen neuen AG in das Auftragsverwaltungssystem eintragen
Auslöser:
AN fügt einen neuen AG hinzu
Vorbedingungen:
Benötigte Daten des AG sind bekannt Der Anwendungsfall erweitert den Anwendungsfall „Neuer Auftrag“ und „Liste der AG“ Mit dem APP (mobiles Gerät) und über Webseite möglich
Nachbedingungen:
Der AG wird ins System eingetragen. Es wird evtl. auch ein neuer Account erstellt
Erfolgsszenario:
1. Das System zeigt dem AN das Formular für neuen AG an 1.1. AN gibt den Name des AGs ein 1.2. AN gibt Anschrift des AGs ein (evtl. abweichende Rechnungsadresse) 1.3. AN kann Telefon eingeben 1.4. AN kann Fax eingeben 1.5. AN kann E-Mail eingeben 1.6. AN kann Kommentar zu AG einfügen 1.7. AN kann den Zugang zum System für den AG freischalten 2. Der AN füllt das Formular aus und speichert die Daten 3. Das System prüft, ob alle Pflichtfelder ausgefühlt sind 4. Das System zeigt die „Liste der AG“ Seite mit einer Meldung, dass die Registrierung erfolgreich war
Fehlerfälle:
2. Wenn AN die Registrierung abbricht, wird das System die „Liste der AG“ Seite zeigen 3. Wenn nicht alle Pflichtfelder ausgefüllt waren, wird das Formular mit ausgefüllten Daten angezeigt, die leeren Pflichtfelder werden markiert
Häufigkeit:
Ein Mal pro AG

3.2.3 AG bearbeiten

Akteur:
AN
Ziel:
Einen AG bearbeiten
Auslöser:
Der AN ändert einige Daten eines AG
Vorbedingungen:
Der Anwendungsfall erweitert den Anwendungsfall „Liste der AG“ Mit dem APP (mobiles Gerät) und über Webseite möglich
Nachbedingungen:
Die Daten eines AG wurden erfolgreich geändert.
Erfolgsszenario:
1. Das System zeigt dem AN das Formular von dem AG an. 2. Der AN kann folgende Daten ändern: 2.1. Name des AGs 2.2. Anschrift des AGs 2.3. Telefon 2.4. Fax 2.5. E-Mail 2.6. Kommentar zu AG hinzufügen 2.7. Zugang zum System freischalten oder ausschalten 2.8. Passwort zurücksetzen 2.9. Status (Wenn der AN neuen Auftrag erstellt, werden in der AG Liste nur die aktiven AG angezeigt) 3. AN speichert die Änderungen 4. Das System prüft, ob alle Pflichtfelder ausgefüllt sind 5. Das System zeigt „Liste der AG“ an mit einer Meldung, dass die Änderungen erfolgreich gespeichert wurden
Fehlerfälle:
3. Wenn nicht alle Pflichtfelder ausgefüllt waren, wird das Formular mit ausgefüllten Daten angezeigt, die leeren Pflichtfelder werden markiert

3.2.4 Rechnungsübersicht

Akteur:
AG und AN
Ziel:
Übersicht von erhaltenen Rechnungen
Auslöser:
Benutzer möchte eine Übersicht von seinen Rechnungen bekommen oder diese herunterladen
Vorbedingungen:
Der Benutzer ist im System registriert Mit dem APP (mobiles Gerät) und über Webseite möglich
Nachbedingungen:
Benutzer hat eine Übersicht bekommen, bzw. Rechnungen heruntergeladen
Erfolgsszenario:
1. Benutzer geht auf Rechnungsübersicht 2. Das System zeigt dem Benutzer Übersicht von seinen Rechnungen 3. Der Benutzer kann eine Filterfunktion benutzen 4. Benutzer kann Rechnungen im PDF-Format herunterladen indem er auf Downloadicon klickt

3.2.5 Auftragsliste

Akteur:
AG und AN
Ziel:
Übersicht von eigenen Aufträgen
Auslöser:
Benutzer möchte seine Auftragsliste anschauen
Vorbedingungen:
Benutzer ist im System registriert Mit dem APP (mobiles Gerät) und über Webseite möglich
Nachbedingungen:
Benutzer hat eine Übersicht von Aufträgen bekommen
Erfolgsszenario:
1. Benutzer geht auf Auftragsübersicht 2. Das System zeigt dem Benutzer eine Liste von seinen Aufträgen an. Jeder Benutzer darf nur seine Aufträge sehen 3. Der Benutzer kann eine Filterfunktion benutzen 4. Benutzer kann auf Details gehen, um eine Übersicht von einem bestimmten Auftrag zu bekommen (Anwendungsfall „Auftragsdetails“) 5. AN kann einen neuen Auftrag hinzufügen (Anwendungsfall „Auftrag erstellen“)

3.2.6 Auftragsdetails

Akteur:
AG und AN
Ziel:
Details von einem Auftrag anschauen
Auslöser:
Der Benutzer möchte Details von einem Auftrag anschauen
Vorbedingungen:
Es muss mindestens ein Auftrag im System gespeichert sein Der Anwendungsfall erweitert den Anwendungsfall „Auftragsliste“ Mit dem APP (mobiles Gerät) und über Webseite möglich
Nachbedingungen:
Benutzer hat eine detaillierte Auftragsübersicht bekommen
Erfolgsszenario:
1. Das System zeigt eine detaillierte Übersicht von diesem Auftrag (Auftragsdatum, AG, Auftragsbeschreibung, AG_Auftragsnummer, Status, eine Liste mit Leistungen, die zu dem Auftrag gehören, AN sieht auch seine Kommentare zu dem Auftrag) 2. AN kann Auftrag abschließen (Anwendungsfall Auftrag abschließen) 3. Der Benutzer kann auch alle Funktionen nutzen, die der Anwendungsfall Leistungsübersicht anbietet

3.2.7 Auftrag stornieren

Akteur:
AN
Ziel:
Einen Auftrag stornieren
Auslöser:
AN möchte einen Auftrag stornieren
Vorbedingungen:
Der Auftrag ist schon im System gespeichert. Dieser Anwendungsfall erweitert den Anwendungsfall „Auftragsübersicht“. Der Auftrag enthält keine Leistungen mit Status „Offen“ oder „Erbracht“
Nachbedingungen:
AN hat einen Auftrag storniert.
Erfolgsszenario:
1. Der AN klickt auf den Button „Auftrag stornieren“ 2. Das System fragt, ob der AN tatsächlich den Auftrag stornieren möchte 3. AN bestätigt die Stornierung 4. Das System ändert den Status des Auftrages auf storniert.
Fehlerfälle:
4.1. Wenn der Auftrag mindestens eine Leistung mit Status bezahlt hat, wird der Vorgang mit einer Fehlermeldung abgebrochen.

3.2.8 Auftrag abschließen

Akteur:
AN
Ziel:
Einen Auftrag abschließen
Auslöser:
AN möchte einen Auftrag abschließen
Vorbedingungen:
Der Auftrag ist schon im System gespeichert Keine Leistung hat den Status Offen Dieser Anwendungsfall erweitert den Anwendungsfall „Auftragsdetails“ Mit dem APP (mobiles Gerät) und über Webseite möglich
Nachbedingungen:
AN hat einen Auftrag abgeschlossen
Erfolgsszenario:
1. Das System fragt, ob der AN tatsächlich den Auftrag abschließen möchte 2. AN bestätigt die Meldung 3. Das System schließt den Auftrag ab, zeigt die Seite Auftragsliste an und benachrichtigt den AN über die erfolgreiche Abschließung
Fehlerfälle:
1. Wenn der Auftrag noch offene Leistungen hat, wird der Vorgang mit einer Fehlermeldung abgebrochen
Häufigkeit:
Ein Mal pro Auftrag

3.2.9 Auftrag erstellen

Akteur:
AN
Ziel:
Einen Auftrag ins System einlegen
Auslöser:
AN speichert in das System einen neuen Auftrag Über Webseite und über mobiles Gerät möglich
Vorbedingungen:
Dieser Anwendungsfall erweitert den Anwendungsfall „Auftragsliste“ Mit dem APP (mobiles Gerät) und über Webseite möglich
Nachbedingungen:
Der Auftrag wird im System gespeichert
Erfolgsszenario:
1. Das System zeigt dem AN das Formular für neuen Auftrag an 1.1. Der AN wählt einen AG aus der AG-Liste aus 1.2. Füllt das Feld Auftragsbeschreibung aus 1.3. Der AN kann einen Kommentar zum Auftrag hinzufügen 1.4. Der AN kann die *AG_Auftragsnummer* eingeben 2. Das System prüft, ob alle Pflichtfelder ausgefühlt sind 3. Falls als AG ein „neuer AG“ ausgewählt wurde, wird das System den Anwendungsfall „Neuen AG registrieren“ aufrufen 4. Das System Speichert den Auftrag, zeigt die Auftragsdetails Seite und eine Meldung, dass der Auftrag erfolgreich hinzugefügt wurde
Fehlerfälle:
2. Wenn nicht alle Pflichtfelder ausgefüllt waren, wird das Formular mit ausgefüllten Daten angezeigt, die leeren Pflichtfelder werden markiert
Häufigkeit:
Pro AG zwischen einmal jährlich bis 5 mal wöchentlich.

3.2.10 Leistungsübersicht

Akteur:
AG und AN
Ziel:
Übersicht von gespeicherten Leistungen
Auslöser:
Benutzer möchte eine Leistungsübersicht bekommen
Vorbedingungen:
Es werden nur Leistungen angezeigt, die zu dem Auftrag gehören Mit dem APP (mobiles Gerät) und über Webseite möglich
Nachbedingungen:
Benutzer hat eine Übersicht von Leistungen bekommen
Erfolgsszenario:
1. Benutzer geht auf Leistungsübersicht 2. Das System zeigt dem Benutzer die Leistungsübersicht 3. Der Benutzer kann eine Filterfunktion benutzen 4. AN kann mehrere Leistungen mit Status „erbracht“ markieren und für diese eine Rechnung erstellen (Anwendungsfall „Rechnung erstellen“) 5. AN kann eine neue Leistung hinzufügen (Anwendungsfall „Leistung hinzufügen“) 6. Benutzer kann Details zu einer Leistung anzeigen lassen (Anwendungsfall „Leistungsdetails")

3.2.11 Leistung hinzufügen

Akteur:
AN
Ziel:
Eine neue Leistung ins System eintragen
Auslöser:
AN möchte eine neue Leistung eintragen
Vorbedingungen:
Es gibt im System schon ein Auftrag, zu dem die Leistung angelegt werden soll Dieser Anwendungsfall erweitert den Anwendungsfall „Leistungsübersicht“ Mit dem APP (mobiles Gerät) und über Webseite möglich
Nachbedingungen:
Die Leistung wird im System gespeichert
Erfolgsszenario:
1. Das System zeigt dem AN das Formular für eine neue Leistung an 2. AN wählt einen Auftrag aus 3. Der AN kann die Felder „Leistung“ und „Kommentar“ ausfüllen 4. Der AN gibt die Terminzeit ein 5. Das System prüft, ob der neue Termin sich mit keinem gespeicherten Termin überschneidet 6. Die Leistung wird im System mit Status „offen“ gespeichert 7. Das System zeigt dem AN die Seite Leistungsübersicht, mit der Meldung, dass die Leistung erfolgreich gespeichert wurde
Fehlerfälle:
4. Wenn der Termin sich mit einem anderen Termin überschneidet, wird das System eine entsprechende Meldung zeigen. Der AN kann: 4.2.1. den Termin trotzdem bestätigen 4.2.2. mit Punkt 4 Erfolgsszenario fortfahren 4.2.3. Der Vorgang „Leistung hinzufügen“ abrechen
Häufigkeit:
Pro Account zwischen einmal jährlich bis 5 Mal wöchentlich

3.2.12 Leistungsdetails

Akteur:
AG und AN
Ziel:
Übersicht einer Leistung
Auslöser:
Benutzer möchte eine Übersicht einer Leistung bekommen
Vorbedingungen:
Dieser Anwendungsfall erweitert den Anwendungsfall „Leistungsübersicht“ Mit dem APP (mobiles Gerät) und über Webseite möglich
Nachbedingungen:
Benutzer hat eine Übersicht von der Leistung bekommen
Erfolgsszenario:
1. Das System zeigt dem Benutzer die detaillierte Übersicht einer Leistung (Datum, Arbeitszeit, Dienstleistung, Betrag, Status, AN sieht auch seine Kommentare zu dieser Leistung) 2. Wenn die Leistung den Status offen hat, kann AN mit dem APP die Arbeitszeiterfassung starten (Anwendungsfall Arbeitszeit erfassen) 3. AN kann auf bearbeiten gehen (Anwendungsfall Leistung bearbeiten) 4. AN kann die Leistung stornieren

3.2.13 Leistung bearbeiten

Akteur:
AN
Ziel:
Leistung bearbeiten
Auslöser:
AN möchte eine Leistung bearbeiten
Vorbedingungen:
Dieser Anwendungsfall erweitert den Anwendungsfall „ Leistungsdetails“ Mit dem APP (mobiles Gerät) und über Webseite möglich
Nachbedingungen:
AN hat die Änderungen vorgenommen
Erfolgsszenario:
1. Wenn die Leistung den Status „offen“ hat, kann AN die Terminzeit ändern 1.1. Das System prüft, ob der neue Termin sich mit keinem gespeicherten Termin überschneidet 2. Wenn die Leistung den Status „offen“ hat, kann AN die Arbeitszeit eingeben 2.1. Das System ändert den Status auf Erbracht 3. Wenn die Leistung den Status „erbracht“ hat, kann AN die Arbeitszeit korrigieren
Fehlerfälle:
1. Wenn der Termin sich mit einem anderen Termin überschneidet, wird das System eine entsprechende Meldung zeigen. Der AN kann: 1.1. den Termin trotzdem bestätigen 1.2. mit Punkt 1 Erfolgsszenario fortfahren 1.3. den Vorgang „Leistung bearbeiten“ abbrechen 3. Wenn die Arbeitsendzeit in der Zukunft liegt, wird die entsprechende Fehlermeldung angezeigt. AN wird die Zeit korrigieren müssen

3.2.14 Rechnung erstellen

Akteur:
AN
Ziel:
Rechnung über erbrachte Dienstleistungen erstellen und evtl. per E-Mail an AG verschicken
Auslöser:
AN erstellt Rechnungen
Vorbedingungen:
Es gibt erbrachte Leistungen, für die noch keine Rechnung erstellt wurde Dieser Anwendungsfall erweitert den Anwendungsfall „Leistungsübersicht“ Mit dem APP (mobiles Gerät) und über Webseite möglich
Nachbedingungen:
Das System hat eine Rechnung für ausgewählte Leistungen erzeugt und evtl. per E-Mail an AG gesendet
Erfolgsszenario:
1. Das System erstellt auf dem Server eine Rechnung für die markierten Leistungen 1.1. Das System nimmt die Rechnungsadresse des AG 1.2. Das System generiert eine neue Rechnungsnummer 1.3. Das System nimmt das aktuelle Datum für Rechnungsdatum 1.4. Das System erstellt eine Liste mit Leistungen (Datum, Dienstleistung, Stunden, Betrag) 1.5. Das System berechnet den Rechnungsbetrag und zeigt die Rechnung dem AN als <rechungsnummer_AG>.pdf 2. AN bestätigt, dass alles stimmt 3. Die Rechnung wird im System gespeichert und evtl. per E-Mail an AG und AN gesendet Bei allen markierten Leistungen wird der Status von „erledigt“ auf „in Rechnung“ geändert
Fehlerfälle:
2. Wenn AN in der Rechnung Fehler findet oder Änderungen vornehmen will, kann er auf abrechen klicken. Das System wird die Datei <rechungsnummer_AG>.pdf löschen und die Leistungsübersicht Seite anzeigen

3.2.15 Arbeitszeit erfassen

Akteur:
AN
Ziel:
Die Arbeitszeit einer Leistung erfassen
Auslöser:
Dieser Anwendungsfall erweitert den Anwendungsfall „Leistungsdetails“
Vorbedingungen:
Nur mit dem APP möglich (mobiles Gerät)
Nachbedingungen:
Die Arbeitszeiten wurden erfasst und im System gespeichert
Erfolgsszenario:
1. Das APP speichert die Anfangszeit 2. Wenn der AN Pause macht, klickt er auf Pause 3. Das App speichert die Anfangszeit der Pause 4. Wenn die Pause zu Ende ist, klickt der AN auf „Pause beenden“ 5. Das App speichert die Endzeit der Pause 6. Wenn die Arbeit beendet wurde, klickt der AN auf „Termin beenden“ 7. Das App speichert die Arbeitsendzeit
Häufigkeit:
Ein mal pro Leistung

3.2.16 Leistung stornieren

Akteur:
AN
Ziel:
Eine Leistung stornieren
Auslöser:
AN möchte eine Leistung stornieren
Vorbedingungen:
Dieser Anwendungsfall erweitert den Anwendungsfall „Leistungsdetails“ Mit dem APP (mobiles Gerät) und über Webseite möglich
Nachbedingungen:
Der Status der Leistung wurde auf storniert geändert
Erfolgsszenario:
1. Der AN klickt auf den Button „stornieren“ 2. Das System fragt den AN, ob er die Leistung wirklich stornieren möchte 3. Der AN bestätigt das Stornieren 4. Das System ändert den Status auf „storniert“ 5. Das System zeigt dem Benutzer die „Leistungsübersicht“ Seite

3.3 Zusammenfassung

In diesem Kapitel wurden umfassend die funktionalen Anforderungen mit Hilfe von Use-Cases in Form von Tabellen dargestellt. Im Designkapitel wird ein prinzipieller Lösungsentwurf für alle Use-Cases angeboten, die sich auf die wesentlichen Business-Objekte (z.B. Auftrag, Auftraggeber, Leistung) beziehen. Dabei wurden die technischen Anforderungen, die sich aus Kapitel 2 ergeben, für eine multikanalfähige Realisierung umgesetzt. Für die Aspekte *„Rechnung erstellen“* und *„Termin Kollision Prüfung“* wurde keine Lösung angeboten, da sie zeitlich dem Umfang der Arbeit nicht entsprechen.

4 Design

Das Designkapitel zeigt einen Entwurf für die Android-Mobilplattform sowie die Java-Server-Faces-Webplattform, der es ermöglicht, durch die Implementierung der im Design vorgestellten Schnittstellen, ein System für beide Plattformen zur Verfügung zu stellen. Hierbei sind die jeweiligen Präsentationsschichten sowie die Datenbankanbindungen für die konkrete Umgebung zu implementieren. Die Logikschicht wird so konzipiert, dass sie unabhängig von der jeweiligen Plattform wiederverwendbar ist.

4.1 System Architektur

Das in der Abbildung 4.1 vorgestellte Diagramm veranschaulicht die Verteilung der Softwarekomponenten auf Prozesse und Server. Im Folgenden wird gezeigt, dass die wesentlichen Softwarekomponenten eine gemeinsame Codebasis haben, die sich aus dem Entwurf für eine multikanalfähige Anwendung ergeben.
Die Softwarekomponenten der Android-App laufen auf einem einzigen System, es hat seine eigene Darstellungsschicht, eigene Logik sowie eine eigene Datenbank.
Die Komponenten der webbasierten Anwendung laufen dagegen auf verschiedenen Systemen verteilt. Ein Browserclient dient dem Benutzer hierbei als Schnittstelle zur Eingabe von Daten und ermöglicht die Wiedergabe gespeicherter Daten. Die Anfragen bzw. die eingegebenen Daten werden über HTTP an den Server übertragen. Das verschafft dem Benutzer den entfernten Zugriff auf seine Daten. Der Server überprüft die Daten und sichert diese über TCP/IP in die Datenbank. Da dieses Modell mit einem Server arbeitet ermöglicht es eine parallele Nutzung dieser Anwendung von mehreren Browsern.
Die Logikschicht (av.logik) sowie der DB-Adapter (av.dbadapter) sind die wiederverwendbaren Komponenten für die beiden Plattformen. Des Weiteren werden die Schnittstellen des DB-Adapter im Kapitel 4.4.2 vorgestellt.

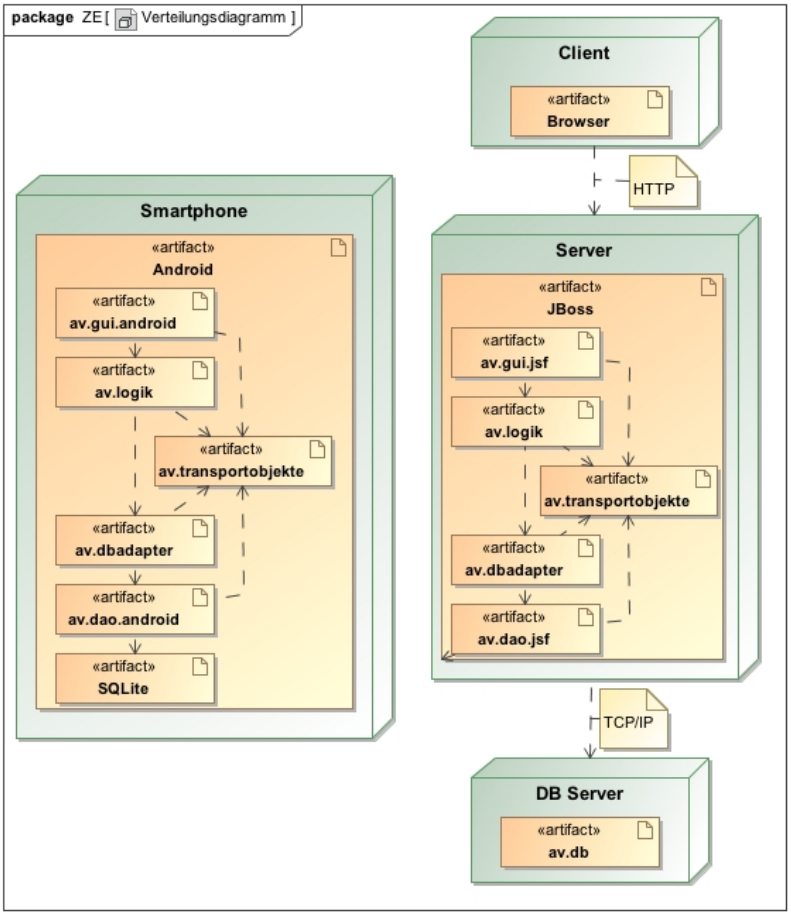

Abbildung 4.1: Verteilungsdiagramm

4.2 Designentscheidungen

Für den Entwurf einer Software gibt es eine Reihe von in der Praxis verwendeten Mustern, welche die Erweiterbarkeit und Wartbarkeit von Software sicherstellen. In dieser Arbeit kommen von diesen Mustern mehrere zum Einsatz, die im Folgenden detailliert erläutert werden.

4.2.1 MVC-Konzept

Da die Anwendung in zwei Programmiermodellen (Android und JSF) realisiert werden soll, wird die User-Interface-Komponente für jedes Modell einzeln entwickelt, die Logik-Komponente soll von der UI-Komponente entkoppelt werden und diese dann für beide Modelle wiederwendbar sein. Um die Wiederverwendbarkeit von einzelnen Komponenten gewährleisten zu können, wurde für den Entwurf das MVC-Konzept zugrunde gelegt.

Die Model-View-Controller-Architektur besteht aus:
„Dem Modell: Das Modell repräsentiert den internen Zustand einer Komponente. In ihm sind alle notwendigen Informationen sowie die anzuzeigenden Daten enthalten. Dazu zählen Informationen wie z.B. die minimale und maximale Größe der darzustellenden Komponente.
Dem View: Der View ist für die eigentliche Präsentation zuständig. Er erhält die darzustellenden Daten vom Model und zeichnet sie auf den Bildschirm.
Dem Controller: Der Controller ist für die Interaktion mit dem Anwender verantwortlich. Er erhält eine Eingabe, z.B. einen Mausklick oder die Auswahl eines Menüeintrags, bearbeitet sie und verständigt das Modell. Dieses nimmt anschließend die entsprechenden Änderungen an den Daten vor, z.B. markiert es ein Listenelement als selektiert.“
[JP1999] S.436

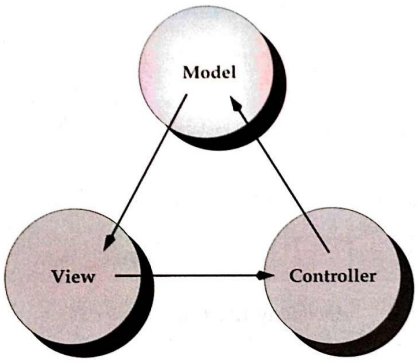

Abbildung 4.2.1 Die MVC-Architektur [JP1999] S.436

4.2.2 Schnittstellen zur Persistenzschicht

Die Anwendung hat zwei unterschiedliche Schnittstellen zur Persistenzschicht: Eine Schnittstelle zu SQLite- und die andere zu einer relationalen Datenbank. Und zwar funktional (SQLite kann nicht alles, was beispielsweise Oracle anbietet), aber auch aus der Verteilungssicht (einmal lokal und einmal remote). Deswegen wird ein DB-Adapter eingesetzt, um diese Unterschiede zu kapseln und die Persistenzschnittstelle für den Anwendungsentwickler transparent zu gestalten.
Die Komponente DB-Adapter (Abb. 4.3.2) besteht aus der Klasse *DbDriverholder*, die in dem Unterkapitel 4.3.2 genauer beschrieben wird, und DAO-Interfaces. DAO-Interfaces sind Interfaces, die Methoden enthalten, um Daten aus der Datenbank zu lesen, bzw. in die Datenbank zu schreiben. Somit wird die Unabhängigkeit der nutzenden Client Software von der jeweiligen Implementierung zur Persistenzschicht sichergestellt.

4.2.3 Factory-Pattern

Da nur die Logik-Komponente auf die Persistenzschicht zugreift, muss beim Start der Applikation eine Instanz erstellt werden, welche die Abbildung der DAO-Interfaces auf die Persistenzschicht übernimmt. Diese wird mit Hilfe des Factory-Pattern für die jeweilige Zielumgebung transparent für die nutzenden Logikkomponenten erzeugt.
„ Eine *Factory* ist ein Hilfsmittel zum Erzeugen von Objekten. Sie wird verwendet, wenn das Instanziieren eines Objekts mit dem *new*-Operator alleine nicht möglich oder sinnvoll ist – etwa weil das Objekt schwierig zu konstruieren ist oder aufwändig konfiguriert werden muss, bevor es verwendet werden kann. Manchmal müssen Objekte auch aus einer Datei, über eine Netzwerkverbindung oder aus einer Datenbank geladen werden, oder sie werden auf der Basis von Konfigurationsinformationen aus systemnahen Modulen generiert. Eine Factory wird auch dann eingesetzt, wenn die Menge der Klassen, aus denen Objekte erzeugbar sind, dynamisch ist und zur Laufzeit des Programms erweitert werden kann.“ [HDJP 2006] S.233

4.2.4 Transportobjekte

Klassen, die nur eine Transportfunktion (Constructor, Getter- und Setter-Methoden) haben, sind in der Komponente *Modell* implementiert. Diese Komponente ist eine Library für alle anderen Komponenten. Das ermöglicht, dass die UI-Komponente auf die Attribute der Transportobjekte zugreifen kann und sie auf einem Bildschirm darstellt. Die Logik-Komponente kann Werte dieser Attribute prüfen, bearbeiten und über DAO-Interfaces an die Persistenzschicht übergeben. Die jeweilige DB-Implementierung kann die Daten mit Hilfe von Getter-Methoden lesen und in die Datenbank schreiben bzw. die Daten aus der Datenbank lesen und mit Settern neue Transportobjekte erzeugen und diese an die *Logikkomponente* weiterleiten.

4.2.5 Parametrisierte Methoden

Da das System für mehrere Plattformen gedacht ist, soll es auf Geräten mit unterschiedlichen Bildschirmgrößen laufen. Um den Bildschirm besser nutzen zu können, werden Anfragen dynamisch generiert. Zum Beispiel, wenn man auf die Auftragsgeberliste geht, werden auf einem kleinen Smartphone-Bildschirm nur die Firmennamen dargestellt, im Gegensatz zur webbasierten Version, bei der auf einem größerem Monitor viel mehr Informationen dargestellt werden können.
Das wird wie folgt realisiert (am Beispiel von *Auftragsgeberliste*):
Wenn der Benutzer auf *Auftragsgeberliste* geht, wird die Methode *getAGListe(List<Datentypen> agKomponente)* aufgerufen. In der Liste *agKomponente* werden nur die Attribute vom Auftragsgeber übergegeben, die dargestellt werden sollen.

4.2.6 Stateless Services

Die Logik-Komponente ist mit Hilfe von zustandslosen Diensten realisiert, die die Datenobjekte für die GUI liefern bzw. Dienste zur Verfügung stellen, um Änderungen über die Persistenz-Fassade vorzunehmen.

4.3 Softwarearchitektur

Aufbauend auf die im Kapitel 4.2 vorgestellten Entwurfsmuster zeigt Abbildung 4.3 den Aufbau der Softwarearchitektur.

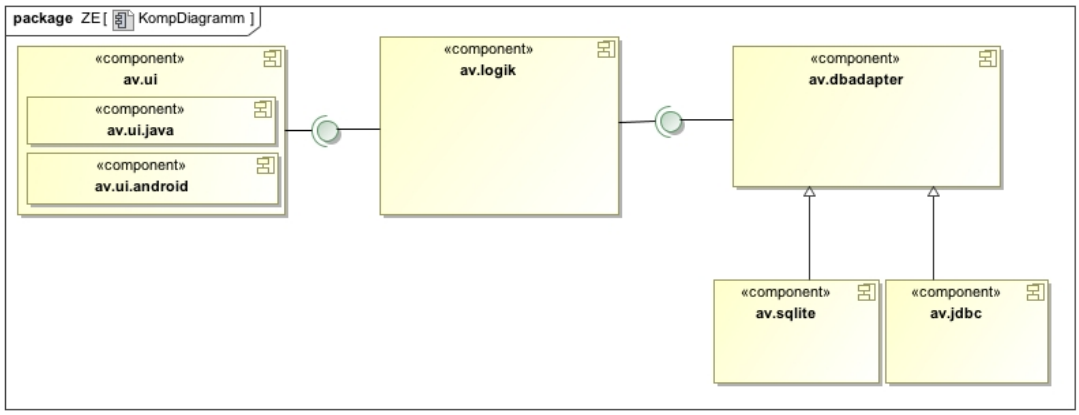

Abbildung 4.3: three-tier Architektur.

Die Trennung zwischen Darstellung und Verarbeitungslogik erfolgt auf Basis des MVC-Musters. Die Schnittstelle zwischen der Logik und der Persistenz-Schicht ist durch eine Fassade, den so genannten „DB-Adapter“, gekapselt. Die Schnittstellen zwischen den jeweiligen GUI-Komponenten auf unterschiedlichen Plattformen als auch die Schnittstellen für die Persistenzschicht sind ausschließlich über die Logik-Komponente gekoppelt.

4.3.1 Logik-Komponente im Detail

Die Logik-Komponente ist als Sammlung von Diensten implementiert, die die Zustände der Transportobjekte manipulieren.

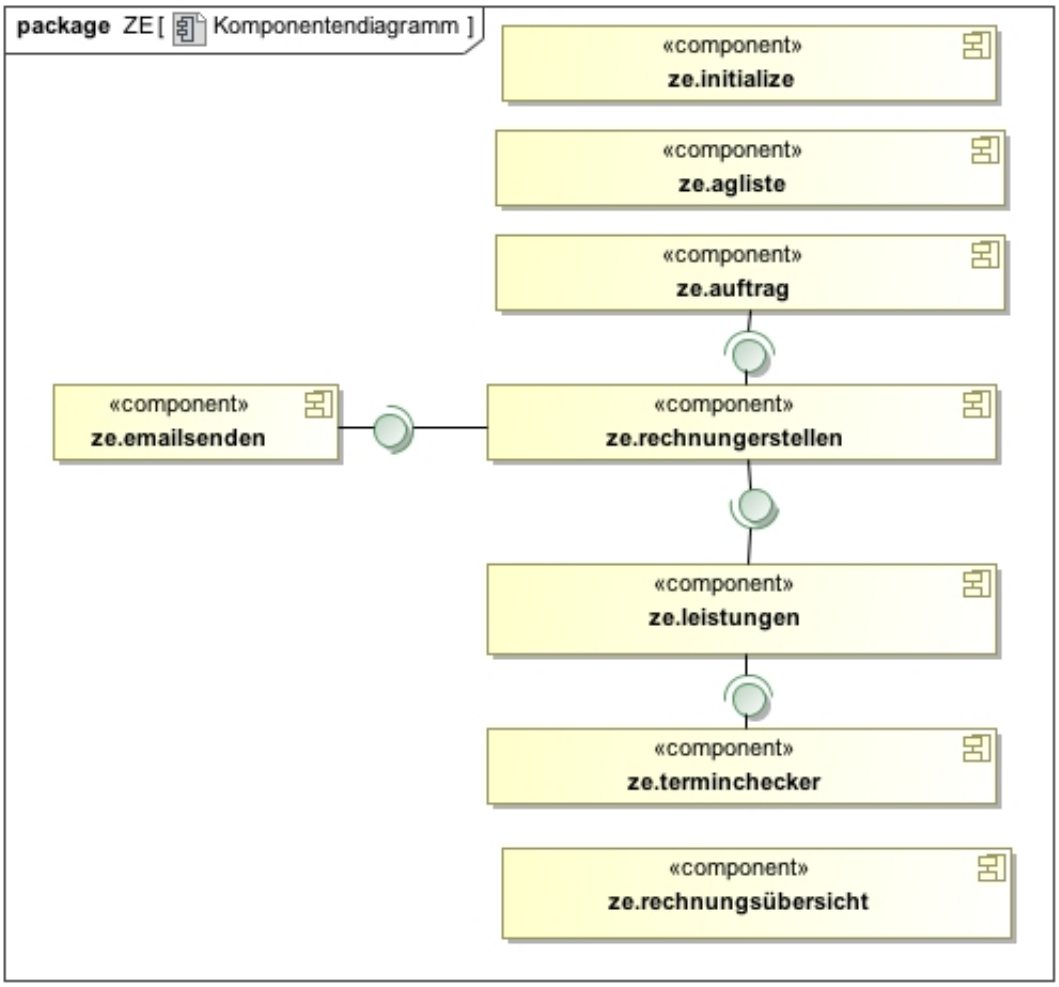

Abbildung 4.3.1.1: Logikschicht-Komponentendiagramm.

Die Funktionalität der Logik-Komponente wird aus zwei Sichten betrachtet. Die erste Sicht erläutert, wie die möglichen Zustandsübergänge der betroffenen Entitäten sind. Abschließend wird gezeigt, wie die Dienste der Logik-Komponente die Zustandsübergänge in verschiedenen Entitäten auslösen.

4.3.1.1. Zustandsdiagramme

Leistung:

1. **offen:** Jede neue Leistung wird mit dem Status „offen“ erstellt. In diesem Status kann der AN die Leistung stornieren oder die Arbeitszeit erfassen.
2. **erbracht:** Sobald eine Leistung eine Arbeitszeit zugewiesen bekommt, wird der Status auf „erbracht“ geändert.
3. **storniert:** der AN kann eine Leistung stornieren, wenn sie den Status „offen“ oder „erbracht“ hat.
4. **in Rechnung:** Wenn eine Leistung den Status „erbracht“ hat, kann AN für diese Leistung eine Rechnung erstellen. Nachdem eine Rechnung erstellt wurde, wird der Status der Leistung auf „in Rechnung“ geändert. Wenn die Rechnung als „bezahlt“ gespeichert wird, wird der Status der Leistung auf „bezahlt“ gesetzt.

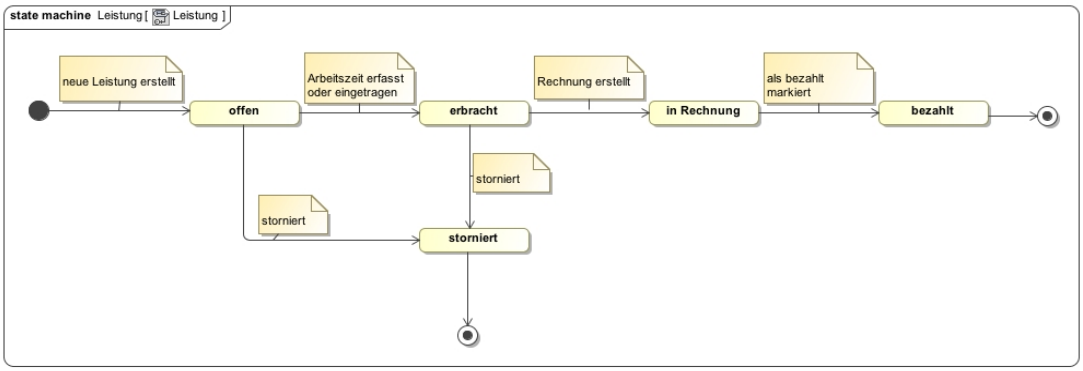

Abbildung 4.3.1.1.1: Zustandsdiagramm Leistung

Auftrag:

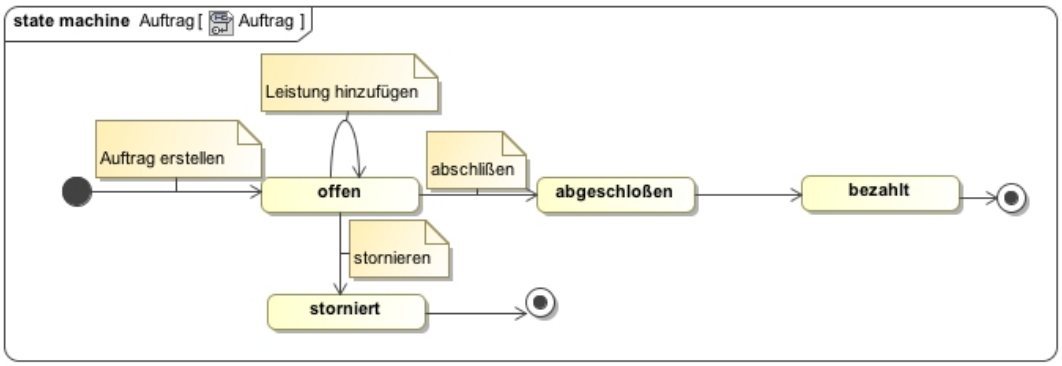

Abbildung 4.3.1.1.2: Zustandsdiagramm Auftrag

1. **offen:** Wenn der AN einen neuen Auftrag erstellt, bekommt der Auftrag den Status „offen“. Solange ein Auftrag den Status „offen“ hat, kann der AN neue Leistungen zum Auftrag hinzufügen.
2. **storniert:** Der AN kann einen offenen Auftrag stornieren, dabei wird der Status aller Leistungen, die zu diesem Auftrag gehören, auf „storniert“ geändert. Der Auftrag darf keine Leistung beinhalten, die im Status „in Rechnung“ oder „bezahlt“ ist.
3. **abgeschlossen:** Wenn ein Auftrag den Status „offen“ hat und es keine beinhaltete Leistung mit Status „offen“ gibt, kann der AN den Auftrag abschließen.
4. **bezahlt:** Sobald alle Leistungen von diesem Auftrag entweder den Status “bezahlt“ oder „storniert“ haben, wird der Status des Auftrages auf „bezahlt“ geändert.

Rechnung:

Abbildung 4.3.1.1.3: Zustandsdiagramm Rechnung

1. **offen:** Wenn eine Rechnung erstellt wird, bekommt sie automatisch den Status „offen“.
2. **bezahlt:** Wenn AN den Status auf „bezahlt“ ändert, wird der Status von den zugehörigen Leistungen und Aufträgen auf „bezahlt“ gesetzt.

4.3.1.2. Sequenzdiagramme

In den Sequenzdiagrammen Abb. 4.3.1.2.1 – Abb. 4.3.1.2.3 wird gezeigt, wie die Dienste der Logik-Komponente verschiedene Zustandsübergänge auslösen.

Auftrag erstellen (Abbildung 4.3.1.2.1)**:**

1. Das Szenario des dargestellten Sequenzdiagramms beginnt mit der Interaktion „Auftragsliste Anzeigen“ des Benutzers.
2. Die Anfrage wird von der Klasse *Auftragsservice* der Logik-Komponente über den DB-Adapter an die Datenbank weitergeleitet. Die Persistenzschicht erstellt eine Liste von Aufträgen als Transportobjekte.
3. Der *Auftragsservice* bekommt diese Liste als Rückgabe.
4. Die Liste wird an die UI-Komponente weitergeleitet.
   Beim Klicken auf „Auftrag Erstellen“ stellt das UI dem Benutzer das Formular für einen neuen Auftrag dar, welches der Benutzer ausfüllen soll.

5. Sobald das Formular ausgefüllt ist und der Benutzer auf „Speichern“ klickt, wird ein neues Transportobjekt *Auftrag* erstellt. Das System weist dem Auftrag den Status „offen“ zu.
6. Das UI bekommt das Objekt als Rückgabewert.
7. Als nächstes wird die Methode *createAuftrag(auftrag)* der Klasse *Auftragsservice* aufgerufen. Als Parameter wird das Transportobjekt *auftrag* übergegeben.
8. Die Klasse Auftragsservice ruft eigene private Methode *pruefeAuftrag(auftrag)* auf, in der alle Attribute auf Gültigkeit geprüft werden.
9. Danach wird über DB-Adapter die Methode *speichereAuftrag(auftrag)* aufgerufen.
10. Als Rückgabewert kommt die *id* des Auftrages.
11. Die UI-Komponente bekommt eine Bestätigung über eine erfolgreiche Erstellung.

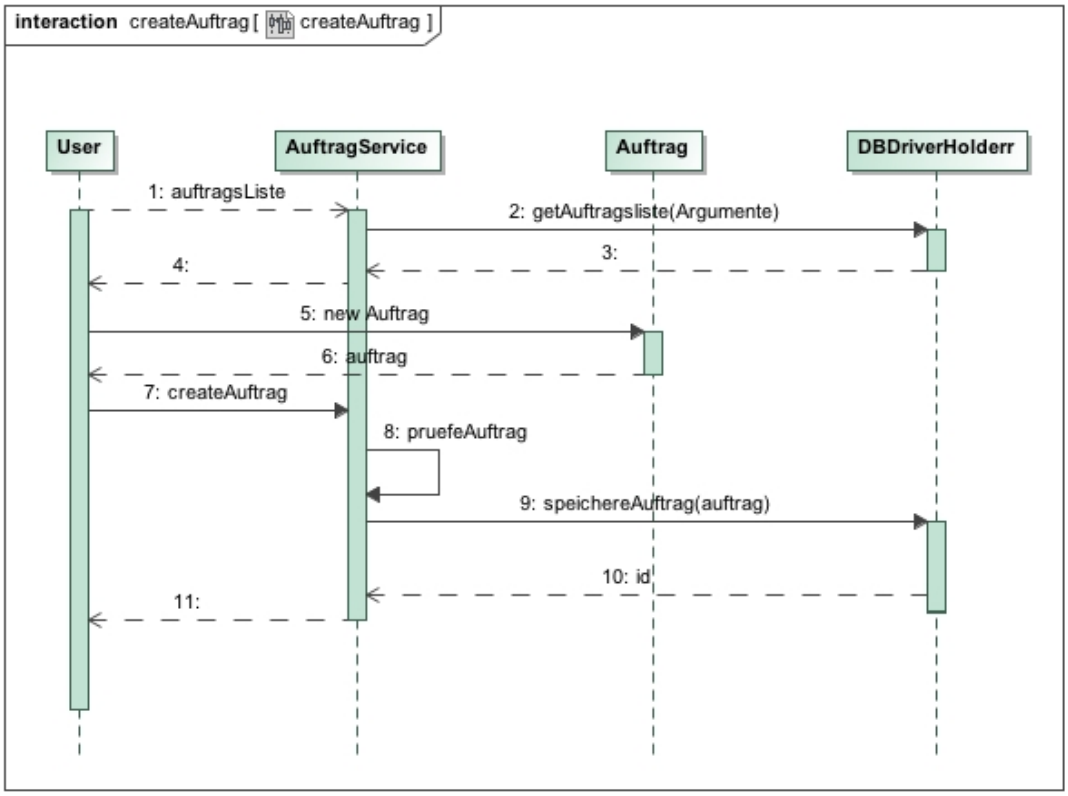

Abbildung 4.3.1.2.1: Sequenzdiagramm Auftrag erstellen

Leistung hinzufügen (Abbildung 4.3.1.2.2)**:**

1. Der Benutzer ruft die Auftragsliste auf.
2. Die UI-Komponente bekommt eine Liste mit Aufträgen zurück.
3. Beim Aufrufen von Auftragsdetails wird die Methode *auftragsDetails(id)* der Klasse *Auftragsservice* aufgerufen.

4. Die Anfrage wird an die Persistenzschicht weitergeleitet.
5. Zurück kommt der Auftrag mit allen Attributen als Transportobjekt.
6. Dieser wird an die UI-Komponente weitergeleitet.

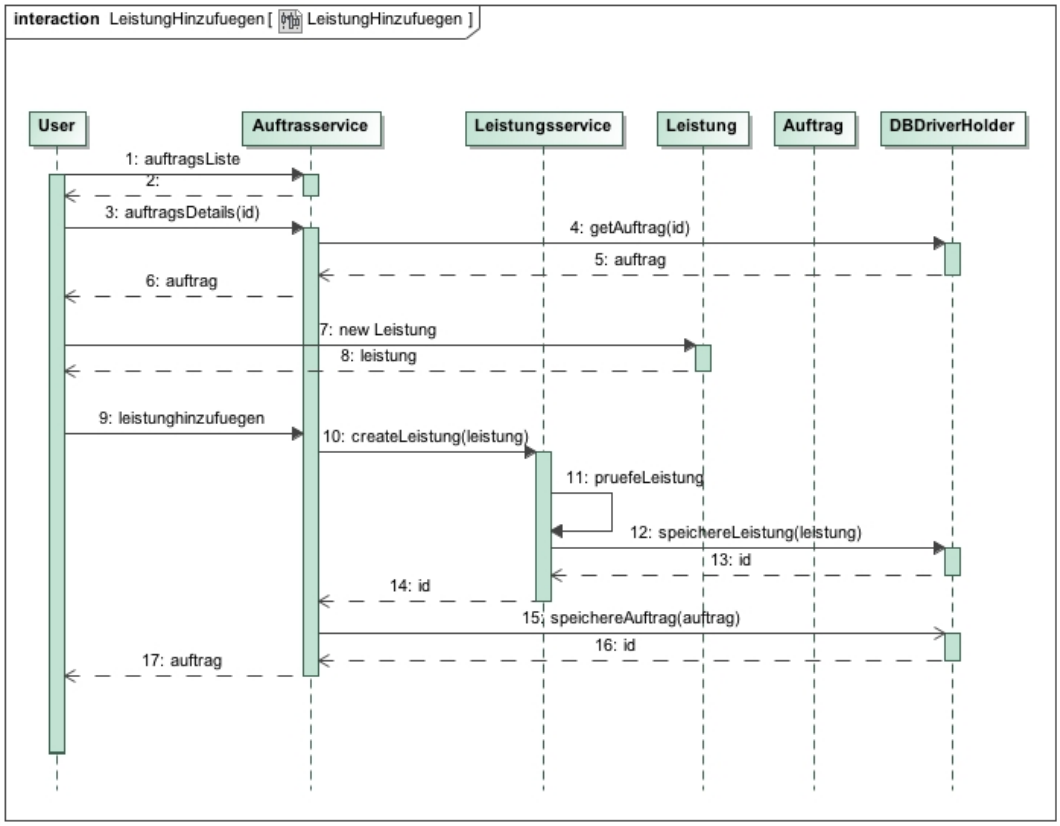

Abbildung 4.3.1.2.2: Sequenzdiagramm Leistung hinzufügen

Beim Erstellen einer neuen Leistung wird das UI dem Benutzer das Formular für eine neue Leistung darstellen, welches der Benutzer ausfüllen soll.

7. Sobald das Formular ausgefüllt ist und der Benutzer auf „Speichern“ klickt, wird ein neues Transportobjekt *Leistung* mit dem Status „offen“ erstellt.
8. Das UI bekommt das Objekt als Rückgabewert.
9. Dann wird die Methode *leistunghinzufuegen(leistung)* der Klasse Auftragsservice aufgerufen.
10. Diese Überprüft, ob der Auftrag den Status offen hat, und ruft dann die Methode *createLeistung(leistung)* der Klasse Leistungsservice der Logik-Komponente auf.
11. Es wird eine private Methode *pruefeLeistung* aufgerufen, um alle Leistungsattribute auf Gültigkeit zu prüfen.
12. Als nächstes wird die Leistung über DB-Adapter in DB gespeichert.
13. Als Rückgabewert kommt die *id* der Leistung.

14. Diese wird an den Auftragsservice weitergeleitet.
15. Danach wird die Leistung *id* zur Leistungsliste des Auftrages hinzugefügt und die Änderungen des Auftrags werden in DB gespeichert.
16. Als Rückgabewert kommt die Auftrags-*id*
17. Die UI-Komponente bekommt eine Bestätigung, dass die Leistung erfolgreich hinzugefügt wurde.

Rechnung erstellen (Abbildung 4.3.1.2.3)**:**

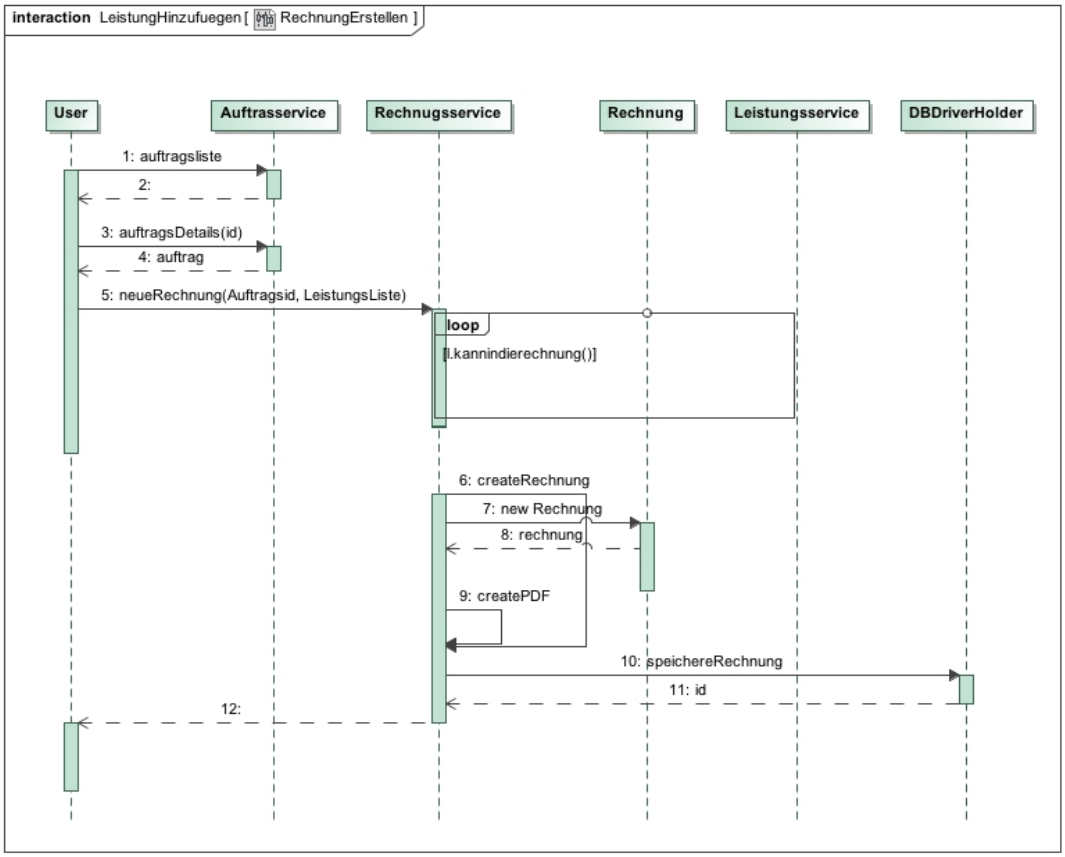

Abbildung 4.3.1.2.3: Sequenzdiagramm Rechnung erstellen

1. Der Benutzer ruft die Auftragsliste auf.
2. Die UI-Komponente bekommt die Liste mit Aufträgen zurück.
3. Beim Aufrufen von Auftragsdetails wird die Methode *auftragsDetails(id)* der Klasse *Auftragsservice* aufgerufen.
4. Zurück kommt der Auftrag mit allen Attributen als Transportobjekt.

5. Beim Erstellen der Rechnung wird die Methode *neueRechnung(auftragsid, leistungsListe)* der Klasse *Rechnungsservice* aufgerufen. Als Parameter wird die ID des Auftrages übergegeben. Wenn die Rechnung nicht für den ganzen Auftrag erstellt werden soll, sondern nur für einige Leistungen, wird auch eine Liste mit Leistungen übergegeben, für die die Rechnung erstellt werden soll.
   Als erstes wird überprüft, ob der Auftrag den Status abgeschlossen hat. Wenn die Rechnung für den ganzen Auftrag erstellt werden soll und der Auftrag den Status offen hat, wird der Auftrag abgeschlossen. Danach werden alle Leistungen überprüft, ob die alle den Status erbracht haben.
6. Dann wird die private Methode *createRechnung* aufgerufen.
7. Als erstes wird ein neues Transportobjekt *Rechnung* erstellt.
8. Das Objekt wird als Rückgabewert zurückgegeben.
9. Dann wird die private Methode *createPDF()* aufgerufen, in der eine PDF-Datei erstellt wird.
10. Als nächstes wird die Rechnung über DB-Adapter in DB gespeichert. Alle Leistungen, evtl. auch der Auftrag, bekommen den Status „In Rechnung“.
11. Als Rückgabewert kommt die ID der Rechnung.
12. Abschließend bekommt die UI-Komponente Bestätigung über erfolgreiche Erstellung der Rechnung.

4.3.2 DB-Adapter im Detail

Nach der ausführlichen Beschreibung der Logik-Komponente wird auch der DB-Adapter vorgestellt. Im Kapitel 4.2.2 gab es schon eine Einführung in die Schnittstelle. Hier wird gezeigt, wie es in der Praxis implementiert wurde. Abbildung 4.3.2 zeigt, dass der DB-Adapter aus der Klasse DBDriverHolder und DAO Interfaces besteht.

IDBDriver Interface

Methoden des *IDBDriver* verwenden an allen Stellen, an denen Interaktion mit der Persistenzschicht notwendig ist, DAO Interfaces, um für den Client die jeweilige Implementierung zu kapseln.

```
public interface IDBDriver {

	public void initialize(Object initializeProperties);
	public IAGDAO getAGDB();
	public IAuftragDAO getAuftragDB();
	public ILeistungDAO getLeistungDB();
	public IRechnungDAO getRechnungDB();
}
```

DAO Interfaces (am Beispiel der Auftraggeber)

Mit *getAGDB()* bekommt man die Klasse *AGDAO*, die das Interface *IAGDAO* implementiert.

```
public interface IAGDAO {

	public List<Auftraggeber> getAGListe(List<Datentypen> agKomponente);
	public int speichereAg(Auftraggeber ag);
	public Auftraggeber getAG(int id);
	public boolean nameFree(int id, String name);
	public void delAG(int id);
}
```

Abbildung 4.3.2: DB-Adapter

Klasse DBDriverHolder
Hat eine Klassenvariable *instance* von Typ DBDriverHolder und eine Instanzvariable *dbDriver* der Persistenzschicht. Wenn die Applikation gestartet wird, wird die Methode *initializeDBDriver* aufgerufen.

```
public static void initializeDBDriver(String clazz, Object
initializeProperties){
	IDBDriver dbDriver;
	dbDriver = (IDBDriver) Class.forName(clazz).newInstance();
	dbDriver.initialize(initializeProperties);
	getInstance().setDbDriver(dbDriver);

}
```

Dabei wird eine Instanz von der jeweiligen DB-Implementierung erstellt und der Variable *dbDriver* zugewiesen.

Da das Grundkonzept für alle Zugriffe auf die Persistenzschicht gleich ist, wird abschließend das Speichern von einem Auftraggeber angezeigt.

```
int neuID =
DBDriverHolder.getInstance().getDbDriver().getAGDB().speichereAg(ag);
```

Was beim Aufruf von einzelnen Methoden passiert, wurde bereits oben beschrieben. Der letzte Aufruf `speichereAg(ag)` gehört zu Persistenzschicht und wird deswegen im Kapitel 4.3.3 beschrieben.

4.3.3 Persistenzschicht. Realisierung für Android

Die Klasse *AGDAO* implementiert alle Methoden des Interfaces *IAGDAO*. Um das Beispiel „Speichere Auftraggeber“ abzuschließen, wird die Methode *speichereAg* der Klasse *AGDAO* vorgestellt.

```
public int speichereAg(Auftraggeber ag) {
	sqliteDB.openWr();
	sqliteDB.mDb.execSQL("UPDATE "+ TABLE_AG +" SET " +
		Datentypen.name +" = '"+ ag.getAgName() + "', " +
		Datentypen.tel + " ='"+ ag.getTel() + "', " +
		Datentypen.fax + " ='"+ ag.getFax() + "', " +
		Datentypen.email + " ='"+ ag.getEmail() + "', " +
		Datentypen.kommentar + " = '" + ag.getKommentar() + "', " + "
		WHERE " + Datentypen._id + " = " + ag.getId());
		//Adresse
	sqliteDB.mDb.execSQL("UPDATE " +TABLE_ADRESSE+ " SET " +
```

```
                Datentypen.strasse + " = '"+ ag.getAdresse().getStrasse() +
                "', " +
                Datentypen.hausnummer + " = '"+
                ag.getAdresse().getHausnummer() + "', " +
                Datentypen.plz + " = "+ ag.getAdresse().getPlz() + ", "+
                Datentypen.ort +" = '" + ag.getAdresse().getOrt()+
                "' WHERE "+ Datentypen._id +" = (SELECT " + Datentypen.adresse
                + " FROM " + TABLE_AG + " WHERE " + Datentypen._id + " = " +
                ag.getId()+ ")");

                sqliteDB.close();
                return ag.getId();
}
```

in der Methode *speichereAg(ag)* wird zuerst die DB zum Schreiben geöffnet, dann wird der AG und seine Adresse in die DB geschrieben. Zum Schluss wird die DB geschlossen und die ID zurückgegeben.

4.3.4 Klassendiagramm der mobilen Anwendung

Die Abbildung 4.3.4 fasst die wichtigsten Klassen als Diagramm zusammen.

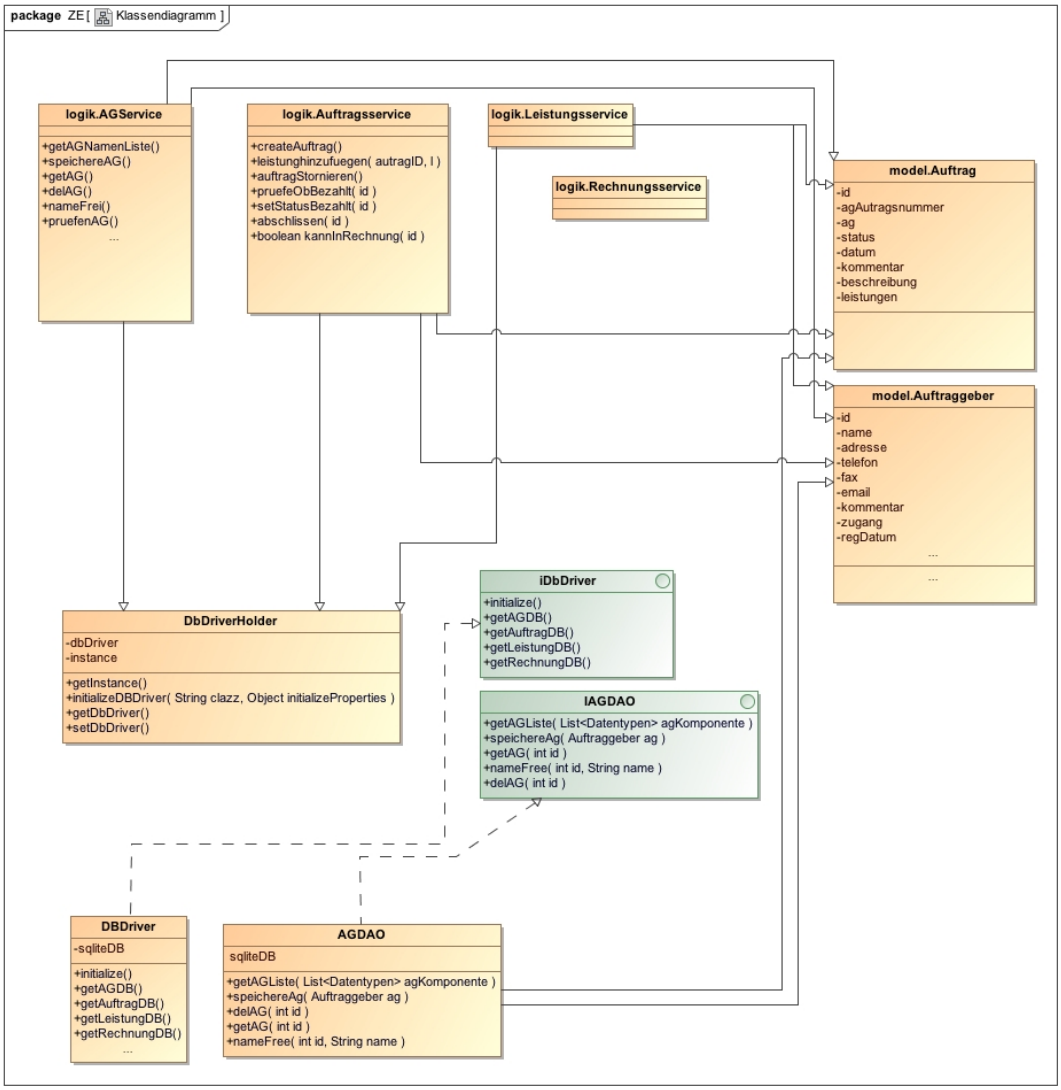

Abbildung 4.3.4: Klassendiagramm

4.4 Entwurfsentscheidungen für eine multikanalfähige Anwendung

Im Kapitel 2 wurden bereits die Grundlagen von beiden Plattformen erwähnt. Um die Multikanalfähigkeit zu erreichen muss man zuerst die Schnittstellen klar definieren. Das sieht man in Abbildung 4.3. Da es sich um zwei unterschiedliche Plattformen handelt, welche auch unterschiedliche Funktionen anbieten und unterschiedliche Muster verwenden, musste man an den jeweiligen Plattformen die fehlenden Funktionen ergänzen und an bestimmte Muster anpassen. Diese Lösungen wurden bereits in diesem Kapitel beschrieben und werden im Folgenden zusammengefasst.

4.4.1 Plausibilitäts- und Vollständigkeitsprüfung auf Benutzereingaben

Mit Hilfe von Value-Change-Listener kann man in JSF die eingegebenen Werte sehr frühe und GUI-nah prüfen. Bei fehlerhaften Eingaben unterbieten die Value-Change-Listener die Aktualisierung von Managed-Beans. Das Android-Framework kümmert sich nicht darum, wie die GUI-Komponente befüllt wird. Es gibt zwar die Möglichkeit Basisdatentypen zu prüfen bzw. einzuschränken, aber man kann die Abhängigkeiten zwischen Eingaben nur mit viel Programmieraufwand erreichen. Diese Funktionen wurden daher in der Logik-Komponente implementiert. Damit wird auch für Android sichergestellt, dass nur konsistente Daten in die Datenbank geschrieben werden. Da die beiden UI-Komponenten die Logik-Komponente verwenden, wird die Überprüfung bei JSF auch in der Logik-Komponente passieren. Ein Nachteil ist, dass man diese Prüfungen in der Logik-Komponente und bei Value-Change-Listener in JSF synchron halten muss.

4.4.2 Model-Objekte als Transportobjekte für Benutzereingaben

Da die Android-Laufzeitumgebung die Aktivitäten zerstören kann, entsteht die Notwendigkeit den Zustand von Aktivitäten und damit die Benutzereingaben zu sichern, um beim Restart der Applikation den Zustand wiederherstellen zu können. Daraus folgt, Transportobjekte zu verwenden, die die Objekte in GUI abbilden, aber noch keine persistente Daten sind. Dazu wurden im Entwurf die sogenannten Model-Objekte eingeführt, die im Android implementiert und in den entsprechenden Lifecycle Methoden der Aktivitäten zwischen gespeichert werden müssen, und in JSF auf Managed-Beans abbilden lassen.

4.4.3 DB-Adapter

Das Interface *IDBDriver* ermöglicht den Zugriff auf alle Methoden, mit deren Hilfe Daten zwischen der Logik-Komponente und der Datenbank übertragen werden. Für jede

Datenbank gibt es eine eigene Implementierung der Persistenzschicht, die diese Methoden implementiert. Somit müssen bei Änderungen in einer Datenbank keine Änderungen oder Anpassungen in der Logik-Komponente passieren, sondern nur die entsprechende Implementierung der Persistenzschicht angepasst werden.

4.5 Zusammenfassung

In diesem Kapitel wurde die Systemarchitektur sowie die wesentlichen Softwarekomponenten vorgestellt. Dann wurden Designentscheidungen beschrieben, die für die Verwendung von Hauptkomponenten auf JSF- und auf Android-Plattform getroffen wurden. Dabei wurden auch Probleme und deren Lösungen beschrieben, die beim Entwurf getroffen wurden. Als Nächstes wurde die Softwarearchitektur entworfen und mit Hilfe von Komponenten- Sequenz-, Zustands- und Klassendiagramm beschrieben. Dann wurden die einzelnen Softwarekomponenten mit Quellcode-Beispielen beschrieben. Zum Schluss wurden die Tragfähigkeit des Entwurfs für eine multikanalfähige Anwendung erläutert.

5 Zusammenfassung und Ausblick

5.1 Zusammenfassung

Für die Entwicklung des Systems wurden zu Beginn dieser Arbeit alle notwendigen Grundlagen erarbeitet. Es wurden Vor- und Nachteile von mobilen Geräten und Web-Anwendungen beschrieben. In Folge dessen wurde die Entscheidung getroffen das System für beide Plattformen zu entwickeln mit der Voraussetzung, dass die wichtigsten Komponenten so modelliert werden sollen, dass sie ohne Anpassungen für beide Plattformen verwendbar sein sollen.

Um die Anforderungen an das System zu beschreiben, wurde eine ausführliche Analyse durchgeführt. Dabei wurden die einzelnen Anwendungsfälle beschrieben und als USE-Case Diagramm modelliert. Dabei wurde insbesondere auf die Vollständigkeit aller beteiligten Anwendungsfälle Wert gelegt, damit sich keine Widersprüche und Ungenauigkeiten bei der Entwicklung des Designs ergeben können.

Nachdem die Arbeit mit Analyse angeschlossen war, wurde die Systemarchitektur vorgestellt. Danach wurde die Softwarearchitektur entwickelt und detailliert beschrieben. Dabei wurden unterschiedliche Muster und Konzepte angeschaut und mehrere Designentscheidungen getroffen, um das System auf einzelne Komponenten zu verteilen und die Schnittstellen zwischen den Komponenten so zu definieren, um die Hauptkomponenten für die beiden Plattformen wiederverwendbar zu entwickeln. Sicherheit und Datenschutz wurden aus zeitlichen Gründen nicht behandelt.

Als Ergebnis liegen die vollständige Spezifikation und der Entwurf vor. Die Implementierung der Auftraggeber – USE-Cases für Android zeigt exemplarisch die Tragfähigkeit des Entwurfs. Am Anfang wurde für Entwicklung das Android 2.3 verwendet. Während dieser Arbeit hat Google die 4. Version „ice cream sandwich“ veröffentlicht. Das ist die erste Version, die sowohl auf Smartphones sowie auch auf Tabs funktioniert. Deswegen wurde

diese Applikation auf Android 4 umgestellt. Die aktuellste Version ist Android 4.1 „Jelly Bean“ [Stand 08.2012].

5.2 Ausblick

In dieser Arbeit wurde eine funktionale Spezifikation in der Analyse erstellt. Ein Teil der Android-App wurde implementiert. Im Entwurf sollte noch das Verfahren zu Terminkollisionen konzipiert werden. Als nächstes sollte man die Implementierung der restlichen USE-Cases für Android und JSF durchführen. Ebenfalls steht das Konzept für das Synchronisieren von Datenbanken zwischen mobilen Geräten und JSF noch aus.

6 Literatur

[JSF 2006] Bernd Müller, **Java Server Faces Ein Arbeitsbuch für die Praxis**. 2006. ISBN-10: 3-446-40677-8

[JSF 2010] Bernd Müller, **Java Server Faces 2.0 Ein Arbeitsbuch für die Praxis**. 2010. ISBN: 978-3-446-41991-6

[JSF 2004] Andy Bosch, **Java Server Faces Das Standard-Framework zum Aufbau webbasierter Anwendungen**. 2004. ISBN 3-8273-2127-1

[ADG 2012] Android Developer Guid, http://www.developer.android.com/ 07.08.2012

[GS 2011] http://googlemobileads.blogspot.de/2012/01/new-research-global-surge-in-smartphone.html 3.07.2012

[JP1999] Gerhard Wilhelms, Markus Kopp, **Java professionell**. 1999. ISBN 3-8266-0395-8

[HDJP 2006] Guido Krüger, **Handbuch der Java-Programmierung** 4., aktualisierte Auflage 2006.

[Andr2010] Arno Becker, Marcus Pant, **Android 2 Grundlagen und Programmierung**. 2010 ISBN 978-3-89864-677-2

[Andn2011] Louis, Müller, **Android Der schnelle Einstig in die Programmierung und Entwicklungsumgebung**. 2011 ISBN 978-3-8272-4715-5

[JSPUS] M.Brantner, S.Schmidt, B. u D. Wabnitz, **Java Server Pages und Servlets**, 2001, ISBN:3-8158-2140-1

[Andr4] Mike Bach, **Apps entwickeln für Android 4 Das umfassende Training**.

[SAGKP] Oliver Vogel, Ingo Arnold, Arif Chughtai, **Software-Architektur: Grundlagen – Konzepte – Praxis**. 2008

[MIJ] Steffen Heinzl, Markus Mathes, **Middleware in Java**, 2005, ISBN: 978-3-528-05912-5

[DWZJP] Michael Inden, **Der Weg zum Java-Profi**, 2011, ISBN: 978-3-89864-668-0

[SFVA] J. Dunkel, A. Eberhart, S. Fischer, C. Kleiner, A. Koschel, **Systemarchitekturen für verteilte Anwendungen**, 2008

Versicherung über Selbstständigkeit

Hiermit versichere ich, dass ich die vorliegende Arbeit ohne fremde Hilfe selbstständig verfasst und nur die angegebenen Hilfsmittel benutzt habe.

Hamburg, den 30.08.2012 _____