Konzeption und Realisierung eines interaktiven Studienbegleiters für Android

Bachelorarbeit vorgelegt von Benjamin Thom

Angefertigt im Studiengang Angewandte Informatik
an der Hochschule für Angewandte Wissenschaften Hamburg

Erstprüferin oder Erstprüfer: Prof. Dr. Olaf Zukunft
Zweitprüferin oder Zweitprüfer: Prof. Dr. Bettina Buth

Vorwort

In der nachfolgenden Arbeit wird zur Wahrung der Übersichtlichkeit und Lesbarkeit ausschließlich die männliche Schriftform verwendet. Selbstverständlich schließt dies, sofern nicht anders kenntlich gemacht, das weibliche Geschlecht nicht aus.

Inhaltsverzeichnis

1 Einleitung

1.1 Motivation

Mobiler Zugang zu Informationen ist in vielen Bereichen längst zur Selbstverständlichkeit geworden. Navigation, Terminplanung und Social Networking sind nur einige der vielen Funktionen, die wir unseren elektronischen Begleitern heute gern abverlangen. Insbesondere mit der Entwicklung von Smartphones haben sich die Möglichkeiten zur grafischen Aufbereitung von Informationen und Vereinfachung der Bedienbarkeit enorm vergrößert. Mittlerweile haben sich unzählige individuelle Anwendungen als ganz neue Werkzeuge zur alltäglichen Information, Kommunikation und Organisation etabliert.

Auch im Studium lassen sich Smartphone-Anwendungen zur Orientierung sinnvoll einsetzen. Das Betriebssystem Android verfügt dazu bereits über sehr umfangreiche Bordmittel, wie zum Beispiel einen Kalender zum Planen von Veranstaltungsterminen, einen Browser zum Abrufen von Webinhalten und eine interaktive Karte, die standortbezogene Informationen liefert. Die einfache Integration dieser vorhandenen Werkzeuge in den Studienalltag scheitert jedoch weitgehend an dem zu geringen Fachbezug; das Grundangebot an Funktionen ist zwar bereits sehr komplex und auch zweckdienlich, der Anwendungsbereich Studienplanung wird aber nur unzureichend abgedeckt. So lassen sich beispielsweise Termine zu regelmäßig anstehenden Veranstaltungen leicht in den Android-Kalender einpflegen, zur Aufstellung eines Semesterplans ist dieser jedoch ungeeignet. Es fehlt nicht nur die Übersicht, wenn die Termine über einen größeren Zeitraum verteilt liegen, nachträgliche Änderungen können auch sehr umständlich sein. Um wichtige Informationen zur Orientierung im Studium zu erfassen und darzustellen, reichen die bestehenden Möglichkeiten nicht aus. Mit den vorhandenen Werkzeugen der Android-Plattform ist eine strukturierte, studienbegleitende Planung darum nur schwer möglich.

1.2 Zielsetzung

Gegenstand dieser Arbeit ist die Konzeptionierung und Entwicklung einer interaktiven Android-Anwendung, die exemplarisch den fachlichen Bezug zur Terminplanung herstellt, indem sie sich an den Bedürfnissen von Studenten der HAW Hamburg orientiert. Das Ziel ist es, basierend auf der Android-API ein erweiterbares System zu entwickeln, mit dem eine einfache und übersichtliche Studienplanung möglich ist. Die Übertragbarkeit der Lösung auf ähnliche fachliche Anwendungsbereiche soll dabei geprüft werden.

1.3 Inhaltlicher Aufbau der Arbeit

Im zweiten Kapitel dieser Arbeit werden zunächst die fachlichen Rahmenbedingungen für eine allgemeine studentische Semesterplanung erfasst. Danach folgt eine kurze Vorstellung grundlegender Kalenderspezifikationen, die in aktuellen Kalendersystemen Verwendung finden und sich zur Semesterplanung eignen. Im Anschluss wird auf die Nutzung mobiler Terminplaner eingegangen und dabei ein Einblick in den Android-Kalender gegeben. Der letzte Teil des zweiten Kapitels beschäftigt sich mit der Entwicklungsplattform und beschreibt einige relevante Framework-Komponenten.

Im weiteren Verlauf wird nach der Analyse des fachlichen Anwendungsbereiches in Kapitel 3 ein Anforderungskatalog aufgestellt, der die Grundlage für den Entwurf des Studienplaners bildet. Dieser ist Thema des vierten Kapitels. Die Vorgehensweise zur Implementierung eines testbaren Studienplaner-Prototyps beschreibt danach das fünfte Kapitel. Das letzte Kapitel gibt schließlich eine rückblickende Zusammenfassung der Ergebnisse und liefert einen Ausblick auf zukünftige Erweiterungen.

2 Grundlagen

2.1 Studienplanung

Die für diese Arbeit relevanten, fachlichen Rahmenbedingungen der studentischen Semesterplanung setzen sich zusammen aus der Festlegung über die Dauer eines Semesters (2.1.1), den verwendeten Veranstaltungsdaten (2.1.2), sowie der zugrunde liegenden Standardisierung der Kalenderwochen-Nummerierung (2.2.1).

2.1.1 Dauer eines Semesters

Für den Zeitraum eines Halbjahres werden seitens der Hochschule die Veranstaltungspläne erstellt. Daher ist die Festlegung auf Beginn und Ende eines Semesters für die Studienplanung wichtig.

Von der Hochschulrektorenkonferenz bestehen bereits seit einigen Jahren Bemühungen, die deutschen Semester- und Vorlesungszeiten mit denen des europäischen Auslands zu harmonisieren und im Zuge dessen bundesweit zu vereinheitlichen [6]. Man konnte sich bislang jedoch nicht auf eine Lösung einigen. Die folgende Stichprobe zeigt, dass auch bei der in Deutschland weitaus gebräuchlichsten Einteilung in Sommer- und Wintersemester die Semesterzeiten jeweils noch unterschiedlich sind:

Sommersemester 2011

Hochschule	Beginn	Ende	Vorlesungszeit
Uni Hamburg (Quelle: [7])	01.04.2011	30.09.2011	04.04.2011 - 16.07.2011
HAW Hamburg (Quelle: [8])	01.03.2011	31.08.2011	14.03.2011 - 15.07.2011
Uni Hildesheim (Quelle: [9])	01.04.2011	30.09.2011	04.04.2011 - 15.07.2011
Uni Kiel (Quelle: [10])	01.04.2011	30.09.2011	04.04.2011 - 22.07.2011
FH Kiel (Quelle: [11])	01.03.2011	31.08.2011	14.03.2011 - 01.07.2011
TU München (Quelle: [12])	01.04.2011	30.09.2011	02.05.2011 - 30.07.2011
Uni Köln (Quelle: [13])	01.04.2011	30.09.2011	04.04.2011 - 15.07.2011
FH Köln (Quelle: [14])	01.03.2011	31.08.2011	07.03.2011 - 08.07.2011
TU Dresden (Quelle: [15])	01.04.2011	30.09.2011	04.04.2011 - 16.07.2011
Uni Mannheim (Quelle: [16])	01.02.2011	31.07.2011	14.02.2011 - 03.06.2011

Tabelle 2.1: Stichprobe der Semesterzeiten- und Vorlesungszeiten an deutschen Hochschulen – Sommersemester 2011 (Uni Mannheim: Frühjahrssemester 2011)

Wintersemester 2011/2012

Hochschule	Beginn	Ende	Vorlesungszeit
Uni Hamburg (Quelle: [7])	01.10.2011	31.03.2012	17.10.2011 - 04.02.2012
HAW Hamburg (Quelle: [8])	01.09.2011	29.02.2012	19.09.2011 - 03.02.2012
Uni Hildesheim (Quelle: [9])	01.10.2011	31.03.2012	17.10.2011 - 10.02.2012
Uni Kiel (Quelle: [10])	01.10.2011	31.03.2012	17.10.2011 - 17.02.2012
FH Kiel (Quelle: [11])	01.09.2011	29.02.2012	12.09.2011 - 10.02.2012
TU München (Quelle: [12])	01.10.2011	31.03.2012	17.10.2011 - 11.02.2012
Uni Köln (Quelle: [13])	01.10.2011	31.03.2012	10.10.2011 - 03.02.2012
FH Köln (Quelle: [14]))	01.09.2011	28.02.2012	19.09.2011 - 10.02.2012
TU Dresden (Quelle: [15])	01.10.2011	31.03.2012	10.10.2011 - 04.02.2012
Uni Mannheim (Quelle: [16])	01.08.2011	31.01.2012	05.09.2011 - 09.12.2011

Tabelle 2.2: Stichprobe der Semesterzeiten- und Vorlesungszeiten an deutschen Hochschulen – Wintersemester 2011/12 (Uni Mannheim: Herbstsemester 2011)

An den meisten deutschen Hochschulen erstrecken sich die Sommersemester vom 1. April bis zum 30. September und die Wintersemester vom 1. Oktober bis zum 31. März des Folgejahres. Im Gegensatz dazu beginnen an den meisten Fachhochschulen in Deutschland die Semester jeweils einen Monat früher. Die Vorlesungszeit beginnt meist in der Mitte des ersten Semestermonats. Sie dauert im Sommer etwa 14 und im Winter etwa 15 Wochen [17]. Die Universität Mannheim hat ihre Semesterzeiten als erste staatliche Hochschule vor einigen Jahren bereits an die international gebräuchlichen Frühjahrs- und Herbstsemester angepasst.

2.1.2 Veranstaltungsdaten

Der Veranstaltungsplan für ein Studiensemester besteht im Wesentlichen aus Serienterminen. Zu den Daten, die der Plan unabhängig vom Studienfach enthält, zählen:

- Semester
- Veranstaltungsbezeichnungen
- Beginn- und Endezeiten
- Angaben zu Dozenten
- Veranstaltungsorte

Diese Daten werden üblicherweise zu Semesterbeginn von der Hochschule in schriftlicher oder elektronischer Form bereitgestellt. Ein einheitliches Format für einen Hochschul-Veranstaltungsplan existiert nicht; es variiert je nach Hochschule und Studienrichtung.

So werden im Fachbereich Informatik an der HAW Hamburg die einzelnen Termine zur Planung jeweils mit einer Reihe von Kalenderwochen veröffentlicht [18], in anderen Fachbereichen werden hingegen konkrete Daten mit Tages- und Monatsangaben für die Veranstaltungen herausgegeben [19]. Unabhängig davon werden die Termine in der Regel der Einfachheit halber einem festen Wochentag zugeordnet.

2.2 Kalenderspezifikationen

In diesem Abschnitt werden die beiden wichtigsten, international gebräuchlichen Standardisierungen für Kalenderinformationen hinsichtlich ihrer Eignung zur Verarbeitung von Serienterminen untersucht.

2.2.1 Kalenderwochen nach ISO 8601

Die Planung nach Kalenderwochen eignet sich für Serientermine sehr gut. So lassen sich die Termine kompakt darstellen und einfach bearbeiten. Statt einer Menge einzelner konkreter Datumsangaben wird zu den Kalenderwochen nicht mehr als der Wochentag benötigt, um die einzelnen Tage eines Serientermins für ein ganzes Studiensemester zu erfassen. Zur Berücksichtigung von Ausnahmen kann ein Einzeltermin über die entsprechende Kalenderwoche herausgenommen werden.

Die numerische Repräsentation von Datumsangaben ist in der weltweit verwendeten Norm ISO 8601 [7] festgehalten. Seit 2006 wird diese auch in Deutschland verwendet. Ein Jahr umfasst nach diesem internationalen Standard entweder 52 oder 53 Kalenderwochen. Dies ergibt sich aus den folgenden Festlegungen: Zunächst wird für jede Woche ein fester Zeitraum von sieben Tagen angesetzt. Dabei gilt hier – im Gegensatz zu einigen religiösen Traditionen – jeweils der Montag als erster Wochentag. Weiterhin gibt es die Festlegung, dass die erste Woche eines Kalenderjahres mindestens vier Tage umfasst. Bis zu drei Tage eines neuen Jahres können somit noch zur letzten Kalenderwoche des Vorjahres gehören. Die erste Kalenderwoche enthält also immer den 4. Januar und immer den ersten Donnerstag im Jahr. Es ist damit eindeutig festgelegt, dass eine Kalenderwoche durch einfaches Zählen von Donnerstagen ermittelt werden kann – innerhalb eines Jahres können dies minimal 52 und maximal 53 sein. Der ISO 8601 Standard bildet insgesamt eine solide Grundlage für die

Integration von entsprechenden Kalenderoperationen in Software, wie zum Beispiel auch in den Klassen java.util.Calendar und android.text.format.Time.

2.2.2 Der iCalendar-Standard

Eine umfangreiche Spezifikation zur Vereinheitlichung von Kalenderinformationen wurde 1998 mit dem iCalendar-Standard [8] hervorgebracht. Er findet unter anderem Verwendung in Apple iCal, IBM Lotus Notes, Microsoft Outlook und einer Vielzahl weiterer Anwendungen. Im Folgenden wird gezeigt, dass er komfortable Möglichkeiten bietet, um für einen Termin verschiedene Wiederholungsregeln zu definieren.

Ein Kalender enthält zunächst mehrere sogenannte VEVENT-Komponenten, welche die Termineinträge repräsentieren. Diese Komponenten fassen jeweils alle notwendigen Eigenschaften wie Bezeichnung, Beginn und Ende eines Termins zusammen, enthalten zudem aber auch viele weitere, optionale Angaben wie Kategorien und Alarmeinstellungen, die mit dem Termin verknüpft sind. Mit der RRULE-Eigenschaft kann für ein solches VEVENT eine Wiederholungsregel festgelegt werden. Das folgende Beispiel aus der Spezifikation [8] definiert den Kalendereintrag für einen Geburtstag:

```
BEGIN:VEVENT
UID:19970901T130000Z-123403@example.com
DTSTAMP:19970901T130000Z
DTSTART;VALUE=DATE:19971102
SUMMARY:Our Blissful Anniversary
TRANSP:TRANSPARENT
CLASS:CONFIDENTIAL
CATEGORIES:ANNIVERSARY,PERSONAL,SPECIAL OCCASION
RRULE:FREQ=YEARLY
END:VEVENT
```

Zu einer Wiederholungsregel erlaubt der iCalendar-Standard zusätzlich Ausnahmen zu speichern. In der sogenannten EXDATE Eigenschaft können Ausnahmen enthalten sein, welche bei der Berechnung der Wiederholungsdatenmenge aus einem RRULE-Wert berücksichtigt werden.

Wenn jedoch Funktionen fehlen, um die Ausnahmen zu einer Terminserie direkt erfassen zu können, bringt diese Eigenschaft beim Erstellen von Wiederholungsterminen keinen Vorteil. In den nachfolgend vorgestellten Kalenderanwendungen ist dies der Fall; erst wenn ein

Einzeltermin nachträglich aus der Serie gelöscht wurde, erhält die EXDATE-Eigenschaft einen entsprechenden Wert.

Der Microsoft Outlook Kalender orientiert sich eng am iCalendar Standard und beinhaltet Import- und Exportfunktionen für iCalendar-Dateien. Die Definition von Terminserien ist sehr einfach möglich; Abbildung 2.3 zeigt das verwendete Auswahlfenster. Es besteht hier allerdings keine Möglichkeit Ausnahmen zu erfassen.

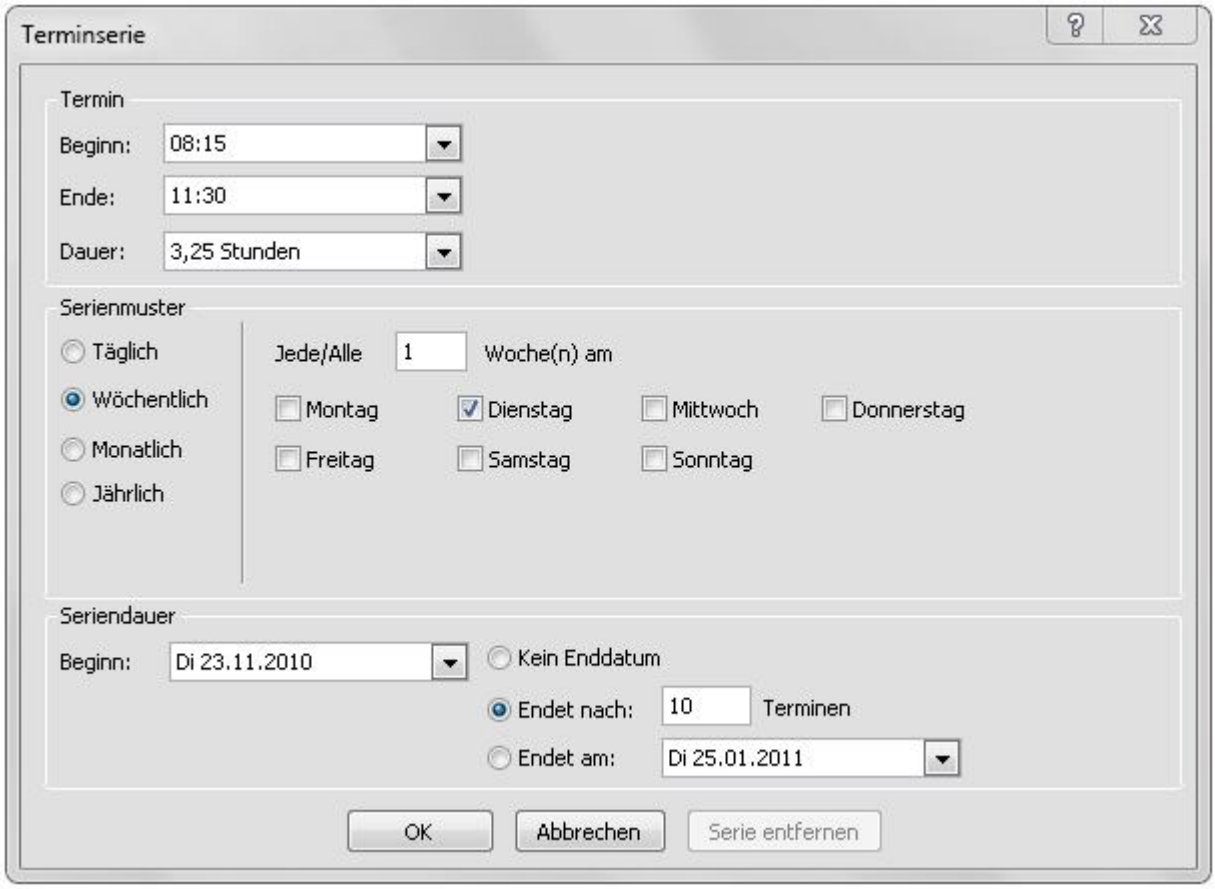

Abbildung 2.3: Definition einer Terminserie in Microsoft Outlook 2007

Auch von der Webapplikation Google Kalender wird der Import und Export von iCalendar-Termindaten unterstützt [11]. Ausnahmen zu einer Terminserie anzugeben ist hier ebenfalls nicht direkt möglich (Abbildung 2.4).

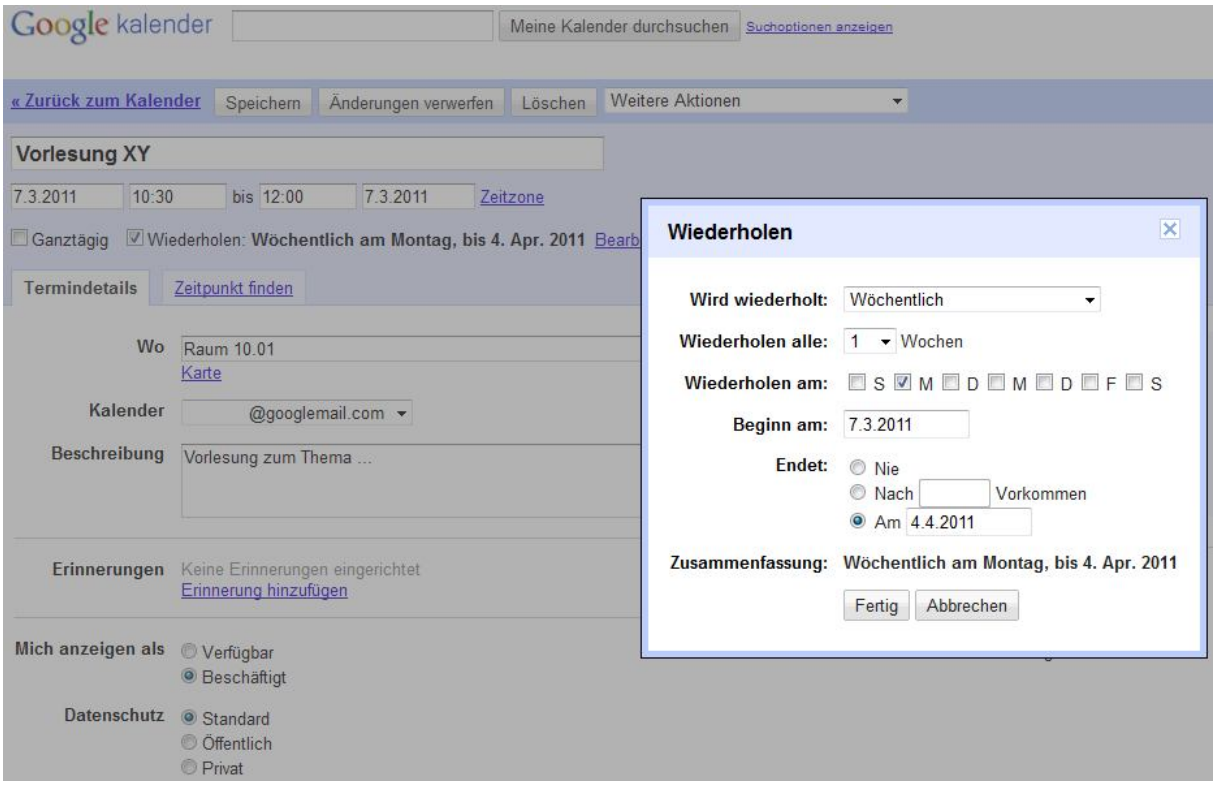

Abbildung 2.4: Definition einer Terminserie im Google Kalender

Dieser Kalender kann allerdings direkt mit dem Android-Kalender synchronisiert werden. Das ist nützlich, denn zum direkten Import und Export von iCalendar-Daten bietet der Android-Kalender bislang keine eigenen Lösungen. Hierzu kann höchstens auf spezielle Tools zurückgegriffen werden, die über den Android Market erhältlich sind.

Zusammenfassend lässt sich sagen, dass seine weite Verbreitung und die dargestellten Möglichkeiten zur Definition von Serien-Terminen große Vorteile bei der Verwendung des iCalendar-Standards sind. Die Verfügbarkeit entsprechender Schnittstellen erleichtert die Integration eines Terminplaners in viele vorhandene Systeme. Allerdings bieten selbst ausgereifte und weitverbreitete Anwendungen, die den Standard verwenden, keine Möglichkeit, bei der Erfassung von Serienterminen auch Ausnahmen mit anzugeben.

2.3 Mobile Terminplaner

Das Handy bietet sich als ständiger Begleiter für die Terminplanung natürlich an. Dass der Nutzwert der Geräte als mobile Kalender im Taschenformat längst erkannt ist, bestätigt auch eine statistische Erhebung der BITKOM/Forsa, nach der die Kalenderfunktion zu den drei meistgenutzten Handy- Funktionen zählt (Stand: August 2010):

Abbildung 2.5: Ein Drittel der Handybesitzer nutzt den Kalender. Quelle: [9]

Eine weitere Umfrage von Forsa ergab, dass in der Altersgruppe der 14- bis 29-Jährigen bereits jeder dritte Handynutzer ein Smartphone besitzt [10]. Da sich die Modelle der neuen Handygeneration in der Leistung und Bedienbarkeit von klassischen Mobiltelefonen enorm abheben, eröffnen sich zur individuellen Terminplanung mit entsprechender Software nochmals komfortablere Möglichkeiten. Da bei den Android-Geräten auch die gesamte Betriebssystem-API für den Entwickler offengelegt wurde, lassen sich die Terminplaner-Systeme dort ideal dem fachlichen Anwendungsbereich anpassen.

2.3.1 Der Android-Kalender

Im Folgenden wird ein kurzer Überblick über die Funktionen des Android-internen Kalenders zur Darstellung und Verwaltung von Terminen gegeben. Er zeichnet sich zunächst durch eine gute, intuitive Bedienbarkeit aus. Es ist sehr einfach Termine anzulegen und sich nützliche Erinnerungen einzustellen. Die Erfassung von Wiederholungsevents ist jedoch teilweise

umständlich, und es fehlen Möglichkeiten die Daten des fachlichen Anwendungsbereiches kompakt und übersichtlich anzuzeigen.

Für die grafische Oberfläche des Kalenders wurden eine interaktive Monats-, Wochen- und Tagesansicht implementiert (Abb. 2.6-2.8), sowie eine vollständige Terminübersicht, die alle eingetragenen Termine chronologisch sortiert in einer scrollbaren Liste aufführt (Abb. 2.9). Einträge für gleiche Zeiten sind in der Wochen- und Tagesansicht nebeneinander dargestellt.

Abb. 2.6: Android-Kalender: Monatsansicht

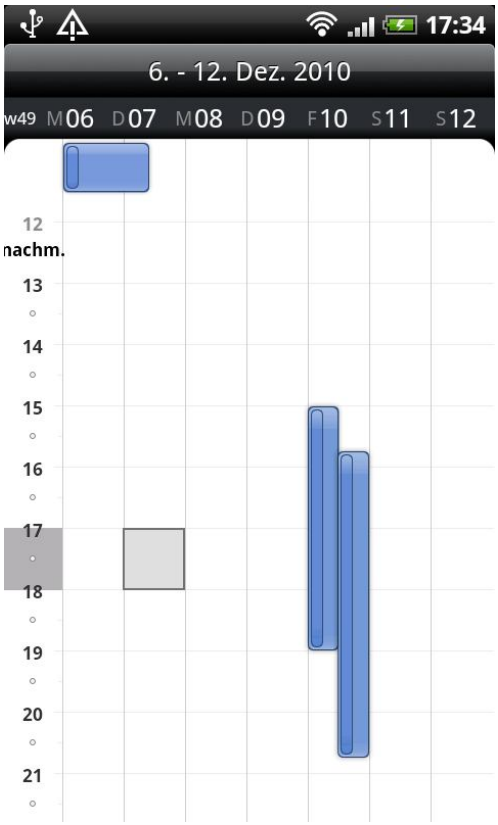

Abb. 2.7: Android-Kalender: Wochenansicht

Über das Optionsmenü oder per Touchevent lassen sich die Bildschirmseiten wechseln. Ein Kontextmenü erlaubt das Anlegen, Bearbeiten und Löschen von Einträgen zum gewählten Datum. Gerade bei vielen unterschiedlichen Terminen ist die Kombination aus verschiedenen Darstellungen und der einfach gestalteten Interaktion sehr praktisch. Möchte man jedoch – wie für einen Semesterplan – hauptsächlich Wiederholungstermine verwalten, wird es schnell unübersichtlich. Die Monatsansicht zeigt lediglich einen kleinen Indikatorbalken für die Tageszeiten der Einträge an. Wurden mehrere Termine für einen Tag erfasst, so kann man dies bestenfalls noch erkennen, wenn diese zeitlich auseinander liegen. Wiederholungstermine, die auf dem gleichen Wochentag liegen, lassen sich in der Monatsansicht also auch nur schwer auseinanderhalten. In der Wochenansicht ist es ähnlich;

auch hier fehlen zur eindeutigen Identifikation eines Termins noch inhaltliche Informationen. Erst auf der Tagesansicht lassen sich anhand der Terminbeschreibungen eindeutig mehrere Wiederholungstermine auseinanderhalten.

Abb. 2.8: Android-Kalender: Tagesansicht

Abb. 2.9: Android-Kalender: Terminübersicht

Außer einer Bezeichnung und dem Ort sind jedoch keine weiteren Informationen in der Tagesansicht aufgeführt. Von Einzelterminen lassen sich Wiederholungstermine selbst in dieser Ansicht nicht unterscheiden. Eingestellte Erinnerungen sind ebenfalls erst nach einem Klick auf einen Eintrag sichtbar.

Etwas transparenter stellt die Terminübersicht die Veranstaltungsdaten dar (Abb. 2.9). Doch einen detaillierten Überblick über mehrere, verschiedene Serientermine liefert diese ebenso wenig wie die anderen drei Darstellungen.

Die wohl größte Schwäche des Kalenders offenbart sich allerdings bei der Eingabe der Serientermine (Abb. 2.10 und 2.11). Aktuell können keinerlei Ausnahmen für die verfügbaren Wiederholungsregeln angegeben werden.

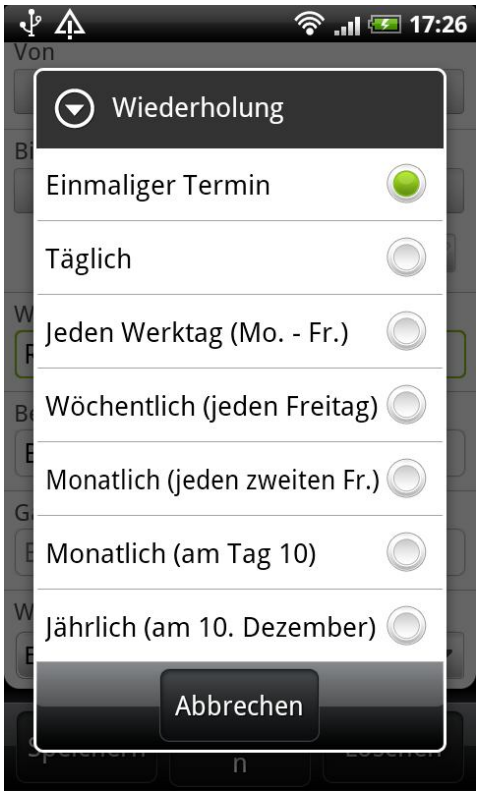

Abbildung 2.10 und 2.11: Android-Kalender: Das Eingabeformular erlaubt keine Angabe von Ausnahmen zu Wiederholungsregeln

Es ist nachvollziehbar, dass damit die Planung von Terminserien, die beispielsweise aufgrund von Feiertagen Lücken enthalten, sehr umständlich ist. Sämtliche Termine, an denen beispielsweise eine wöchentliche Veranstaltung nicht stattfindet, müssen zunächst einzeln im Kalender herausgesucht und wieder aufwendig aus der Terminserie entfernt werden. Nur über diesen Umweg können die Ausnahmen zu einer Wiederholungsregel definiert werden.

2.4 Entwicklung für Android

Im Folgenden werden einige für diese Arbeit relevante Aspekte zur Entwicklung für die Zielplattform Android erläutert. Einführend wird die Architektur des Betriebssystems dargestellt und im Anschluss der Aufbau einer Anwendung anhand einiger ausgewählter Kernkomponenten der Plattform beschrieben. Für ausführlichere Informationen sei an dieser Stelle auf den Android Developer Guide [1] verwiesen, dem die Informationen im Wesentlichen entnommen sind.

2.4.1 Architektur des Betriebssystems

Die Android-Plattform besteht aus einem Software-Stack mit vier Schichten (Abb. 2.12). Ein wesentliches Merkmal der Architektur ist die komponentenbasierte Struktur, die eine einfache Wiederverwendbarkeit von vorhandenen Komponenten für die eigene Anwendung gewährleistet. Den Kern des Betriebssystems bildet ein Linux-Kernel, der die Gerätetreiber enthält und als Abstraktionsschicht zwischen Hardware und restlichem Stack gesehen werden kann. Die Laufzeitumgebung setzt sich aus der sogenannten Dalvik Virtual Machine (DVM) und den Java-Kernbibliotheken zusammen, die als Basis für das Application Framework dienen.

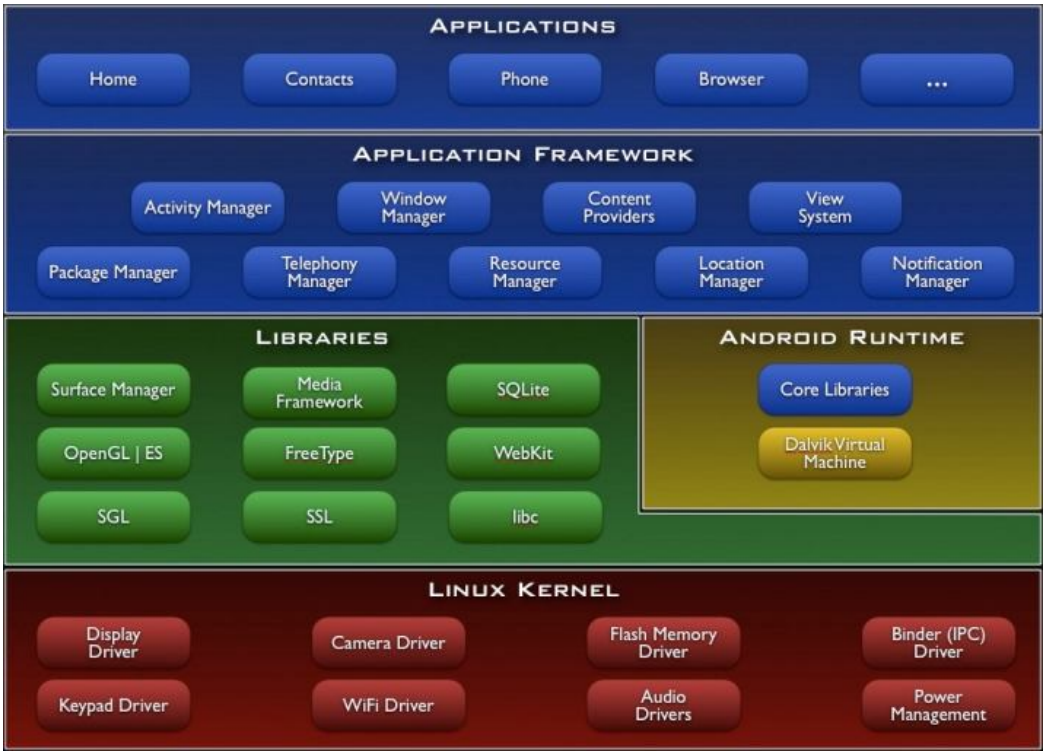

Abbildung 2.12: Die Architektur von Android ist auf eine einfache Wiederverwendbarkeit vorhandener Komponenten ausgelegt. Quelle: [12]

Mit der DVM wurde in das System eine effizientere Weiterentwicklung der Java Virtual Machine (JVM) eingebunden, die den Anforderungen mobiler Geräte gerecht wird. Sie verwendet im Gegensatz zur klassischen JVM beispielsweise die verfügbaren, schnellen Prozessorregister und verbraucht nur sehr wenig Speicher. Letzteres ist wichtig, da Android nach einem Sandbox-Prinzip arbeitet; jede Anwendung wird mit einer eigenen DVM in einem eigenen Linux-Prozess gestartet. Damit ist der verwendete Hauptspeicher vor dem Zugriff von außen geschützt. Unerlaubte Zugriffe auf anwendungseigene, im Dateisystem liegende Daten werden über das Linux-Berechtigungssystem verhindert. Alle von einer Anwendung benötigten Rechte zur Verwendung anderer Komponenten müssen explizit im sogenannten Manifest (siehe Abschnitt 2.4.2) vergeben werden.

Das eigentliche Android Application Framework bildet eine weitere Architekturschicht. Sie besteht aus den Java-Klassen, die dem Entwickler letztendlich zur Verfügung stehen. Sämtliche, für den Betrieb einer Android-Anwendung notwendigen Funktionalitäten werden diesen Klassen durch eine Menge von C/C++ Standardbibliotheken der darunterliegenden Schicht zur Verfügung gestellt; dazu zählen beispielsweise eine 2D- und 3D-Grafik-Engine, eine Bibliothek, die das Aufnehmen und Abspielen von Multimediaformaten ermöglicht, eine Browserumgebung für den Webzugriff, sowie eine Datenbank, welche in Abschnitt 2.4.8 noch genauer vorgestellt wird. Die oberste Schicht in der Architektur bilden schließlich die eigenen und alle bereits in die Plattform integrierten Anwendungen, wie beispielsweise der in Abschnitt 2.3.1 beschriebene Kalender, das Telefon, der Browser, die Kamera, die Kontaktverwaltung und vieles mehr.

2.4.2 Das Manifest

Ohne die Angaben in der Manifest-Datei ist eine Android-Anwendung nicht lauffähig. Es müssen hier alle Kernkomponenten aufgeführt werden, aus denen sie sich zusammensetzt. Neben dem jeweiligen Klassennamen werden auch wichtige Eigenschaften genannt. Die Laufzeitumgebung muss wissen, welche Operationen auf einer Komponente anwendbar sind. Zu dem Zweck werden sogenannte Intent Filter definiert, die entsprechende Intents (siehe Abschnitt 2.4.5) beschreiben. Weiterhin sind im Manifest zum Beispiel die Prozesse, in denen die Komponenten laufen, zusätzliche Bibliotheken, die eingebunden werden, Themes oder Titel für Bildschirmseiten angegeben. Eine andere, wichtige Funktion ist die Rechtevergabe. Sämtliche Rechte, die zur Verwendung anderer Komponenten benötigt werden, sowie die von

anderen Komponenten zur Verwendung der Anwendungskomponenten erforderlichen Rechte, müssen vollständig aufgeführt werden. Außerdem sind noch einige unbedingt notwendige Informationen über die Anwendung, wie der Java Package Name, die aktuelle Version und der benötigte API-Level im Manifest enthalten.

2.4.3 Views und eigene Layouts

Alle Elemente zur Gestaltung der grafischen Benutzeroberfläche einer Android-Anwendung leiten von der Klasse View ab. Dazu gehören einfache Objekte wie Buttons und Textfelder, aber auch komplexere, welche wiederum andere Views enthalten. Letztere sind Erweiterungen der von View ableitenden, abstrakten Klasse ViewGroup. Von dieser erben alle vorhandenen Layout-Klassen, wie zum Beispiel LinearLayout, RelativeLayout und GridLayout. Eine (eigene) ViewGroup implementiert Methoden, um festzustellen, ob eine enthaltene Child-View darstellbar ist, um die Pixelbreite und -höhe jeder darstellbaren Child-View zu bestimmen und um allen die ermittelten Abmessungen und Positionen zuzuweisen, mit denen sie sich zeichnen können.

2.4.4 Activities und Subactivities

Mit Activities werden grafische Benutzeroberflächen verwaltet und Interaktionen mit dem Benutzer ermöglicht. Meist beinhalten Android-Anwendungen als Programmeinstiegspunkt eine Activity, die das erste User Interface anzeigt. Sehr oft wird der gesamte Lebenszyklus einer Anwendung durch den ihrer Activities (siehe Abschnitt 2.4.9) bestimmt. Zu den grundlegenden Aufgaben von Activities zählen beispielsweise das Darstellen sämtlicher Grafikelemente einer Bildschirmseite, das Auslesen von Formularen, die Reaktion auf alle Arten von Benutzereingaben, die Initialisierung von Options- und Kontextmenüs, sowie das geeignete Verhalten bei einer Menüauswahl. Zur Erledigung ihrer Aufgaben kann eine Activity auch wiederum andere Activities starten. Die Kommunikation zwischen Ihnen geschieht dabei mittels sogenannter Intents, auf die im nächsten Abschnitt ausführlich eingegangen wird.

Wenn eine Activity ausschließlich dazu dient bestimmte Ergebnisse an die aufrufende Instanz zurückzuliefern, spricht man bei ihr auch von einer Subactivity. Beim Start wird ihr dazu ein spezieller Request-Code mitgeteilt. Nach ihrer Beendigung kann der Aufrufer dann in einer

Callback-Methode anhand dieses Codes die zurückgelieferten Daten identifizieren und sie entsprechend eines zusätzlichen Result Codes auswerten. Der Result Code gibt in der Regel Aufschluss über den Erfolg oder Misserfolg des Aufrufs. Falls zum Beispiel die Subactivity aufgrund eines Absturzes unerwartet beendet wird, erhält der Aufrufer per Callback einen Result Code, der ihn über diesen Abbruch informiert. Der Entwickler braucht lediglich in der Subactivity das Ergebnis setzen, die Rückgabe an die aufrufende Instanz erledigt dann das Betriebssystem.

2.4.5 Intents

Mit sogenannten Intents wurde eine einheitliche Beschreibung des Informationsaustausches unter Android-Komponenten geschaffen. Sie beschreiben jeweils eine gewünschte Operation und transportieren die Daten, mit denen diese ausgeführt werden soll. Der Absender spricht entweder einen bestimmten, oder mehrere unbestimmte Empfänger an. Wird der Empfänger beim Versenden ausdrücklich genannt, spricht man auch von expliziten Intents. Ohne die konkrete Angabe eines Empfängers kann hingegen jede Komponente jeder installierten Anwendung auf ihre Art reagieren. So besteht zum Beispiel auch für das Betriebssystem die Möglichkeit, das Auftreten spezieller Systemereignisse zu signalisieren, ohne festzulegen, für welche Anwendungen diese von Bedeutung sind. Bei solchen sogenannten impliziten Intents versucht Android einen geeigneten Empfänger zu finden. Dieser Vorgang nennt sich Intent Resolution. Komponenten legen hierzu selbst fest, auf welche Nachrichten sie reagieren. In einem sogenannten Intent Filter – der zum Beispiel im Manifest (vgl. 2.4.2) deklariert werden kann, ist die genaue Aktion spezifiziert, die die Komponente ausführt, die Bedingungen, unter denen diese Aktion stattfindet, sowie die Daten, die zur Ausführung benötigt werden.

Das verzögerte Ausführen per Intent definierter Aktionen ist mit Hilfe sogenannter Pending Intents möglich. Über die Factory-Methode `PendingIntent.getBroadcast(…)` wird zum Beispiel ein Pending Intent zum verzögerten Versenden eines Broadcast Intent erzeugt. Wird er einer anderen Anwendung übergeben, kann er dort mit den Rechten seines Erzeugers die gewünschte Aktion ausführen.

Broadcast Intents werden hauptsächlich vom System versendet, um über spezielle Ereignisse wie zum Beispiel das Abschalten des Bildschirms, einen niedrigen Akkuladestand oder den Abschluss des Bootvorgangs zu benachrichtigen. Mit Hilfe eines Broadcast Receivers (vgl. 2.4.6) kann eine Anwendung geeignet auf diese Intents reagieren.

2.4.6 Broadcast Receiver

Die einzige Aufgabe eines Broadcast Receivers ist es, während seiner Lebensdauer auf bestimmte Broadcast Intents zu reagieren. Er kann zum einen statisch im Android-Manifest (siehe 2.4.2) deklariert, zum anderen dynamisch, zum Beispiel in einer Activity erzeugt werden. Wird er mittels Manifest in eine Anwendung eingebunden, kann er in einem eigenen Prozess laufen und völlig unabhängig davon, ob die Anwendung läuft oder nicht, auf die erwarteten Intents reagieren. Für die impliziten Intents, auf die der Receiver reagieren kann, wird ein IntentFilter entweder vom Erzeuger definiert, oder bei der Deklaration direkt im Manifest mit angegeben. Ein Anwendungsbeispiel für einen Broadcast Receiver ist die Wiedergabe eines akustischen Signals, sobald der AlarmManger (siehe 2.4.7) einen bestimmten Alarm versendet. In seiner Callback-Methode `onReceive()` empfängt der Receiver den Alarm, also die Broadcast-Message, in Form eines Intent und verarbeitet ihn. Nach dem Verlassen der Methode hat der Receiver seine Arbeit beendet.

2.4.7 Alarm Manager

Möchte man eine Anwendung zu einem bestimmten Zeitpunkt ausführen, kann man dazu den AlarmManager verwenden. Dieser löst einen Alarm aus, indem er einen Intent per Broadcast versendet, der mit der Alarmzeit vorgemerkt wurde. Über den Anwendungskontext per `getSystemService(Context.ALARM_SERVICE)`erhält man eine Referenz auf den AlarmManager. Zum Setzen eines Alarms akzeptiert er einen PendingIntent (siehe 2.4.5), eine Zeit für die Ausführung, sowie einen Alarmtyp. Ist bereits ein PendingIntent für den gleichen Intent eingeplant, so wird er mit der neuen Alarmzeit überschrieben. Mehrere gleiche Alarme können nur bei einem festen Wiederholungsintervall auf einmal vorgemerkt werden. Um auch unregelmäßige Abstände zwischen gleichen Intents zu berücksichtigen, muss man allerdings die entsprechenden Alarme einzeln nacheinander einstellen.

Die Reaktion auf einen Alarm erfolgt in einem BroadcastReceiver (siehe 2.4.6). Solange dieser sich in der `onReceive()` Methode befindet, hält der AlarmManager einen Wakelock. Die Ausführung kann damit nicht dadurch unterbrochen werden, dass sich das Gerät in einen Energiesparmodus schaltet. Nach Verlassen der Methode wird der Wakelock allerdings sofort wieder freigegeben. Darum muss er in asynchronen Threads, die ein Receiver zur Reaktion auf einen Alarm startet, für die benötigte Zeit weiter aufrecht erhalten werden.

Die zu jedem Alarm angegebene, bereits erwähnte Alarmzeit wird in Millisekunden angegeben, und der Typ gibt an, ob diese nur seit dem letzten Bootvorgang oder als UTC (koordinierte Weltzeit) gewertet wird. Außerdem entscheidet der Typ darüber, ob das Gerät durch den Alarm auch aus dem Sleep-Zustand aufgeweckt werden kann, oder ob erst darauf gewartet wird bis das Gerät sich wieder in einem aktiven Zustand befindet. Die eingestellten Alarme werden jedoch in jedem Fall bis zu ihrer Ausführung oder Löschung vorgemerkt; nur wenn das Gerät komplett ausgeschaltet wird, gehen sie verloren. Liegt die Zeit für einen Alarm in der Vergangenheit, so wird er sofort ausgelöst.

2.4.8 SQLite-Datenbank

Zur Persistierung von Daten beinhaltet Android eine SQLite-Datenbank – eine leichtgewichtige SQL-Datenbank, die vor allem für den Embedded-Einsatz entworfen wurde und heute in einer Vielzahl von Anwendungen eingesetzt wird [13]. Es müssen zwar auf dem Android-Gerät keine großen Datenmengen für den effizienten, zeitgleichen Zugriff mehrerer Anwendungen bereitgehalten werden. Das Datenbanksystem erfüllt hier vielmehr einen anderen Zeck. Es wird verwendet, um strukturierte Daten dauerhaft in einer Datei zu speichern.

Das System benötigt keinen Server und eine aufwändige Installation oder Konfiguration entfällt [2]. Eine Datenbank wird jeweils für eine Anwendung im Dateisystem unter „`/data/data/packagename/databases`“ als Datei abgelegt, und die Zugriffsrechte ergeben sich aus den Lese- und Schreibberechtigungen für diese Datei. Separate Rechtevergabe für einzelne Tabellen oder Benutzer sind nicht vorgesehen.

Um aus der eigenen Anwendung heraus einen einfachen Zugriff auf die Datenbank zu ermöglichen, steht im Android-Framework die abstrakte Klasse SQLiteOpenHelper zur Verfügung. Diese hält eine Referenz auf ein SQLiteDatabase-Objekt, um den Lese- oder Schreibzugriff auf die Datenbank zu gewähren. Der SQLiteOpenHelper prüft zudem bei jedem Zugriff, ob die Datenbank nicht existiert und daher neu angelegt, oder ob die Version des Datenbankschemas aktualisiert werden muss. Eine Erweiterung der Klasse enthält daher die beiden folgenden Methoden, in der die SQL-Anweisungen für die Erstellung und das Upgrade des Datenbankschemas definiert sind:

- `onCreate(SQLiteDatabase db)`
- `onUpgrade(SQLiteDatabase db, int oldVer, int newVer)`

Die für sämtliche Aufgaben des Datenbankmanagement benötigte, umfangreiche Schnittstelle zur Datenbank bietet die Klasse SQLiteDatabase an. Auch Datenbankabfragen werden über sie abgewickelt. Selektionsanfragen werden beispielsweise mit der Methode `query()` gestellt. Diese liefern dann einen Cursor auf die Ergebnismenge zurück. Für einzelne SQL-Statements, die keine Daten zurückliefern, kann die Methode `execSQL()` verwendet werden. Per `compileStatement()` lassen sich wiederverwendbare SQL-Statements kompilieren, also Prepared Statements generieren, die über ein `execute()` ausgeführt werden können. Darüber hinaus können auch Transaktionen mittels `beginTransaction()` und `endTransaction()` kontrolliert werden. Über diese Schnittstelle ist also eine sehr kompakte und überschaubare Verwaltung einer Android-Datenbank möglich.

2.4.9 Activity-Lifecycle

Ein Activity durchläuft in ihrem Lebenszyklus von der Erzeugung bis zu ihrer Zerstörung verschiedene Zustände (siehe Abbildung 2.13). An den Zustandsübergängen werden von der Laufzeitumgebung jeweils Hook-Methoden aufgerufen, deren Funktionalität der Entwickler für eigene Zwecke erweitern kann. Auf diese Weise kann er an geeigneten Stellen in den Aufrufzyklus eingreifen.

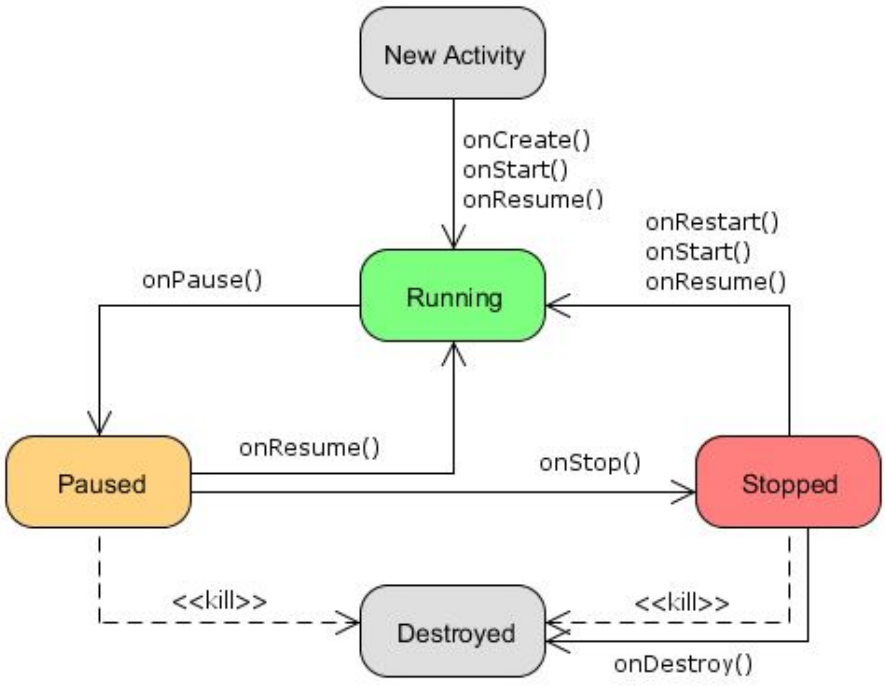

Abbildung 2.13: Der Lebenszyklus einer Android Activity

Eine neu erstellte Activity befindet sich zunächst im ausführenden Zustand, und ihre Bildschirmoberfläche im Vordergrund. Die Methode `onCreate()` wird für eigene Klassen im Allgemeinen immer erweitert, sie hat die Funktionalität eines Konstruktors. Die nachfolgend durchlaufenen Methoden `onStart()` und `onResume()` werden sowohl nach jeder Neuerzeugung aufgerufen, als auch dann, wenn eine Activity wieder vollständig sichtbar wird, die sich im Hintergrund befindet. In den Hintergrund gelangt sie, wenn sie von einer anderen Activity überlagert wird. Sie verliert dann den Focus und reagiert nicht mehr auf Benutzereingaben. Dies kann zum Beispiel beim Einblenden von Dialogen oder eingehenden Anrufen der Fall sein. Damit befindet sie sich in einem pausierten Zustand. Diesen verlässt sie über `onResume()`, wenn sie wieder in den Vordergrund kommt und auf Benutzereingaben reagiert. Wird eine Activity – wie bei der Betätigung des Home-Buttons auf dem Gerät – vollständig unsichtbar, wird sie vom System gestoppt. Sowohl im pausierten, als auch im gestoppten Zustand sind die Statusinformationen der Instanz nicht verloren, sie befinden sich weiterhin im Hauptspeicher. Eine Activity, die sich in einem dieser Zustände befindet, kann jedoch bei extrem geringer Speicherverfügbarkeit automatisch vom System beendet werden. In diesem Extremfall ist es möglich, dass die Methoden `onStop()` und `onDestroy()` vor der Zerstörung gar nicht mehr aufgerufen werden. Wird sie allerdings regulär beendet, so erfolgt in jedem Fall unmittelbar vor der Entfernung aus dem Speicher der Aufruf von `onDestroy()`. Erst danach gibt sie die von ihr reservierten Systemressourcen wieder vollständig frei.

2.4.10 Asynchrone Tasks / Threadsicherheit

Android verwendet ein Single-Thread-Modell; beim Start einer Anwendung wird genau ein „main“-Thread erstellt – auch UI-Thread genannt. In diesem Thread erledigt die Anwendung prinzipiell sämtliche Aufgaben; vom Zeichnen der grafischen Oberfläche, über die Interaktion mit dem Benutzer, bis hin zu Datenbankabfragen und anderen, zum Teil sehr zeitaufwendigen Operationen. Bei länger andauernden Aufgaben kann es deshalb beispielsweise vorkommen, dass die Anwendung nicht mehr auf Benutzereingaben reagiert. Falls der UI Thread länger als fünf Sekunden nicht reagiert, zeigt Android einen sogenannten ANR-Dialog an – „application not responding“. Der Entwickler sollte also von Hintergrund-Threads Gebrauch machen, um den UI Thread zu entlasten. Das Android UI toolkit ist allerdings nicht thread-safe, das heißt UI-Manipulationen außerhalb des UI-Thread können zu Komplikationen führen und müssen vermieden werden. Es bestehen generell mehrere Möglichkeiten, die kritischen UI-Updates

nur über den UI-Thread durchzuführen. Oft werden länger andauernde Aufgaben in separaten Threads erledigt und die Ergebnisse an eine spezielle Handler-Klasse übergeben, um die Kontrolle rechtzeitig an den UI-Thread zurückzugegeben. Jedoch müssen bei dieser Lösung die Threads und Handler immer explizit erstellt werden.

Stattdessen empfiehlt es sich, die Klasse AsyncTask zu verwenden, die neben einem komfortablen Thread-Management auch auf einfache Art GUI-Updates zur Fortschrittsanzeige erlaubt. Für eine Erweiterung der Klasse AsyncTask werden zunächst drei generische Typen bestimmt: Der Typ der Parameter, die zur Ausführung übergeben werden, der Typ der Fortschrittseinheiten – z.B. Integer, um einen Fortschrittsbalken zu aktualisieren – sowie der Typ des Ergebnisses, das zurückgeliefert wird. Mit diesen Typangaben können dann die Methoden eines AsyncTask implementiert werden. Für viele Zwecke genügt es lediglich `doInBackground(..)` und `onPostExecute(..)` zu überschreiben:

1. `doInBackground(..)` – Beinhaltet den Code für die lang andauernden Operationen, die in einem separaten worker-Thread erledigt werden. Es kann hier auf die Parameter zugegriffen werden, die zur Ausführung übergeben wurden.
2. `onPostExecute(..)`– Nimmt nach der Beendigung von `doInBackground()` das Ergebnis entgegen und liefert es an den Aufrufer zurück.
3. `onPreExecute(..)`– Wird vor dem Aufruf von `doInBackground()` zur Vorbereitung aufgerufen.
4. `onProgressUpdate(..)`– Für Aktualisierungen, die bei jedem Aufruf von `publishProgress(..)` innerhalb `doInBackground(..)` geschehen.

Außer `doInBackground(..)` werden alle Methoden im UI-Thread ausgeführt. Der Entwickler muss sich um die explizite Implementierung zusätzlicher worker-Threads keine Gedanken machen. Ein Nebeneffekt ist zudem, dass auch der Code leserlicher wird.

3 Analyse

Um die Anforderungen für den Studienplaner formulieren zu können, werden zu Beginn dieses Kapitels zunächst einige Nachteile vorhandener Lösungen zur Semesterplanung herausgestellt. Im Anschluss folgt eine ausführliche Beschreibung aller funktionalen, nichtfunktionalen und technischen Anforderungen.

3.1 Vorhandene Lösungen

Um einen Veranstaltungsplan zusammenzustellen und in ein druckbares Format zu bringen, hat ein Student der HAW Hamburg eine praktische Anwendung entwickelt (Abbildung 3.1). Sie ermöglicht es, die von der Hochschule veröffentlichten Semesterpläne in Form von Textdateien einzulesen, um daraus einen individuellen Plan zusammenzustellen. Der fertige Plan kann dann gedruckt oder in das iCalendar-Format (2.2.2) exportiert werden.

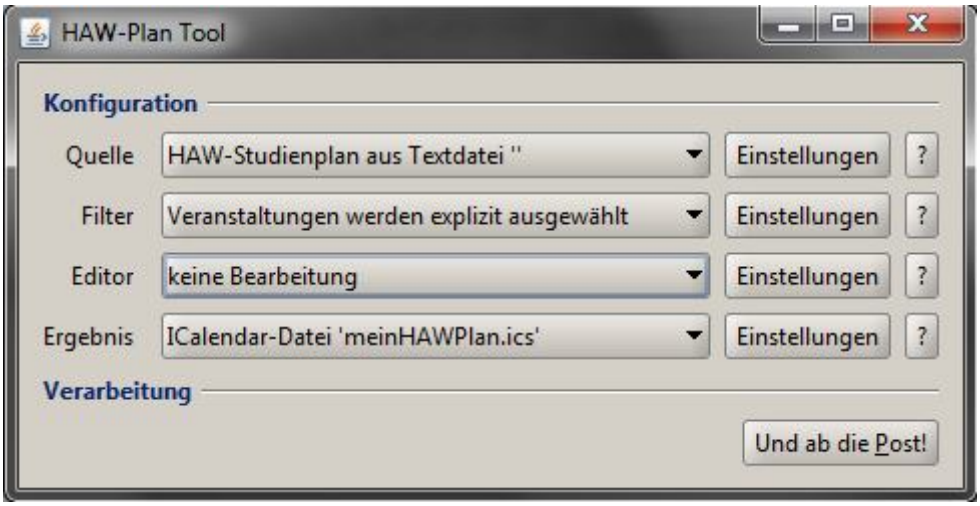

Abbildung 3.1: Screenshot des „HAW-Plan Tool“

Dieses Tool ist leider nur eine Nischenlösung, die ein spezielles Eingabeformat für die Veranstaltungsdaten erfordert. Ein einheitliches Datenformat ist nicht gegeben (vgl. 2.1.2), darum muss ein für jeden Studenten nützliches System zur Semesterplanung die Möglichkeit bieten, die Veranstaltungsdaten direkt einzugeben.

Oft wird von Studenten auch einfach auf Kalenderanwendungen zurückgegriffen, denen weitgehend der fachliche Bezug fehlt, wie zum Beispiel den Kalender in Microsoft Outlook, den Google- oder Android-Kalender. Die Erfassung studienbezogener Veranstaltungsdaten wie beispielsweise Dozenten und Semester muss für eine strukturiertere Studienplanung jedoch explizit möglich sein. Weitere Aspekte, die die Einsetzbarkeit dieser Kalenderanwendungen zur Studienplanung zusätzlich einschränken, wurden bereits im Zusammenhang mit dem iCalendar-Standard in Abschnitt 2.2.2 erwähnt. Um die hier unberücksichtigten Ausnahmen zu Serienterminen auf sehr einfache und übersichtliche Weise definieren zu können, bietet sich die Kalenderwochenplanung (vgl. 2.2.1) an.

Von vielen Studenten wird der Semesterplan sogar auch noch in Papierform geführt, was natürlich in vielerlei Hinsicht unpraktisch ist. Im Allgemeinen ist es aufwendig, einen vollständigen Semesterplan auf dem Papier zu erstellen oder zu diesem Zweck einen Taschenkalender zu pflegen. Oft hat man Papierunterlagen nicht zur Hand, wenn sie gerade benötigt werden. Die Papierform birgt ein gewisses Verlustrisiko, und durch nachträgliche Korrekturen kann ein handgeschriebener Plan mit der Zeit leicht unleserlich werden. Hier liegt der Nutzen eines Smartphone-Planers als Alternative also auch auf der Hand.

3.2 Grundsätzliche Anforderungen

Der Studienplaner hat grundsätzlich die Aufgabe Studenten bei ihrer Organisation zu unterstützen. Er soll ihnen dabei helfen, eine individuelle Semesterplanung nach Kalenderwochen aufzustellen. Hierfür erforderlich sind eine einfache Erfassung, dauerhafte Speicherung und übersichtliche Darstellung von studienbezogenen Veranstaltungsdaten. Ergänzungen oder Nachbesserungen an einem erstellten Semesterplan sollen jederzeit vorgenommen werden können. Mit einer Funktion, die das Smartphone während der eingetragenen Veranstaltungszeiten automatisch in den Lautlosmodus versetzt, soll darüber hinaus ein zusätzlicher, praktischer Nutzen gegeben sein.

3.3 Funktionale Anforderungen

3.3.1 Hauptbildschirm

R1

Beim Start des Studienplaners wird ein Auswahlmenü angezeigt. Aus diesem Menü lassen sich die einzelnen Komponenten der Anwendung, wie der Veranstaltungsplan und eventuelle, spätere Erweiterungen, starten.

3.3.2 Veranstaltungsplan

R2

Der Plan wird in einer Wochenübersicht für das eingestellte Semester gezeigt, mit Tabulatoren für die Wochentage von Montag bis Freitag und optional Samstag. Sonntage können bei der Planung von Hochschulterminen als allgemeine Ruhetage vernachlässigt werden. Für die einzelnen Wochentage befinden sich unter der Tabulatorleiste die zugehörigen Tagespläne. Über den entsprechenden Tabulator kann jeder Tagesplan zur Anzeige ausgewählt werden. So ist eine übersichtliche Darstellung des Gesamtplans auch auf kleinen Smartphone-Bildschirmen gewährleistet. Zudem können auch Pläne für bereits vergangene Semester angezeigt werden.

R3

Für den in der Wochenübersicht ausgewählten Tag werden alle Veranstaltungen in einem Tagesplan dargestellt, der der allgemein üblichen Form eines tabellarischen Stundenplans entspricht; in der linken Spalte stehen die Uhrzeiten und rechts die Veranstaltungen. Dieser Stundenplan ist senkrecht scrollbar, um eine Mindesthöhe und somit die Lesbarkeit sicherzustellen.

R4

Ein Tagesplan reagiert auf Long-Touch-Events zum Öffnen eines Kontextmenüs, das eine Option zum Eintragen einer neuen Veranstaltung anbietet. Über die Option wird das entsprechende Eingabeformular geöffnet und mit dem Wochentag des Tagesplans, sowie der Uhrzeit initialisiert, für die das Touch-Event registriert wurde. Auf diese Art ist eine schnelle und komfortable Eingabe neuer Veranstaltungsdaten möglich.

R5

Beim Start wird der Stundenplan für den aktuellen Wochentag angezeigt, um dem Anwender stets Aktualität zu vermitteln und ihm den Schritt in der Navigation zu ersparen jeweils noch den aktuellen Plan extra auszuwählen.

R6

Die Veranstaltungen sind durch einfache Rechtecke im Stundenplan dargestellt. Diese enthalten die folgenden Daten zur Veranstaltung: Name, Beginn, Ende, Kalenderwochen, Raum, Dozent. Die Höhe der Rechtecke und die Position im Stundenplan sind jeweils durch Veranstaltungsbeginn und -ende festgelegt. In der Breite wird der vorhandene Bildschirmplatz ausgenutzt, damit für die Schrift, in der Veranstaltungsdaten dargestellt werden, eine Größe von mindestens 10dp gewählt werden kann. Zeitgleich stattfindende Veranstaltungen werden im Tagesplan nebeneinander platziert.

R7

Die Veranstaltungen können direkt aus dem Stundenplan heraus bearbeitet oder gelöscht werden. Die Rechtecke reagieren auf Long-Touch-Events, um ein Kontextmenü hervorzurufen. Mit diesem Menü wird das Bearbeiten oder Löschen der jeweiligen Veranstaltung in einer einfachen Interaktion möglich. Zum Bearbeiten einer Veranstaltung wird das Formular mit den entsprechenden Veranstaltungsdaten gezeigt, zum Löschen ein Auswahldialog, in dem die Aktion zur Sicherheit noch einmal bestätigt werden muss.

R8

Das eingestellte Semester und die aktuelle Kalenderwoche mit ihren Datumsgrenzen werden über dem Plan angezeigt. Dies erleichtert die Orientierung.

3.3.3 Formular zur Erfassung von Veranstaltungsdaten

R9

Über ein Formular können neue Veranstaltungen im Plan eingetragen werden. Es ist über eine Option im Optionsmenü erreichbar, sowie über ein Kontextmenü im Veranstaltungsplan. Dies entspricht einer üblichen Lösung zur Gewährleistung intuitiver Bedienbarkeit. Je nachdem, ob das Formular zum Bearbeiten einer vorhandenen oder Erstellen einer neuen Veranstaltung

gestartet wurde, werden die Formularelemente entweder mit den zu ändernden Veranstaltungsdaten, oder – im zweiten Fall – mit dem Semester und Wochentag des Tagesplans, aus dem es aufgerufen wurde, vorbelegt. Wurde es per Kontextmenü aus dem Tagesplan aufgerufen, wird zudem die ausgewählte Uhrzeit übernommen (siehe R4). Dies erspart dem Benutzer unnötige Eingaben.

R10

Das Formular enthält alle Eingabefelder, die zur Erfassung der zugrunde liegenden Veranstaltungsdaten in einem lesbaren und verständlichen Format notwendig sind. Die Angaben zu Raum und Dozent sind optional, alle übrigen sind zur Festlegung eines Veranstaltungstermins zwingend erforderlich:

- Name (Pflichtangabe, mindestens 2, maximal 30 alphanumerischen Zeichen)
- Wochentag (Pflichtangabe, Montag bis Samstag auswählbar)
- Beginn (Pflichtangabe, minutengenaue Uhrzeit im 24h-Format: HH:MM)
- Ende (Pflichtangabe, minutengenaue Uhrzeit im 24h-Format: HH:MM)
- Kalenderwochen (Pflichtangabe, siehe R11-R13)
- Raum (optionale Angabe, maximal 20 alphanumerischen Zeichen)
- Dozent (optionale Angabe, maximal 20 alphanumerischen Zeichen)

R11

Die Erfassung der Kalenderwochen erfolgt per Eingabe in einem Textfeld und zusätzlich per Auswahl in einem Checkbox-Dialog. Dieser Dialog enthält neben den Wochennummern die entsprechenden Datumsgrenzen. Wird er geöffnet, sind dort bereits alle im Textfeld enthaltenen, zulässigen Kalenderwochen als ausgewählt markiert. Umgekehrt wird das Textfeld mit der Auswahl im Checkbox-Dialog aktualisiert, sobald der Dialog bestätigt wurde. Auf diese Art wird dem Anwender zum einen die Möglichkeit gegeben, ganze Bereiche von Wochennummern schnell im Textfeld einzugeben, und zum anderen für die so definierten Terminserien im Dialog Ausnahmen anzugeben, wobei das konkrete Datum des Einzeltermins jeweils anhand der Wochen-Datumsgrenzen geprüft werden kann.

R12

Der zur Planung verfügbare Kalenderwochenbereich ist durch das eingestellte Semester festgelegt. Für Sommersemester umfasst dieser die Wochennummern 10 bis 50 und für Wintersemester die Woche 30 des Semesterstartjahres bis 20 des Folgejahres. Eine solche

Festlegung ist erforderlich, um die Eingabe mehrerer Wochennummern in einem Textfeld zu ermöglichen. So muss nicht zu jeder Woche noch das Jahr eingegeben werden, da es durch das voreingestellte Plansemester schon festgelegt ist. Es ist beispielsweise eindeutig definiert, dass in einem Wintersemester die zweite Kalenderwoche nie für das Semesterstartjahr gilt. Zugleich werden die unterschiedlichen Semesterzeiten deutscher Hochschulen durch diese Bereiche ausreichend berücksichtigt.

R13

Das Format, in dem Kalenderwochen im Textfeld angegeben werden können, ist wie folgt definiert: Wochennummern grösser 53 sind grundsätzlich zulässig und werden für das eingestellte Semester normalisiert. Nur falls dabei der Kalenderwochenbereich für das Plansemester (siehe R12) verlassen wird, ist die Eingabe ungültig. Die einzelnen Nummern werden durch Kommata getrennt, dabei können zusammenhängende Bereiche von mindestens drei Wochen jeweils mit einem Bindestrich zwischen erster und letzter Kalenderwoche des Bereichs zusammengefasst werden. Leerzeichen sind beliebig erlaubt. So ist das Eingabeformat für den Anwender leicht verständlich und umsetzbar. Zur Eingabehilfe wird das Format neben dem Textfeld im Formular beispielhaft dargestellt.

R14

Es besteht die Möglichkeit, die Eingaben in der Datenbank zu speichern oder sie zu verwerfen. Hierfür befinden sich zwei Schaltflächen zum Speichern und Abbrechen am unteren Ende des Formulars. Da das Formular nach erfolgreichem Speichern oder Abbruch nicht mehr benötigt wird, aktualisiert die Anwendung danach die Bildschirmseite, über die der Anwender es geladen hatte.

R15

Vor dem Speichern erfolgt ein Korrektheitstest, um inhaltlichen Mängeln vorzubeugen, die die Verständlichkeit, Lesbarkeit oder Konsistenz der eingegebenen Daten beeinträchtigen. Schlägt der Test fehl, wird dem Anwender eine detaillierte Hinweismeldung angezeigt, die eine Nachkorrektur der Angaben erleichtert. Es werden die folgenden Kriterien geprüft:

- der Name einer Veranstaltung hat mindestens 2 Zeichen
- die Mindestdauer einer Veranstaltung beträgt 15 Minuten
- die Kalenderwochen sind für das eingestellte Semester zulässig

Nach dem Speichervorgang wird eine Erfolgsmeldung angezeigt, oder eine Fehlermeldung, falls ein Datenbankfehler auftritt.

3.3.4 Persistenz

R16

Alle zur Spezifizierung von Hochschulveranstaltungen erforderlichen Daten werden dauerhaft in der Datenbank gespeichert:

- Semester
- Name
- Wochentag
- Beginn
- Ende
- Kalenderwochen
- Raum
- Dozent

3.3.5 Einstellungen

R17

Die Einstellungen der Anwendung können auf dem Wege angezeigt und geändert werden, wie der Anwender es üblicherweise erwartet – über einen Menüpunkt im Optionsmenü. Dieses befindet sich sowohl auf der Startseite, als auch auf der Wochenübersicht des Veranstaltungsplans.

R18

Das Semester, für das der Veranstaltungsplan gilt, kann eingestellt werden. Für dieses Semester werden alle Veranstaltungen geladen und gespeichert. Eine Durchmischung von Terminen für unterschiedliche Semester wird damit verhindert. Unmittelbar vor der Umschaltung des Semesters wird automatisch sichergestellt, dass die Lautlosfunktion deaktiviert ist, und dass vorgemerkte Lautlostermine abgemeldet sind, damit das Gerät später nicht unerwartet den Klingelmodus ändert.

R19

Die Beginn- und Ende-Zeit für die Anzeige der Stundenpläne kann angegeben werden, um den verfügbaren Bildschirmplatz des Anwenders optimal zu nutzen.

R20

Die zeitliche Auflösung der Stundenpläne ist zwischen 5, 15, 30 und 60 Minuten variierbar. Diese Einstellung verursacht keine Änderung von Veranstaltungsdaten, sie dient ausschließlich der Übersichtlichkeit der Stundenplan-Anzeige; auf größeren Bildschirmen kann eine höhere Auflösung gewählt werden.

R21

Da der Samstag nicht immer in der Planung berücksichtigt werden muss, ist dieser in der Wochenübersicht lediglich optional sichtbar. Dies spart Bildschirmplatz und erhöht die Übersichtlichkeit.

R22

Die Lautlosfunktion kann je nach Bedarf aktiviert oder deaktiviert werden.

3.3.6 Lautlosfunktion

R23

Die Lautlosfunktion ist eine Funktion, die das Gerät zum Beginn einer Veranstaltung in den Lautlosmodus versetzt und zum Veranstaltungsende zurück in den normalen Klingelmodus. Sie wird über die Einstellungen umgeschaltet, damit sie für den Anwender jederzeit schnell zugänglich ist.

R24

Beim Starten der Anwendung wird überprüft, ob die Funktion bereits eingeschaltet ist. Falls dies der Fall ist, wird der Lautlosmodus für die nächste Veranstaltung sofort vorgemerkt. So ist sichergestellt, dass die Anwendung auch dann noch auf die zuletzt gespeicherte Einstellung des Anwenders reagiert, wenn sie schon längere Zeit nicht mehr verwendet wurde.

R25

Bei Aktivierung der Funktion wird der Lautlosmodus für die nächste Veranstaltung ebenfalls sofort vorgemerkt. Für das Vormerken wird der Beginn der nächsten Veranstaltung gewählt, die in der aktuellen oder einer der beiden darauffolgenden Kalenderwochen des Semesters ansteht. Läuft zu dem aktuellen Zeitpunkt bereits eine Veranstaltung, schaltet es sich sofort lautlos. Dadurch kann der Anwender die Funktion sowohl vorausplanend, als auch spontan nutzen. Einen Termin vorzumerken, der noch weit in der Zukunft liegt, ist nicht notwendig.

R26

Bei Deaktivierung der Funktion verlässt das Smartphone den Lautlosmodus und schaltet in den Normalmodus. Ein gegebenenfalls im System vorgemerkter Termin zur Lautlosmodus-Aktivierung wird entfernt. Der Anwender kann die Funktion also jederzeit ausschalten, falls er an einer Veranstaltung beispielsweise nicht mehr teilnimmt.

R27

Der Lautlosmodus wird auch dann zum nächsten Termin aktiviert/deaktiviert, wenn die Studienplaner-Anwendung nicht läuft. Somit verhindert auch das Schließen der Anwendung nicht, dass die Lautlosfunktion korrekt arbeitet.

3.4 Nichtfunktionale Anforderungen

R28

Die Benutzbarkeit sämtlicher Bildschirmoberflächen ist sowohl im Portrait-, als auch im Landscape-Modus für die spezifizierten Auflösungen gegeben. Dies macht insbesondere die Stundenpläne beim Drehen der Geräte zusätzlich übersichtlicher.

R29

Die Anwendung ist um eine Importfunktion für iCalendar-Dateien erweiterbar, um beispielsweise Outlook-Usern eine Möglichkeit zu geben, ihre Termine auch wie gewohnt am PC zu erstellen.

3.5 Technische Anforderungen

R30

Der Studienplaner ist kompatibel mit Android ab v1.6, da bis zu dieser Version derzeit noch viele Geräte in Verwendung sind. Die Kompatibilität ist bis zur Version 2.3 gewährleistet.

R31

Die unterstützten Geräte-Auflösungen sind:

- QVGA (240x320, low density, small screen)
- HVGA (320x480, medium density, normal screen)
- WVGA800 (480x800, high density, normal screen)
- WVGA854 (480x854 high density, normal screen)
- WQVGA400 (240x400, low density, normal screen)
- WQVGA432 (240x432, low density, normal screen)

Damit werden sowohl die kleinen, als auch die mittleren und großen Smartphone-Bildschirme unterstützt.

3.6 Ausgrenzungen

R32

Auf einigen Geräten werden Softkeyboards verwendet, deren Kompatibilität mit der Anwendung nicht gewährleistet ist. In dem Fall kann das Gerät vor dem Start der Anwendung nur möglicherweise auf ein anderes, kompatibles Keyboard eingestellt werden.

R33

Die Kompatibilität zu Android-Versionen nach v2.36 („Gingerbread“) ist nicht gewährleistet.

R34

Bei zwei oder mehr nebeneinander im Tagesplan positionierten Veranstaltungen werden möglicherweise nicht mehr alle Veranstaltungsdaten im Plan angezeigt.

R35

Die zur Nutzung der Lautlosfunktion eingestellten Alarme werden nur bis zu ihrer Ausführung oder Löschung vorgemerkt, da sie verloren gehen, falls das Gerät komplett

ausgeschaltet wird. In dem Fall wird die Funktion nur durch einen Neustart der Anwendung wieder nutzbar.

3.7 Use Cases

Im Folgenden werden die einzelnen Anwendungsfälle, die sich aus den funktionalen Anforderungen ableiten lassen, tabellarisch beschrieben. Die systemseitig unterstützten Vorgänge sind in der Beschreibung jeweils hervorgehoben. Einen Überblick über alle erfassten Use Cases geben die beiden Diagramme in Abbildung 3.2 und Abbildung 3.3.

3.7.1 Kernfunktionen

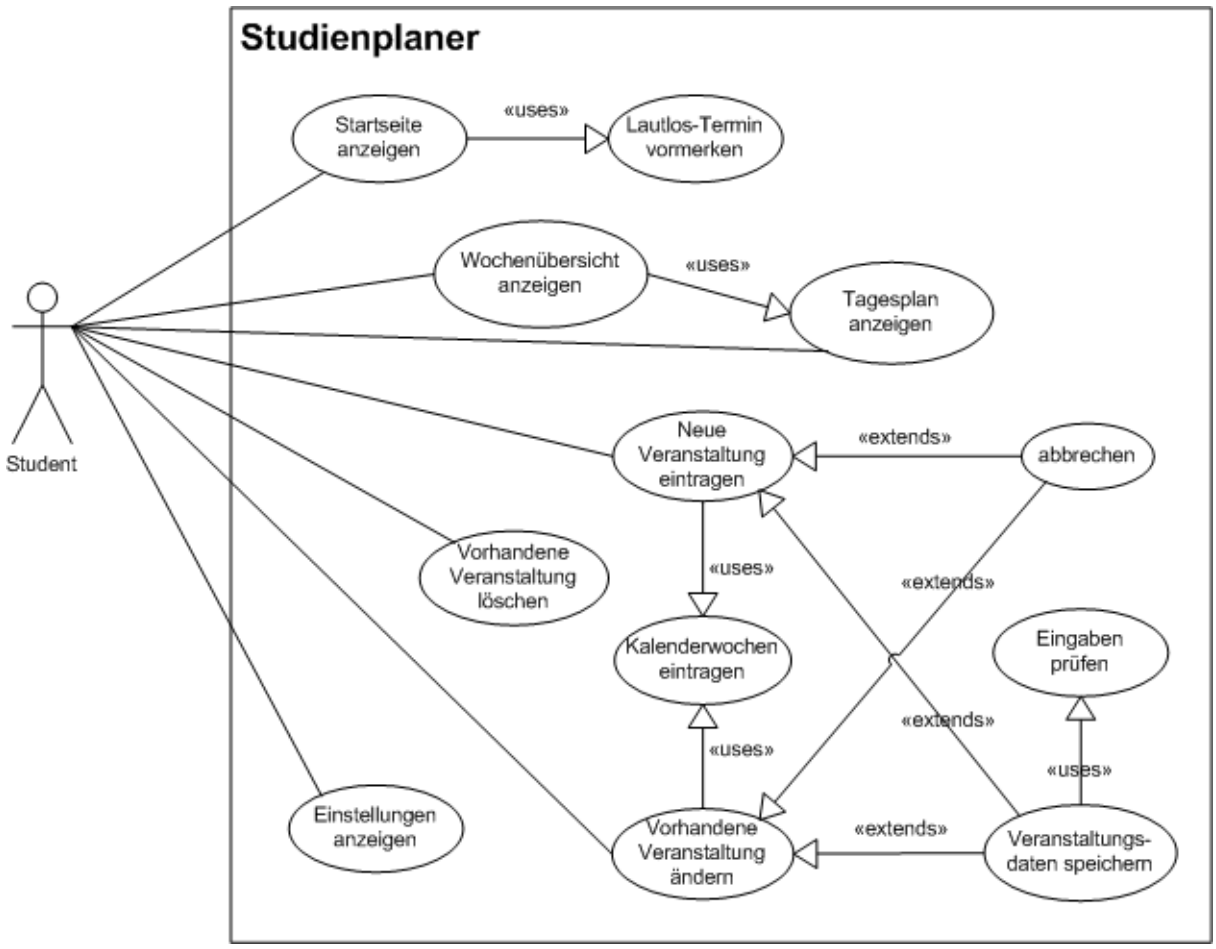

Abbildung 3.2: Anwendungsfälle des Studienplaner-Systems

Anwendungsfall	**Startseite anzeigen**
Akteur	Student
Ziel	Anzeigen einer Bildschirmseite mit dem Hauptmenü der Anwendung
Häufigkeit	Mehrmals täglich bis mehrmals pro Semester
Beschreibung	1. Der Benutzer startet die Anwendung. ***2. Das System zeigt die Bildschirmseite mit dem Hauptmenü an [R1] und merkt den nächsten Termin zum Wechseln in den Lautlosmodus vor, falls die Lautlosfunktion aktiviert ist [R25]. Dies erfolgt nach dem Anwendungsfall „Lautlostermin vormerken“.***

Tabelle 3.1: Anwendungsfall „Startseite anzeigen“

Anwendungsfall	**Einstellungen anzeigen**
Akteur	Student
Ziel	Anzeigen einer Bildschirmseite mit den Einstellungen zur Anwendung
Häufigkeit	Mehrmals pro Semester
Beschreibung	1. Der Benutzer wählt auf der Startseite oder der Wochenübersicht des Veranstaltungsplans über das Optionsmenü den Menüpunkt „Einstellungen“. ***2. Das System zeigt die Bildschirmseite mit den Einstellungen an [R17].***

Tabelle 3.2: Anwendungsfall „Einstellungen anzeigen“

Anwendungsfall	**Wochenansicht anzeigen**
Akteur	Student
Ziel	Anzeige einer Wochenübersicht mit dem Veranstaltungsplan
Häufigkeit	Mehrmals pro Semester bis mehrmals täglich
Vorbedingungen	Der Benutzer hat die Bildschirmseite mit dem Hauptmenü aufgerufen
Nachbedingungen	Das System zeigt die Veranstaltungsplan-Wochenübersicht an
Beschreibung	1. Der Benutzer ruft über einen Button aus dem Hauptmenü den Veranstaltungsplan auf. ***2. Das System startet die Bildschirmseite mit der Wochenübersicht zur Anzeige der Wochentag-Tabs für die scrollbaren Tagespläne [R2+R3].*** ***3. Das System zeigt in der Titelleiste der Wochenansicht das Plan-Semester und die aktuelle Kalenderwoche an [R8].*** ***4. Das System lädt die Veranstaltungsdaten entsprechend der Plan-Einstellung für Semester und Anzahl der Wochentage aus der Datenbank, erstellt daraus die Tagespläne und registriert alle Tagesplan-Zeilen und Veranstaltungen für das Kontextmenü in der Wochenansicht [R4+R7].*** ***6. Das System selektiert den Tab für den aktuellen Wochentag [R5].***

Tabelle 3.3: Anwendungsfall „Wochenübersicht anzeigen“

Anwendungsfall	**Tagesplan anzeigen**
Akteur	Student (1b) / System (1a)
Ziel	Anzeige des Stundenplans für einen Wochentag
Häufigkeit	Mehrmals pro Semester bis mehrmals täglich
Vorbedingungen	Der Benutzer hat die Bildschirmseite mit der Wochenübersicht aufgerufen
Nachbedingungen	Der Plan für den aktuellen (1a) oder vom Benutzer gewählten (1b) Wochentag wird angezeigt
Beschreibung	***1a. Das System lädt die Bildschirmseite mit der Wochenübersicht (siehe Anwendungsfall „Wochenübersicht anzeigen“).*** 1b. Der Benutzer wählt einen Wochentag aus dem Tab-Menü auf der Bildschirmseite mit der Wochenübersicht. ***2. Das System zeigt unter dem Tab-Menü der Wochenübersicht einen scrollbaren Stundenplan für den aktuellen (nach 1a) oder den vom Benutzer ausgewählten (nach 1b) Wochentag an [R2+R5].***

Tabelle 3.4: Anwendungsfall „Tagesplan anzeigen“

Anwendungsfall	**Neue Veranstaltung eintragen**
Akteur	Student
Ziel	Erfassen von Veranstaltungsdaten
Häufigkeit	Mehrmals zu Semesterbeginn
Nachbedingungen	Das System hat die vom Benutzer eingegebenen Veranstaltungsdaten in der Datenbank gespeichert
Erweiterungen	Der Benutzer klickt auf den Button „Abbrechen“ oder „Speichern“
Fehlerfälle	5a. Die Prüfung der vom Benutzer eingegebenen Daten schlägt fehl, das System zeigt einen Hinweis zur Korrektur der Eingaben an. 5b. Das System kann die Veranstaltungsdaten nicht in der Datenbank speichern und zeigt eine entsprechende Fehlermeldung an.
Beschreibung	1. Der Benutzer wählt in der Wochenübersicht über das Optionsmenü oder über das Kontextmenü eines Tagesplans den Menüpunkt zum Eintragen einer neuen Veranstaltung. ***2. Das System zeigt das Formular zur Datenerfassung an [R9] und initialisiert es mit dem Wochentag des Tagesplans. Sofern per Kontextmenü eine Uhrzeit im Plan ausgewählt wurde, übernimmt das System diese als Startzeit [R4].*** 3. Der Benutzer trägt die Veranstaltungsdaten in die entsprechenden Formularfelder ein. Die Eingabe der Kalenderwochen erfolgt dabei nach dem Anwendungsfall „Kalenderwochen eintragen“. 4a. Der Benutzer klickt auf den Button „Abbrechen“. Weiter bei 7. 4b. Der Benutzer klickt auf den Button „Speichern“. ***5. Das System prüft die Eingaben auf Korrektheit [Kriterien: R15].*** ***6. Das System speichert die Eingaben in der Datenbank [R14].*** ***7. Das System beendet die Bildschirmseite zur Datenerfassung und lädt die Bildschirmseite mit der Wochenübersicht neu [R14].***

Tabelle 3.5: Anwendungsfall „Neue Veranstaltung eintragen“

Anwendungsfall	**Vorhandene Veranstaltung ändern**
Akteur	Student
Ziel	Ändern der Daten zu einer in der Datenbank gespeicherten Veranstaltung
Häufigkeit	Mehrmals pro Semester
Vorbedingungen	Es ist mindestens eine Veranstaltung im Veranstaltungsplan eingetragen
Nachbedingungen	Die Daten zu einer Veranstaltung wurden in der Datenbank geändert
Erweiterungen	Der Benutzer klickt auf den Button „Abbrechen“ oder „Speichern“
Fehlerfälle	5a. Die Prüfung der vom Benutzer eingegebenen Daten schlägt fehl, das System zeigt einen Hinweis zur Korrektur der Eingaben an. Weiter bei 3. 5b. Das System kann die Veranstaltungsdaten nicht in der Datenbank speichern und zeigt eine entsprechende Fehlermeldung an. Weiter bei 3.
Beschreibung	1. Der Benutzer wählt in der Wochenübersicht über das Kontextmenü einer Veranstaltung den Menüpunkt „Veranstaltung ändern“. ***2. Das System zeigt das Formular zur Datenerfassung an [R9+R10].*** 3. Der Benutzer trägt die Veranstaltungsdaten in die entsprechenden Formularfelder ein. Die Eingabe der Kalenderwochen erfolgt nach dem Anwendungsfall „Kalenderwochen eintragen“. 4a. Der Benutzer klickt auf den Button „Abbrechen“. Weiter bei 7. 4b. Der Benutzer klickt auf den Button „Speichern“. ***5. Das System prüft die Eingaben auf Korrektheit [Kriterien: R15].*** ***6. Das System speichert die Eingaben in der Datenbank [R14].*** ***7. Das System zeigt eine Erfolgs- oder Fehlermeldung an, beendet die Bildschirmseite zur Datenerfassung und lädt die Bildschirmseite mit der Wochenübersicht neu [R14+R15].***

Tabelle 3.6: Anwendungsfall „Vorhandene Veranstaltung ändern“

Anwendungsfall	**Kalenderwochen eintragen**
Akteur	Student
Ziel	Eintragen von Kalenderwochen in das Formular zur Erfassung von Veranstaltungsdaten
Häufigkeit	Einmal pro Neueintrag oder Änderung einer Veranstaltung
Vorbedingungen	Das System zeigt die Seite zur Erfassung der Veranstaltungsdaten an
Nachbedingungen	Die Daten zu den ausgewählten Kalenderwochen stehen im entsprechenden Textfeld des Eingabeformulars
Erweiterungen	Der Benutzer klickt auf den Button „Abbrechen“ oder „Bestätigen“
Beschreibung	1a. Der Benutzer gibt per (Soft-)Keyboard die Kalenderwochen in geforderter Form in das dafür vorgesehene Textfeld ein. Ende des Anwendungsfalls. 1b. Der Benutzer klickt auf einen Button neben dem Textfeld zum Öffnen eines Kalenderwochen-Auswahldialoges. ***2. Das System analysiert die im Textfeld eingegebenen Zeichen auf zulässige Kalenderwochen [R12] und zeigt einen Checkbox-Dialog an, in dem alle für das Semester zulässigen Kalenderwochen aufgelistet sind. Die im Textfeld eingegebenen Kalenderwochen markiert das System als ausgewählt, alle übrigen als nicht ausgewählt [R11].*** 3. Der Benutzer selektiert beliebige Kalenderwochen-Checkboxen 4a. Der Benutzer klickt im Dialog auf den Button „Abbrechen“. Ende des Anwendungsfalls. 4b. Der Benutzer klickt im Auswahldialog auf den Button „OK“. ***5. Das System erstellt aus allen im Checkbox-Dialog als ausgewählt markierten Wochen eine Zeichenkette für die Kurzschreibweise [R13] und setzt diese als neuen Inhalt des Kalenderwochen-Textfeldes [R11].***

Tabelle 3.7: Anwendungsfall „Kalenderwochen eintragen“

Anwendungsfall	**Vorhandene Veranstaltung löschen**
Akteur	Student
Ziel	Löschen einer Veranstaltung aus der Datenbank
Häufigkeit	Mehrmals pro Semester
Vorbedingungen	Das System zeigt eine Wochenübersicht mit einem Veranstaltungsplan an, in dem mindestens eine Veranstaltung eingetragen ist
Nachbedingungen	Das System hat die Daten zu der vom Benutzer ausgewählten Veranstaltung aus der Datenbank gelöscht
Erweiterungen	Der Benutzer klickt auf den Button „Abbrechen“ oder „Bestätigen“
Fehlerfälle	4b. Das System kann die Daten zur gewählten Veranstaltung nicht aus der Datenbank löschen. 5. Das System zeigt eine Meldung mit einer Fehlerbeschreibung an.
Beschreibung	1. Der Benutzer wählt in der Wochenübersicht über das Kontextmenü einer Veranstaltung den Menüpunkt „Veranstaltung löschen“. ***2. Das System zeigt einen Auswahldialog zum endgültigen Bestätigen oder Abbrechen des Vorgangs an [R7].*** 3a. Der Benutzer klickt im Dialog auf den Button „OK“. Weiter bei 4a. 3b. Der Benutzer klickt im Dialog auf den Button „Abbrechen“. Ende des Anwendungsfalls. ***4a. Das System löscht die Daten zur ausgewählten Veranstaltung aus der Datenbank [R7].***

Tabelle 3.8: Anwendungsfall „Vorhandene Veranstaltung löschen“

Anwendungsfall	**Lautlostermin vormerken**
Akteur	Student (1b) / System (1a, 1c, 1d)
Ziel	Den nächsten Termin zum De-/Aktivieren des Lautlosmodus vormerken
Häufigkeit	Mehrmals täglich bis mehrmals pro Semester
Beschreibung	***1a. Das System lädt die Startseite.*** 1b. Der Benutzer aktiviert die Lautlosfunktion in den Einstellungen. ***1c. Das System hat das Gerät in den Lautlosmodus geschaltet.*** ***1d. Das System hat das Gerät in den normalen Klingelmodus geschaltet.*** ***2. Das System merkt den Termin für den nächsten Veranstaltungsbeginn (nach 1a, 1b und 1d) vor, der in der aktuellen oder einer der beiden folgenden Kalenderwochen des Semesters ansteht. Läuft aktuell eine Veranstaltung, schaltet das System das Gerät sofort lautlos. Wurde der Lautlosmodus gerade aktiviert (1c), merkt das System sofort den Termin für das nächste Veranstaltungsende zum Zurückschalten des Gerätes in den normalen Klingelmodus vor. [R24-R26]***

Tabelle 3.9: Anwendungsfall „Lautlostermin vormerken“

Anwendungsfall	**Lautlostermin canceln**
Akteur	Student
Ziel	Einen vorgemerkten Termin zum De-/Aktivieren des Lautlosmodus canceln
Häufigkeit	Mehrmals täglich bis mehrmals pro Semester
Vorbedingungen	Ein Termin zur De-/Aktivierung des Lautlosmodus wurde vorgemerkt
Beschreibung	1. Der Benutzer deaktiviert die Lautlosfunktion in den Einstellungen. ***2. Das System cancelt den zur Aktivierung (Veranstaltungsbeginn) oder Deaktivierung (Veranstaltungsende) des Lautlosmodus vorgemerkten Termin [R27].***

Tabelle 3.10: Anwendungsfall „Lautlostermin canceln“

3.7.2 Einstellungen

Die Anwendungsfälle zum Bearbeiten der unterschiedlichen Einstellungen lassen sich zusammenfassend beschreiben. Der Anwendungsfall „Plan-Semester einstellen“ beinhaltet eine automatische Deaktivierung der Lautlosfunktion (siehe R18 und R23).

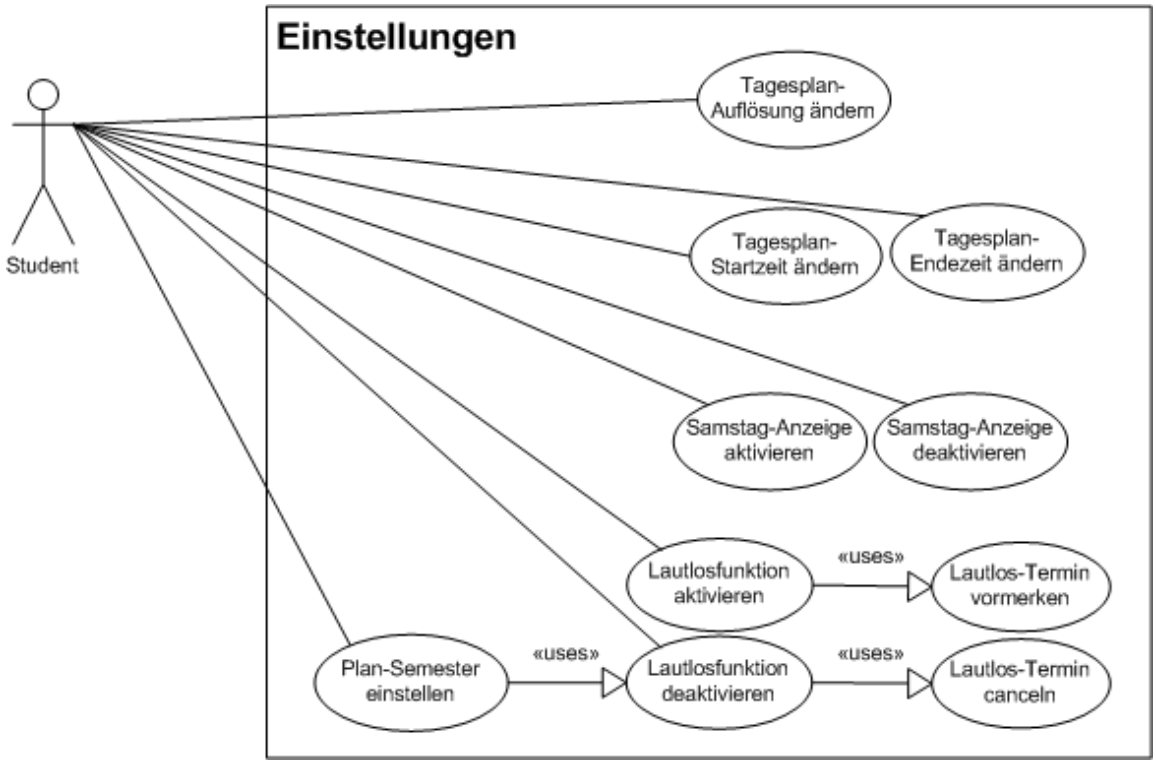

Abbildung 3.3: Auf die Einstellungen bezogene Anwendungsfälle des Systems

Eine De-/Aktivierung der Lautlosfunktion wird sofort – direkt nach der Änderung auf der Bildschirmseite mit den Einstellungen – vom System verarbeitet, um Verzögerungen auszuschließen, die der Benutzer nicht erwartet. Alle anderen Änderungen beziehen sich auf die Anzeige der Wochenübersicht und werden darum erst durch ein späteres, erneutes Laden der persistierten Anwendungseinstellungen wirksam.

Anwendungsfälle	**Plan-Semester einstellen, Tagesplan-Auflösung ändern, Tagesplan-Startzeit/-Endezeit ändern, Samstagsanzeige de-/aktivieren, Lautlosfunktion de-/aktivieren**
Akteur	Student
Ziel	Einstellungen ändern
Häufigkeit	Mehrmals pro Semester
Vorbedingungen	Das System zeigt die Bildschirmseite mit den Einstellungen an [R18-R23]
Nachbedingungen	Die Änderungen wurden in den Anwendungseinstellungen gespeichert
Beschreibung	1. Der Benutzer bearbeitet auf der Bildschirmseite Einstellungen die angezeigte Systemkonfiguration. Bei Änderung des Plan-Semesters weiter bei 2, bei Änderung der Lautlosfunktion weiter bei 3, sonst weiter bei 4. ***2. Das System deaktiviert die Lautlosfunktion und zeigt die Änderung an.*** ***3. Das System ändert den Lautlosmodus des Gerätes*** ***4. Das System persistiert alle Änderungen in den Anwendungseinstellungen.***

Tabelle 3.11: Anwendungsfälle zum Bearbeiten der Einstellungen

4 Konzeption

Dieses Kapitel beschreibt die notwendigen Konzepte zur Realisierung der Studienplaner-Anwendung. Zunächst wird die Gesamtarchitektur des Systems vorgestellt, mit der auch ein Blick auf unterschiedliche Erweiterungsmöglichkeiten gegeben werden soll. Danach folgt der Entwurf des Datenbankschemas, das dem Veranstaltungsplan zugrunde liegt. Darauf aufbauend werden dann die einzelnen Komponenten im System näher beschrieben.

4.1 Architekturentwurf

Für das Gesamtsystem können zunächst fünf Hauptkomponenten identifiziert werden (Abb. 4.1), wobei die Komponenten „Startmenü“, „Einstellungen“ und „Veranstaltungsplan“ für Interaktionen mit dem Anwender ausgelegt sind. Die „Startmenü“-Komponente repräsentiert hier lediglich die erste Bildschirmseite der Anwendung mit dem Hauptmenü, das zum Starten der Kernkomponenten wie dem Veranstaltungsplan dient. Sie wird allerdings zwecks zukünftiger Erweiterungen des Studienplaners als wichtiger Bestandteil des Systems bedacht.

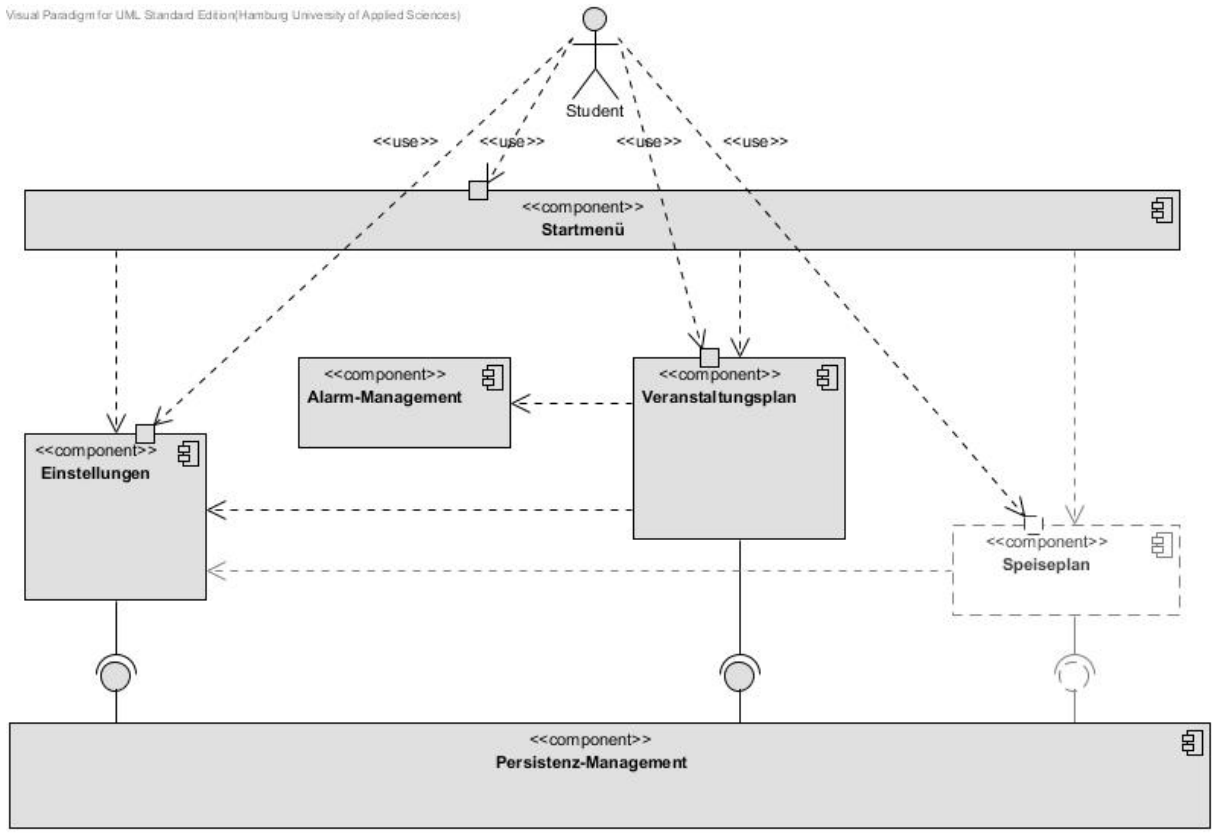

Abbildung 4.1: Gesamtarchitektur des Studienplaner-Systems

Die Komponente „Einstellungen“ stellt ein zentrales Element in der Architektur dar, denn sie bündelt die Zugriffe aller Komponenten auf die gemeinsamen Anwendungseinstellungen. Für jede neue Erweiterungskomponente, die eigene Einstellungen verwendet, wird sie entsprechend ausgebaut.
Gleiches gilt für das Persistenz-Management. Hier sind komponentenspezifische Schnittstellen zur Datenbank implementiert. Bei einer Erweiterung des Systems um eine zusätzliche Komponente, wie zum Beispiel einen Mensa-Speiseplan oder eine iCalendar-Importfunktion, wird das Persistenz-Management um die Implementierung der benötigten Schnittstelle ergänzt. Das Diagramm zeigt mit dem Speiseplan eine mögliche Erweiterung.

Die Kernfunktionalität der Anwendung wird von der Komponente „Veranstaltungsplan“ bereitgestellt. Sie verwendet die Persistenz-Schnittstelle zum Laden und Speichern von Veranstaltungsdaten und bietet Möglichkeiten zur Erfassung und Visualisierung dieser Daten. Mit der Lautlosfunktion beinhaltet sie daneben eine Funktionalität, die dem System durch das Alarm-Management zur Verfügung steht. Dieses ermöglicht das Einstellen und Empfangen beliebiger System-Alarme zu definierten Zeitpunkten, was auch im Hinblick auf zukünftige Erweiterungen ein zweckmäßiges Feature des Studienplaners darstellt. So kann beispielsweise die Alarm-Management Komponente auch dazu genutzt werden, um über System-Alarme an wichtige Termine im Veranstaltungsplan zu erinnern.

4.2 Datenbankschema

Zur Realisierung des Veranstaltungsplans stellt sich die Frage nach einem geeigneten Datenbankschema. Der Entwurf lässt sich im ersten Schritt durch die Konstruktion eines ER-Modells bewerkstelligen (Abbildung 4.2). Die zu speichernden Daten sind aus R16 des Anforderungskatalogs übernommen.

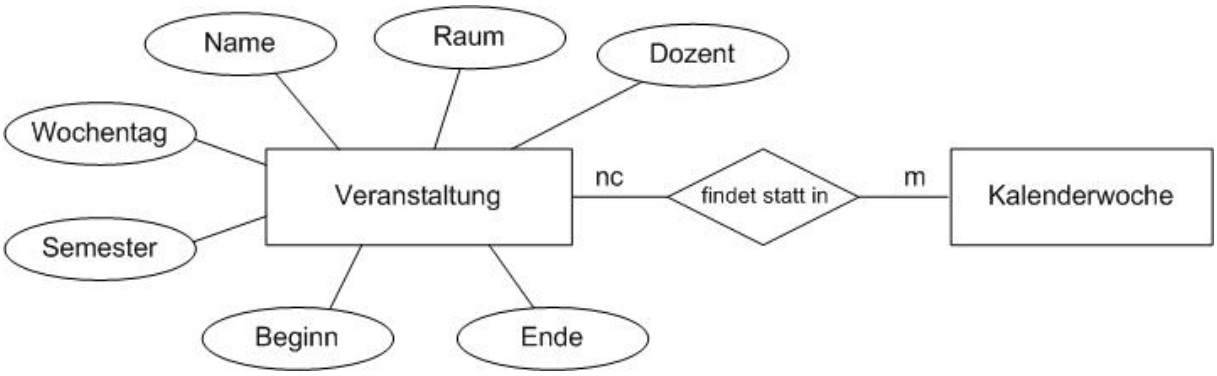

Abbildung 4.2: ER-Modell zum Entwurf des Datenbankschemas für den Veranstaltungsplan

Zwischen der Entität „Veranstaltung“ und den als deren Attribute dargestellten Elementen besteht jeweils eine n:1– beziehungsweise nc:1 – Beziehung; jeder Attributwert kann von mehreren Veranstaltungen angenommen werden, aber es ist für jede Veranstaltung nur genau ein Wert zulässig. Die Beziehung zwischen Veranstaltungen und Kalenderwochen ist dagegen nc:m, da eine Veranstaltung auch in mehreren Kalenderwochen stattfinden kann.

Das im nächsten Schritt entwickelte Datenbankmodell umfasst lediglich zwei Tabellen:

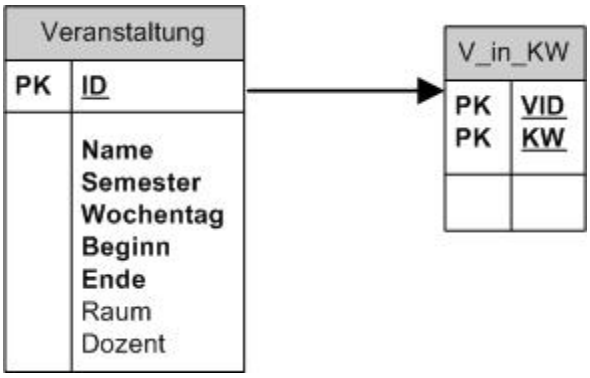

Abbildung 4.3: Relationales Datenbankmodell für den Veranstaltungsplan

Die Entität „Veranstaltung“ wird mit allen Attributen in der gleichnamigen Tabelle realisiert. Sie wird um eine eindeutige ID erweitert, die als Primärschlüssel dient und den gezielten Zugriff zum Ändern oder Löschen eines Datensatzes erleichtert. Die „findet statt in“ – Beziehungsentität erfordert aufgrund der Kardinalitäten eine separate Tabelle. Allerdings hat die Entität Kalenderwoche im ER-Modell keine Attribute, und es besteht kein Grund von vornherein alle möglichen Kalenderwochen in der Datenbank zu speichern. Daher genügt es, die Beziehung in einer Tabelle zu modellieren, deren Primärschlüssel sich zusammensetzt aus der Kalenderwoche und dem Fremdschlüssel Veranstaltungs-ID. Für jeden Datensatz der Tabelle „Veranstaltung“ ist dort mindestens ein Datensatz vorhanden.

4.2.1 Datentypen

Die Festlegung der verwendeten Datentypen und -längen erfolgt nach der Anforderung R10; Id und Kalenderwochen können jeweils als Integer gespeichert werden, ebenso der Wochentag (1=Montag bis 6=Samstag). Für alle übrigen Angaben werden Strings verwendet, um unnötige Konvertierungen zu vermeiden.

4.3 Entwurf des Veranstaltungsplans

Das nachfolgende Diagramm zeigt den Komponentenentwurf für den Veranstaltungsplan und stellt die Beziehungen zu den Hauptkomponenten des Studienplaners dar. Das Starten der Lautlosfunktion aus den Einstellungen wurde im Architekturentwurf in Abbildung 4.1 zunächst vernachlässigt.

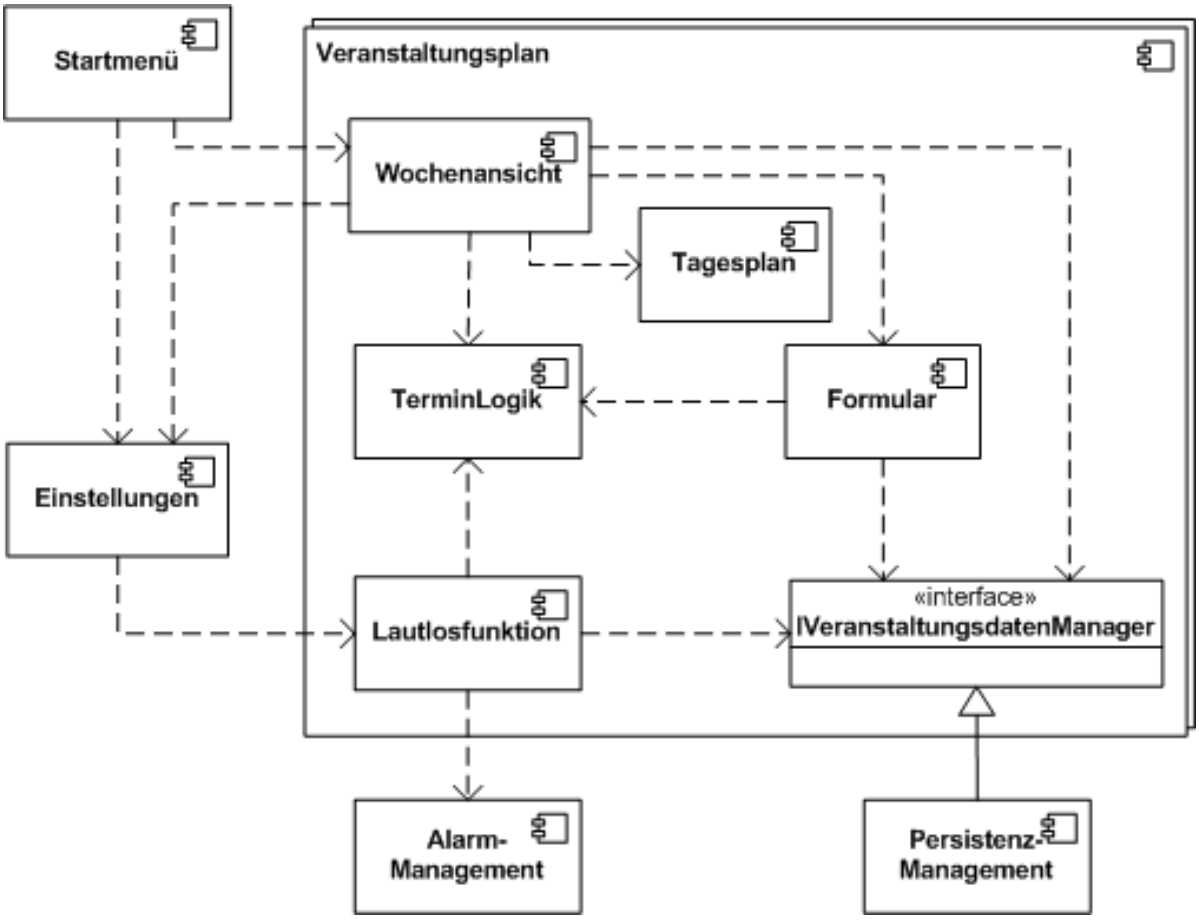

Abbildung 4.4: Einbettung der Veranstaltungsplan-Komponente in das Studienplaner-System

Aus dem Startmenü wird die Wochenansicht des Veranstaltungsplans aufgerufen. Diese zeigt den Semesterplan an, der aus mehreren Tagesplänen besteht. Dabei greift sie auf die Veranstaltungsplan-spezifischen Einstellungen zurück, wie zum Beispiel Angaben zum Semester und zur zeitlichen Auflösung des darzustellenden Plans.

Die Komponente „Einstellungen“ gewährt nicht nur den Zugriff auf die persistierten Benutzereinstellungen, sie stellt auch die Bildschirmseite zur Konfiguration der Anwendung

bereit, die über das Optionsmenü der Wochenansicht und der Startseite aufgerufen werden kann.

Das Formular zur Erfassung von Veranstaltungsdaten wird über das Kontext- oder Optionsmenü aus der Wochenansicht heraus gestartet. Es enthält alle notwendigen Formularelemente, um neue Veranstaltungen in den Plan einzutragen oder vorhandene Einträge zu ändern.

Die Komponente „Lautlosfunktion“ konkretisiert die Funktionen des AlarmManagement, das zur Verwaltung beliebiger Alarme mit Hilfe des Android-AlarmManagers dient, und sich aus zwei Teilen zusammensetzt: Zum einen beinhaltet es das An- und Abmelden von Terminen beim AlarmManager, zu denen ein Alarm-Intent versendet wird. Bei der Lautlosfunktion sind es Veranstaltungstermine, zu denen das Gerät in den Lautlosmodus schaltet. Der zweite Bestandteil des AlarmManagement ist ein Empfänger, der auf die vom Framework versendeten Alarme reagiert. Ein solcher Empfänger veranlasst bei der Lautlosfunktion als Reaktion auf einen Alarm den Lautlosmodus-Wechsel des Gerätes.

Alle drei Kernkomponenten des Veranstaltungsplans – Wochenansicht, Formular und Lautlosfunktion – benötigen jeweils eine zentrale TerminLogik-Komponente. Diese kann einfach als ein statisches Werkzeug im System gesehen werden, um Semester- und Kalenderwochen-basierte Termine zu ermitteln oder zu prüfen.

Für sämtliche Zugriffe auf die Datenbank dient die Persistenz-Schnittstelle IVeranstaltungsdatenManager. Auch Sie wird hier von allen drei Kernkomponenten verwendet.

4.4 Beschreibung der Komponenten

Im Folgenden werden die einzelnen Komponenten mit Hilfe von Klassendiagrammen näher vorgestellt. Dabei werden die Schnittstellen erläutert, die sich aus den Systemoperationen der in Abschnitt 3.7 beschriebenen Anwendungsfälle ergeben. In den Diagrammen wird zugunsten der Übersichtlichkeit teilweise auf die Darstellung von Methodenparametern verzichtet. Die vollständigen Signaturen sind in dem Fall den Erläuterungen zu entnehmen. Die Komponentenstruktur soll durch die Organisation der Klassen in entsprechenden Java-Paketen deutlich werden.

4.4.1 Startmenü

Der Hauptbildschirm der Anwendung ist in der Komponente „Startmenü“ mit einer Activity realisiert, die über einen Menübutton die Veranstaltungsplan-Wochenansicht und über ein Optionsmenü die Einstellungen anzeigt (Abb. 4.5). Für das Optionsmenü werden die Standardmethoden einer Activity überschrieben: `onCreateOptionsMenu(..)` und `onOptionsItemSelected(..)`. Die Kommunikation geschieht jeweils per Intent.

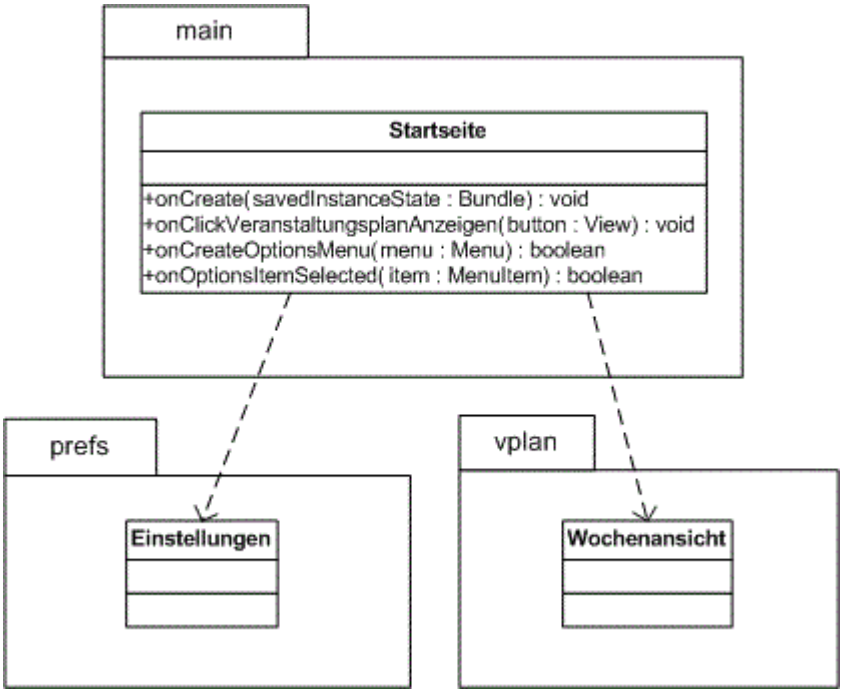

Abbildung 4.5: Klassendiagramm für die Studienplaner-Komponente „Startmenü“

Die benötigten Systemoperationen für den Anwendungsfall „Startseite anzeigen“ veranschaulicht das Sequenzdiagramm in Abbildung 4.6. Beim Aufrufen der Startseite wird die Lautlosfunktion geprüft und, falls notwendig, der Lautlosmodus (Klingelmodus „silent“) aktiviert. Die Prüfung geschieht in der `onCreate(..)`-Methode der Startseite. Über die Klasse „Einstellungen“ wird dabei sowohl der Status abgefragt, als auch der LautlosModusService gestartet.

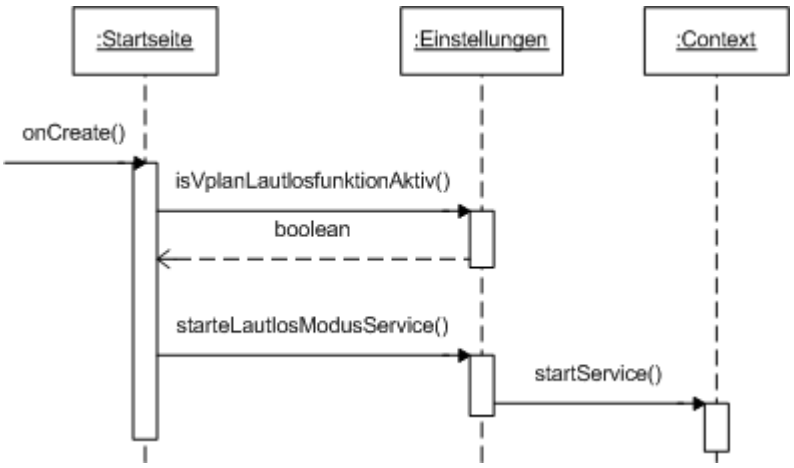

Abbildung 4.6: Sequenzdiagramm für den Anwendungsfall „Startseite anzeigen“

Der weitere Ablauf nach dem Start des Service erfolgt nach dem Anwendungsfall „Lautlostermin vormerken“ (vgl. Abschnitt 3.73.7.1).

4.4.2 Einstellungen

Die Komponente „Einstellungen“ beinhaltet lediglich eine gleichnamige PreferenceActivity, die die Bildschirmseite zur Konfiguration enthält und über statische Methoden den Zugriff auf die gemeinsamen Einstellungen der Anwendung gewährt. In dem Klassendiagramm in Abbildung 4.7 sind die bereitgestellten Methoden aufgeführt. Die beiden Methoden `starteLautlosModusService(..)` und `stoppeLautlosModusService(..)` dienen zum Starten und Stoppen des LautlosModusService aus einem beliebigen Context heraus. Der zum Start übermittelte Boolean gibt an, ob der Lautlosmodus über den Aufruf aktiviert oder deaktiviert werden soll. Zur De-/Aktivierung der Lautlosfunktion über die entsprechende Preference werden die beiden Methoden beispielsweise aufgerufen.

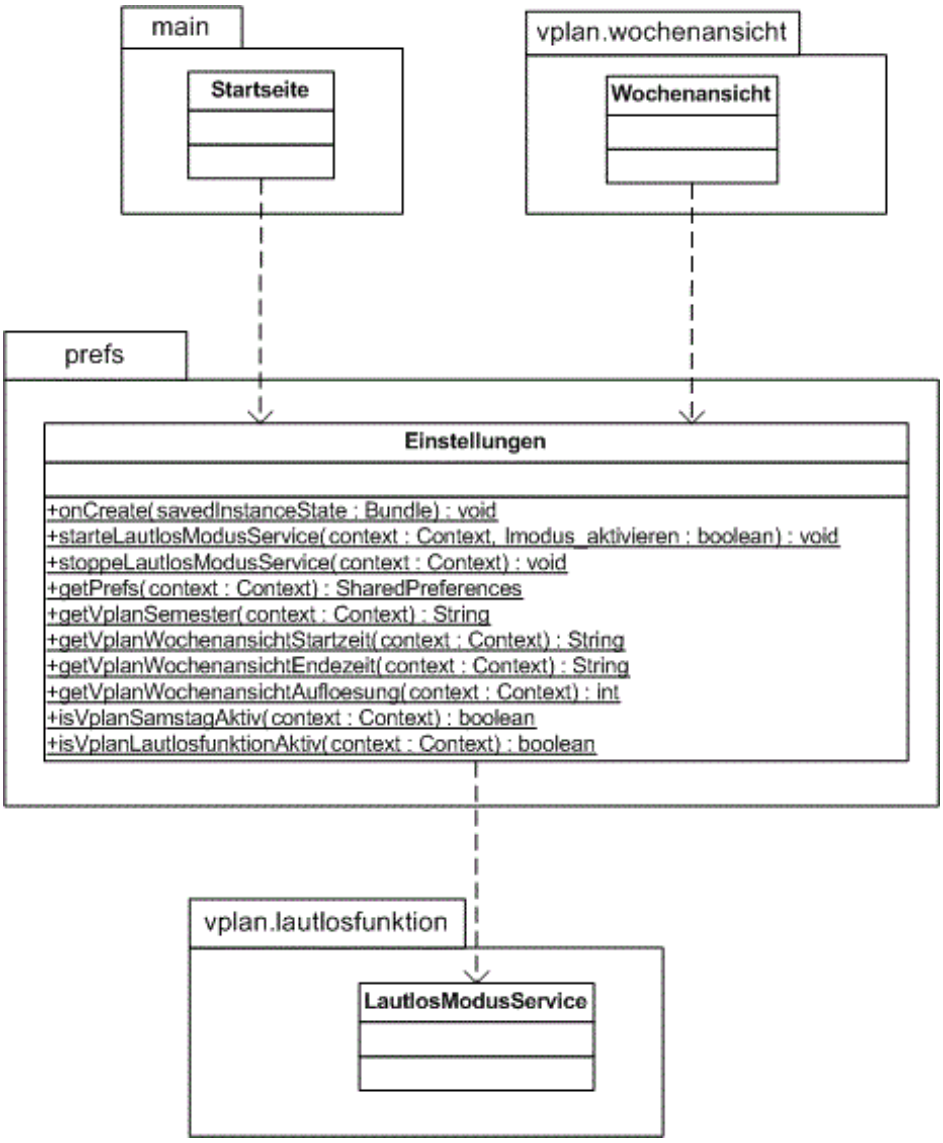

Abbildung 4.7: Klassendiagramm für die Studienplaner-Komponente „Einstellungen“

Die nachfolgenden Sequenzdiagramme veranschaulichen den Aufruf der Bildschirmseite mit den Einstellungen aus dem Optionsmenü (Abb. 4.8), sowie die Systemoperationen nach dem Anwendungsfall „Einstellungen ändern“ (Abb. 4.9) am Beispiel der Semestereinstellung.

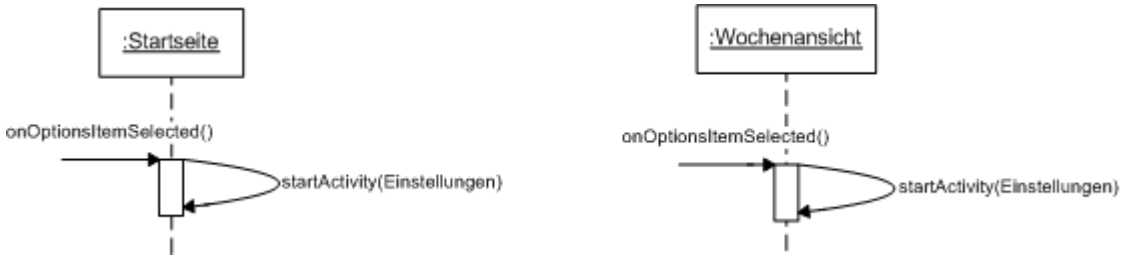

Abbildung 4.8: Sequenzdiagramme für den Anwendungsfall „Einstellungen anzeigen“

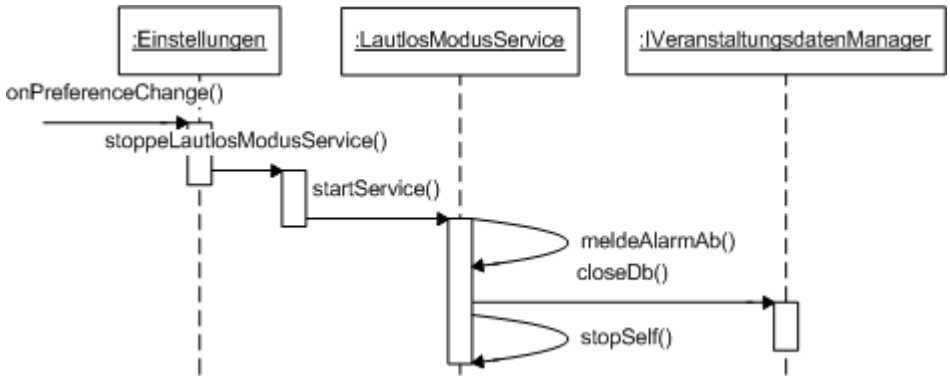

Abbildung 4.9: Sequenzdiagramm für den Anwendungsfall „Einstellungen ändern“ – Änderung des Semesters

Sobald das System eine Änderung an der Semester-Preference registriert, wird der LautlosModusService gestoppt und vorher dazu veranlasst, alle eingestellten Alarme abzumelden. Das System persistiert die neuen Einstellungen und zeigt die Änderungen an.

4.4.3 Persistenz-Management

Das Datenbankschema der Anwendung (vgl. Abschnitt 4.2) ist in den beiden Klassen Tabelle_Veranstaltungen und Tabelle_V_in_Kw des Persistenz-Managements definiert. Sie enthalten Konstanten für den jeweiligen Tabellennamen und für eine Reihe von SQL-Anweisungen, mit denen die Tabellen erstellt und modifiziert werden können. Damit sind häufiger verwendete SQL-Statements zentral abgelegt; was die spätere Wartung erleichtert.

Die physische Datenbank der Studienplaner-Anwendung wird durch die Klasse StudienplanerDatenbank verwaltet. Diese leitet von SQLiteOpenHelper ab (siehe Abschnitt 2.4.8) und dient dazu, die Datenbank mit Hilfe der Schema-Klassen zu erstellen oder zu aktualisieren, falls es erforderlich ist. Hierfür sind die beiden Methoden `onCreate (SQLiteDatabase db)` und `onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion)`.

Der DatenbankManager implementiert die Schnittstelle IVeranstaltungsdatenManager (Abbildung 4.10) und stellt damit Methoden bereit, um Veranstaltungsdaten zu verwalten. Er verwendet die StudienplanerDatenbank, um diese Veranstaltungsdaten auszulesen und zu speichern.

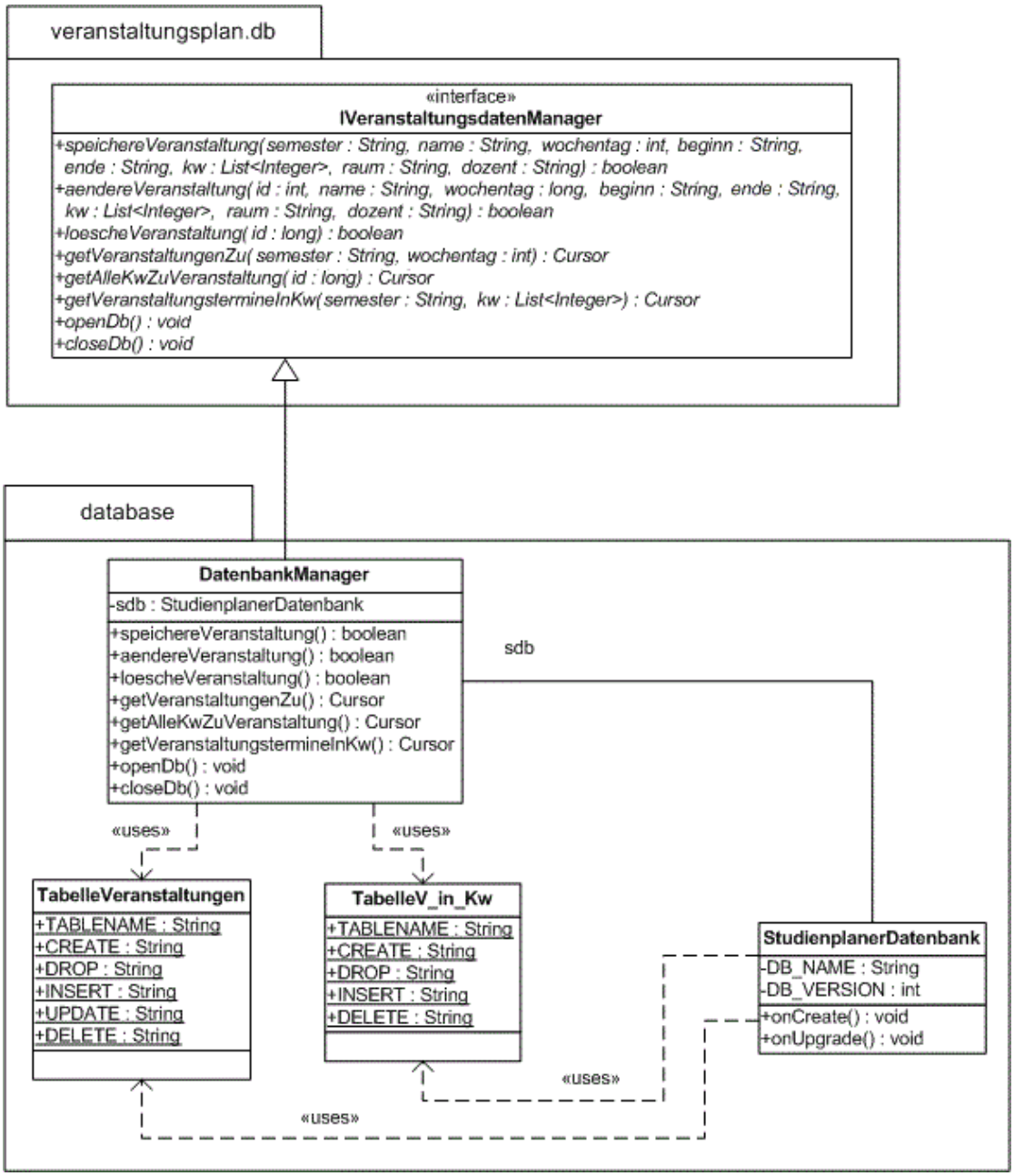

Abbildung 4.10: Die Studienplaner-Komponente „PersistenzManagement“ (Package „database“)

4.4.4 Alarm-Management

Das Alarm-Management kann in zwei unterschiedliche Aufgabenbereiche unterteilt werden – zum einen das Anmelden von Alarmen beim AlarmManager per IntentService und zum anderen das Reagieren auf die Alarme per BroadcastReceiver. Diese beiden Funktionen erfüllt die Komponente in drei abstrakten Klassen, die im Folgenden beschrieben werden.

Der WakeLockedService ist ein IntentService, der verhindert, dass das Gerät während seiner Ausführung die CPU abschaltet, um Energie zu sparen. Zum Beispiel könnte ein Service beim

automatischen – oder durch Betätigung der Power-Taste am Gerät ausgelösten – Eintritt in den Standby-Modus nicht mehr korrekt arbeiten. Um zu gewährleisten, dass die CPU aktiv bleibt, fordert der WakeLockedService vor der eigentlichen Ausführung des Service in `onHandleIntent(..)` mit `aquireLock()` einen „partial wakelock“ vom `android.os.PowerManager` an. Nach Erledigung der Arbeiten in der Methode `doWakeLockedService(..)` gibt er diesen mit `releaseLock()` wieder frei, so dass die CPU anschließend abgeschaltet werden kann.

Erweitert wird die Funktionalität des WakeLockedService von der Klasse AlarmAnmeldeService, um Alarme beim Android AlarmManager (siehe Abschnitt 2.4.7) zu verwalten. Er fordert von den Subklassen, dass sie einige Methoden implementieren, um den Service zu beschreiben. Der empfangene Request kann hierzu die nötigen Informationen enthalten und darum in den einzelnen Methoden ausgewertet werden.
Zwingend bereitgestellt werden müssen von den Subklassen der Alarm-Intent, der bei der Auslösung des Alarms per Broadcast versendet werden soll, sowie der Alarm-Termin. Dies geschieht über die beiden Methoden `getAlarmIntent(..)` und `getAlarmTermin(..)`. Da der Service auch übermittelte Alarme wieder abmelden können soll, wird zusätzlich der Status der Alarmfunktion festgelegt. Hierzu implementieren die Subklassen die Methode `isAlarmFunktionAktiv(..)`, deren Rückgabewert zum Setzen eines Statusflags genutzt wird. In `doWakeLockedService(..)` fragt der AlarmAnmeldeService dieses Flag ab. Bei aktiver Alarmfunktion meldet er einzelne Alarme an, ansonsten meldet er einen gegebenenfalls vorliegenden Alarm ab und beendet sich anschließend. Das eigentliche An- und Abmelden der Alarme wird in den privaten Methoden `meldeAlarmAn(..)` und `meldeAlarmAb(..)` erledigt. Die Methoden `doBeforeStartService(..)` und `doBeforeStopService(..)` können von den Subklassen optional überschrieben werden, um Arbeiten vor dem Start und vor der Beendigung des Service auszuführen.

Die dritte Klasse, der RingerModeChanger, ist ein BroadcastReceiver zur Entgegennahme, Auswertung und Verarbeitung der Alarm-Intents. Er ändert über den AudioManager des Frameworks den Klingelmodus des Gerätes. Hierzu wird von den Subklassen der einzustellende Modus in `getRequestedRingerMode(..)` aus dem Alarm-Intent bestimmt. Da es häufig sinnvoll ist, das Gerät wieder in den ursprünglichen Klingelmodus zurückzuschalten, kann ein konkreter RingerModeChanger zusätzlich die Methode

`scheduleNextMode(..)` implementieren. Der Context, in dem der RingerModeChanger läuft, kann dabei zum Beispiel verwendet werden, um einen Service zu starten, der mit Hilfe des ursprünglichen Request den zuletzt eingestellten Modus identifiziert und entsprechend eine neue, entgegengesetzte Anfrage stellt.

Das nachfolgende Diagramm veranschaulicht die drei vorgestellten Klassen der AlarmManagement-Komponente:

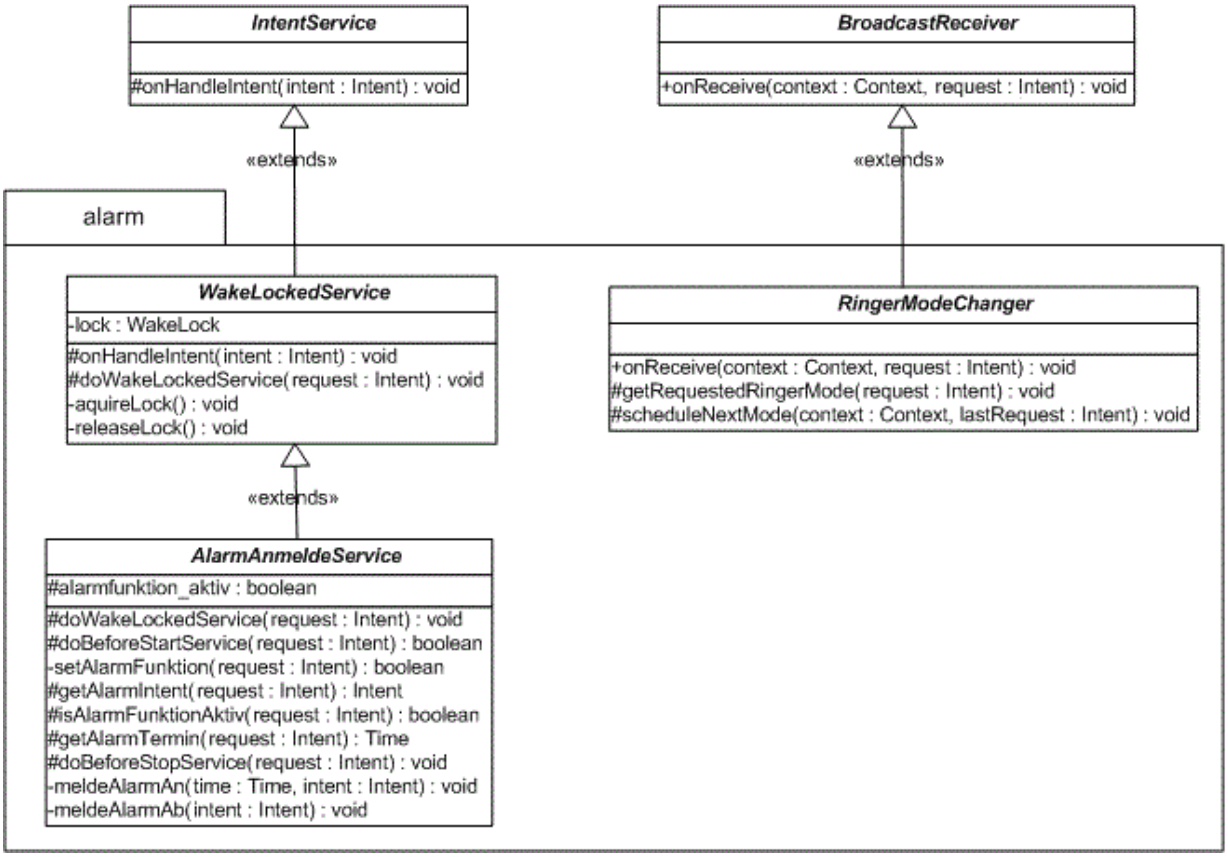

Abbildung 4.11: Die Studienplaner-Komponente „AlarmManagement“

4.4.5 Veranstaltungsplan

Der Veranstaltungsplan lässt sich nach Abbildung 4.4 zunächst in sechs Subkomponenten einteilen:

- Wochenansicht
- Tagesplan
- Formular
- Terminlogik
- Persistenz-Schnittstelle
- Lautlosfunktion

Das Klassendiagramm zeigt die Paketstruktur und gibt einen Überblick über die bestehenden Abhängigkeiten:

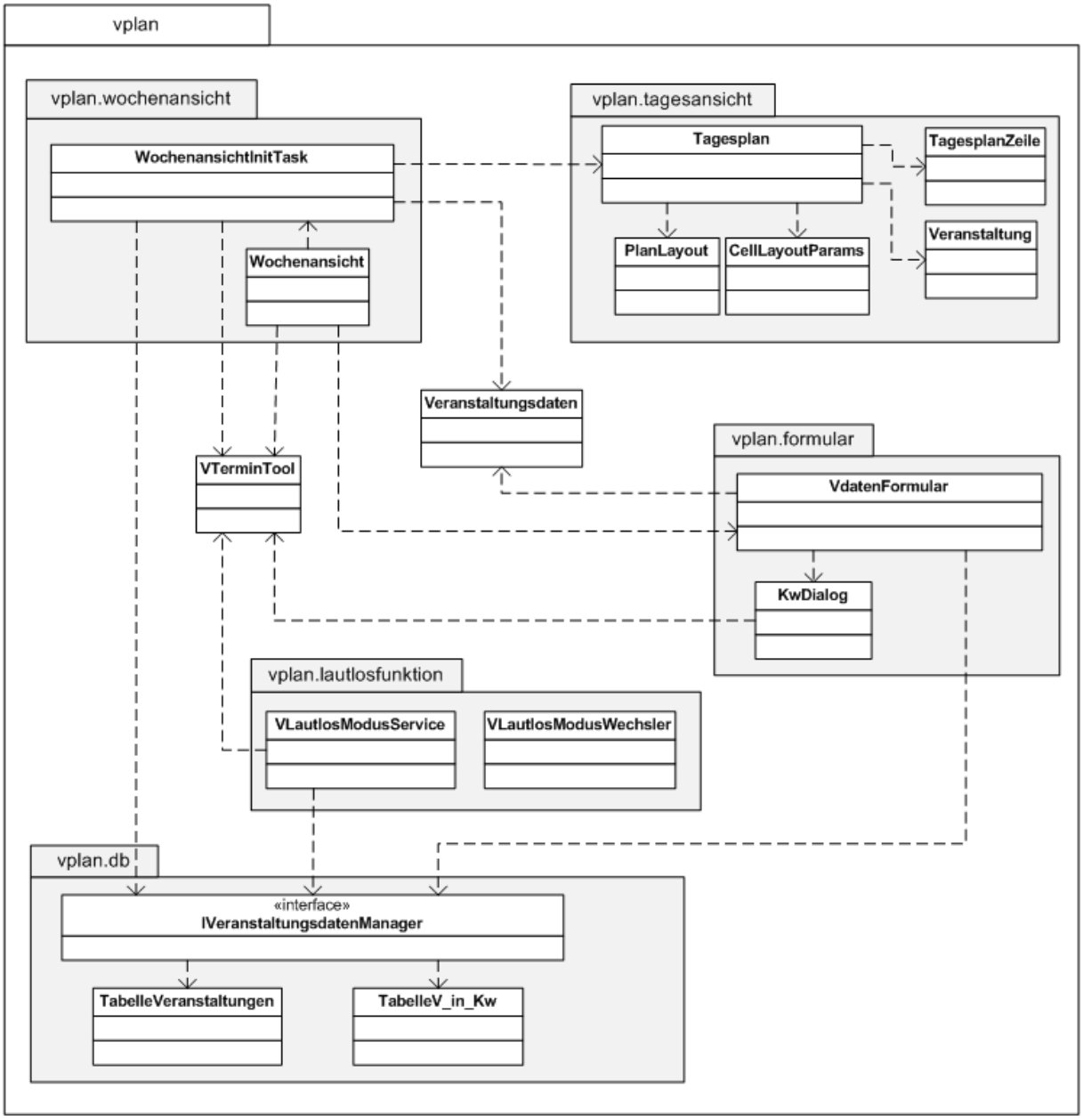

Abbildung 4.12: Klassendiagramm für die Komponente „Veranstaltungsplan“

In den folgenden Abschnitten werden die einzelnen Veranstaltungsplan-Komponenten, die in den dargestellten Paketen organisiert sind, näher beschrieben.

Wochenansicht

Die Wochenansicht-Komponente beinhaltet eine TabActivity „Wochenansicht“ zur Darstellung der einzelnen Tagespläne, sowie einen asynchronen Task „WochenansichtInitTask“ zum Laden der Veranstaltungsdaten aus der Datenbank und zum Zusammenstellen der Tagespläne für die einzelnen Tabs in der TabActivity.
Der Einsatz eines AsyncTask für diese Aufgaben bietet im Wesentlichen zwei Vorteile. Zum einen blockiert auch ein länger andauernder Vorgang zum Laden der Veranstaltungsdaten nicht den UI-Thread (vgl. 2.4.10): Es ist vorstellbar, die Daten statt direkt aus der lokalen Datenbank zum Beispiel über eine langsame Netzwerkverbindung zu laden, was dann keinen störenden Einfluss auf die Reaktion der Benutzeroberfläche – zum Beispiel zum Öffnen eines Menüs – hätte. Zum anderen kapselt ein AsyncTask die aufwendigen Aufgaben zur Initialisierung der Wochenansicht, was zu einer besseren Wartbarkeit und Lesbarkeit des Codes beiträgt. Darüber hinaus wird, falls notwendig, die Anzeige eines Fortschrittsbalkens unterstützt.
Über die Schnittstelle zu der Komponente „Einstellungen“ werden die Benutzereinstellungen abgefragt, welche das Layout des Veranstaltungsplans und die zu ladenden Veranstaltungsdaten bestimmen:

`isVplanSamstagAktiv(Context c) : boolean`
`getVplanSemester(Context c) : String`
`getVplanWochenansichtStartzeit(Context c) : String`
`getVplanWochenansichtEndezeit(Context c) : String`
`getVplanWochenansichtAufloesung(Context c) : int`

Tabelle 4.1: Die Schnittstelle der Wochenansicht zu den Einstellungen

Um die Veranstaltungsdaten aus der Datenbank zu laden, stellt das PersistenzManagement mit der Schnittstelle IVeranstaltungsdatenManager die folgenden Operationen bereit:

`getVeranstaltungenZu(semester, wochentag) : Cursor`
`getAlleKwZuVeranstaltung(int vid) : Cursor`
`loescheVeranstaltung(int vid) : boolean`
`openDb() : void`
`closeDb() : void`

Tabelle 4.2: Die Schnittstelle der Wochenansicht zum PersistenzManagement

Die Methode `getVeranstaltungenZu(String semester, int wochentag)` liefert die Daten zu allen Veranstaltungen im Semester für einen Tagesplan: Id, Name,

Beginn, Ende, Raum und Dozent. Die Kalenderwochen sind hier nicht enthalten. Sie stellen eine Besonderheit dar, da das Format, in dem sie in der Datenbank gespeichert sind, für die Darstellung im Plan zunächst in einen String der geforderten Kurzschreibweise umgewandelt werden muss. Mit `getAlleKwZuVeranstaltung(int vid)` können alle Kalenderwochen zu einer Veranstaltung geladen werden, um sie anschließend für die Anzeige zu formatieren.
Zum Löschen einer Veranstaltung aus dem Kontextmenü der Wochenansicht heraus stellt die Schnittstelle die Methode `loescheVeranstaltung(int vid)` zur Verfügung. Die hierbei benötigte Verbindung zur Datenbank wird an die Lebenszeit der Activity gekoppelt. So bleibt sie verfügbar, bis diese beendet ist: In `onCreate(..)` wird die Verbindung per `openDb()` geöffnet, in `onDestroy(..)` per `closeDb()` geschlossen.

Nach dem Laden aus der Datenbank müssen die Angaben zu den Kalenderwochen – wie erwähnt – noch formatiert werden. Bewerkstelligt wird dies durch eine Schnittstelle zur TerminLogik, die in der nachfolgenden Tabelle beschrieben ist:

`getShortKwStringFor(String semester, List<Integer> kws) : String`
`getAktuelleKw() : int`
`getKwAnfangEndeDatum(String semester, int kw) : String`
`getAktuellerWochentag() : int`

Tabelle 4.3: Die Schnittstelle der Wochenansicht zur TerminLogik

Mit `getShortKwStringFor(String semester, List<Integer> kws)` liefert die TerminLogik einen Kalenderwochen-String, der eine Liste von Kalenderwochen für ein Semester in der Kurzschreibweise zusammenfasst. Die für die Titelzeile in der Wochenansicht benötigten Angaben zur aktuellen Kalenderwoche mit ihren Datumsgrenzen werden über `getAktuelleKw()` und `getKwAnfangEndeDatum(String semester, int kw)` abgerufen. Die Methode `getAktuellerWochentag()` dient dazu, den Tab zum aktuellen Wochentag beim Start aktivieren zu können.

Das Sequenzdiagramm in Abbildung 4.13 zeigt die Verwendung der Schnittstellen für den Anwendungsfall „Wochenansicht anzeigen“ im chronologischen Zusammenhang. Die dargestellten Methoden zur Verwendung eines AsyncTask `onDoInBackground(..)` und `onPostExecute(..)` wurden bereits im Abschnitt 2.4.10 erläutert.

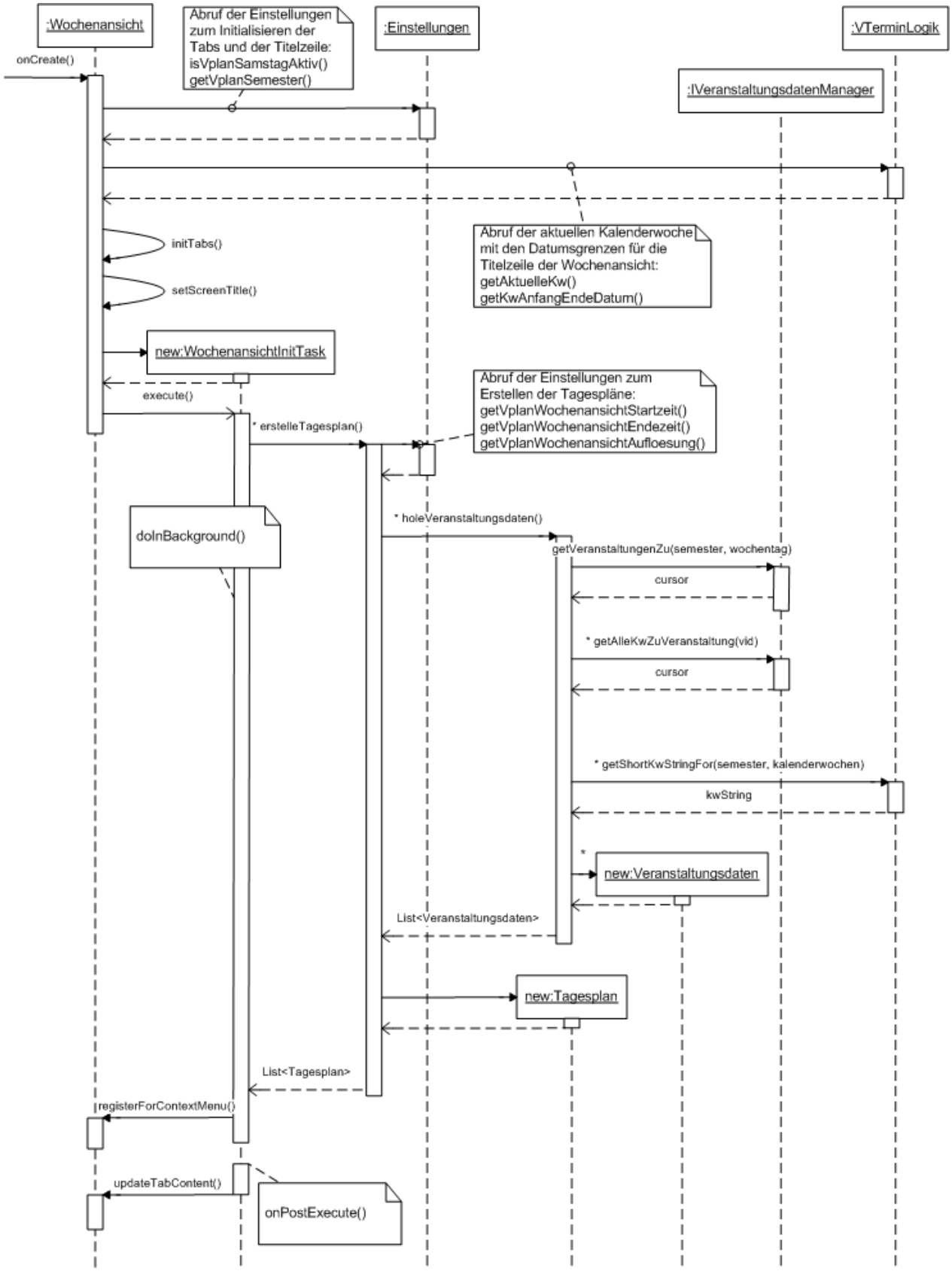

Abbildung 4.13: Sequenzdiagramm für den Anwendungsfall „Wochenansicht anzeigen“

Nachdem der Task die Tagespläne aus den Veranstaltungsdaten zusammengestellt hat, registriert er alle Veranstaltungen und Tagesplanzeilen für das Kontextmenü. Danach ruft er die Callback-Methode `updateTabContent(List<Tagesplan>)` der Wochenansicht auf, um die Tabs mit den Tagesplänen zu aktualisieren.

Das Löschen einer Veranstaltung per Kontextmenü aus der Wochenansicht heraus ist durch die Systemoperationen zum Anwendungsfall „Veranstaltung löschen“ beschrieben:

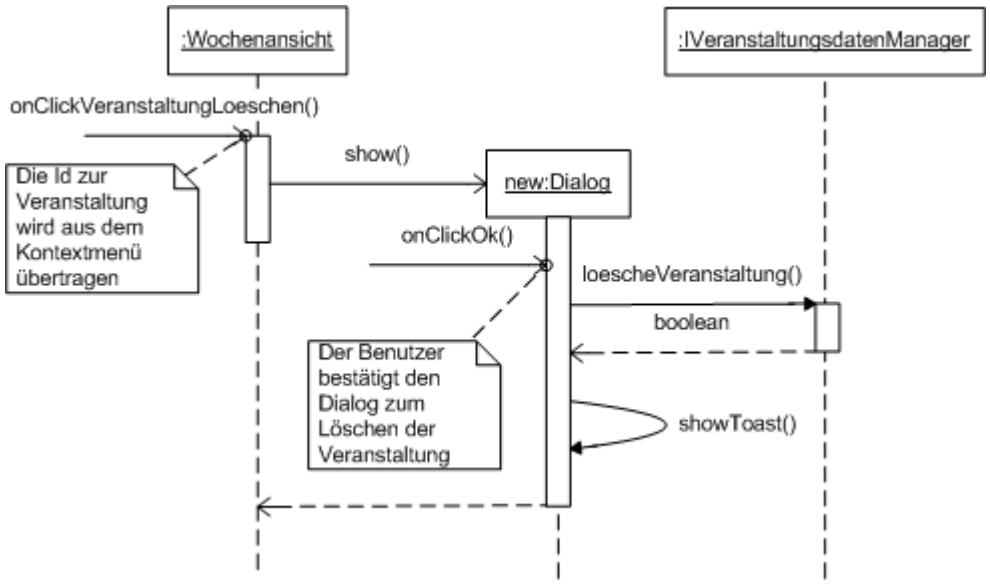

Abbildung 4.14: Sequenzdiagramm für den Anwendungsfall „Veranstaltung löschen“

In einem Dialog muss die Anforderung zum Löschen vom Benutzer mit „OK“ bestätigt werden. Der von der Methode `loescheVeranstaltung(..)` zurückgelieferte Boolean wird ausgewertet, um den Benutzer über Erfolg oder Misserfolg der Aktion zu informieren und die Wochenansicht im Erfolgsfall neu zu starten, damit die Änderungen sichtbar werden.

Einen detaillierten Klassenentwurf für die Wochenansicht-Komponente zeigt die nachfolgende Abbildung. Eine Beschreibung der ebenfalls dargestellten Tagesplan-Komponente folgt im nächsten Abschnitt.

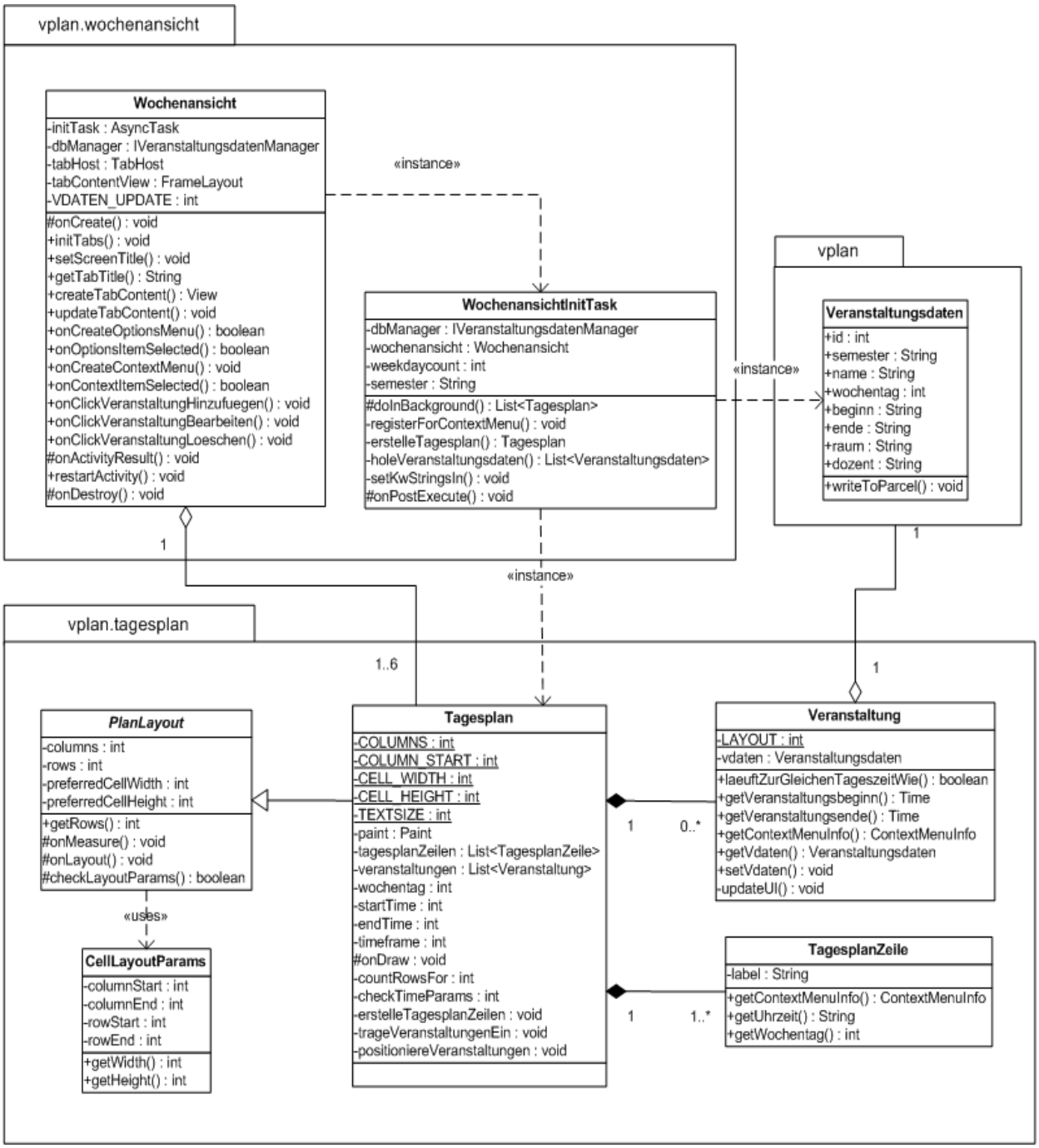

Abbildung 4.15: Die Veranstaltungsplan-Komponenten „Wochenansicht“ und „Tagesplan“

Zur Realisierung der Options- und Kontextmenüs überschreibt die Klasse „Wochenansicht“ entsprechende Activity-Methoden: In `onCreateOptionsMenu(Menu menu)` und `onCreateContextMenu(ContextMenu menu, View v, ContextMenuInfo menuInfo)` werden die jeweiligen Menüs aus einer XML-Ressource geladen. Das Verhalten bei Auswahl einer Menüoption ist in `onOptionsItemSelected(MenuItem item)` und `onContextItemSelected(MenuItem item)` festgelegt. Hierzu

gehören das Starten der Bildschirmseite mit den Einstellungen per `startActivity(Intent intent)`, sowie das Hinzufügen, Bearbeiten und Löschen von Veranstaltungen.

Die Kontextmenü-Option zum Löschen einer Veranstaltung wurde in diesem Abschnitt bereits beschrieben. Für die ebenfalls über das Kontextmenü verfügbare Option zum Bearbeiten einer Veranstaltung implementiert die Wochenansicht zusätzlich die Methode `onClickVeranstaltungBearbeiten()`. Das Eintragen einer neuen Veranstaltung veranlasst sie in der Methode `onClickVeranstaltungHinzufuegen()`, welche aus dem Options- oder Kontextmenü heraus aufgerufen wird.

Sowohl zum Bearbeiten, als auch zum Hinzufügen von Veranstaltungen wird per `startActivityForResult(Intent intent, int requestCode)` der Start des Formulars zur Erfassung von Veranstaltungsdaten ausgelöst. Mit diesem Aufruf wird das Formular zur Subactivity der Wochenansicht (vgl. Abschnitt 2.4.4). Zum Bearbeiten übermittelt die Methode in dem Start-Intent die Daten der im Plan ausgewählten Veranstaltung in Form eines Veranstaltungsdaten-Objekts. Die Klasse „Veranstaltungsdaten“ implementiert die Parcelable-Schnittstelle und ermöglicht damit den Transport der Daten per Intent. Bei der Menüoption „Hinzufügen“ werden entweder nur der selektierte Wochentag (Aufruf aus dem Optionsmenü), oder der Wochentag und die im Plan ausgewählte Uhrzeit (Aufruf aus dem Kontextmenü) an das Formular übermittelt.

Neben den Informationen im Intent wird dem Formular auch ein spezieller Request-Code (`VDATEN_UPDATE`) übergeben. Dieser wird in der Callback-Methode `onActivityResult(int requestCode, int resultCode, Intent intent)` zusammen mit einem Result-Code ausgewertet, nachdem die Formular-Activity beendet wurde. Falls neue Veranstaltungsdaten in der Datenbank vorliegen, wird per `restartActivity()` ein Neustart der Wochenansicht veranlasst, um die Änderungen anzuzeigen.

Tagesplan

Da das Android TableLayout keine zeilenübergreifenden Zellen („row spanning“) unterstützt, wird für das Layout eines Tagesplans eine eigene ViewGroup verwendet. Diese ist in der abstrakten Klasse „PlanLayout“ realisiert. Ein PlanLayout enthält eine bestimmte Anzahl an Zeilen festgelegter Höhe, sowie Spalten festgelegter Breite, innerhalb der spalten- und zeilenübergreifende Zellen beliebig positioniert werden können. Zu diesem Zweck definiert es die Layout-Parameter für eine Child-View (eine Zelle) mit einer Start-und End- Zeile, sowie einer Start- und End- Spalte. Die folgende Abbildung skizziert die Indizierung der Zellen in dem Layout – sie ist zur Einfachheit an den Gitterlinien orientiert:

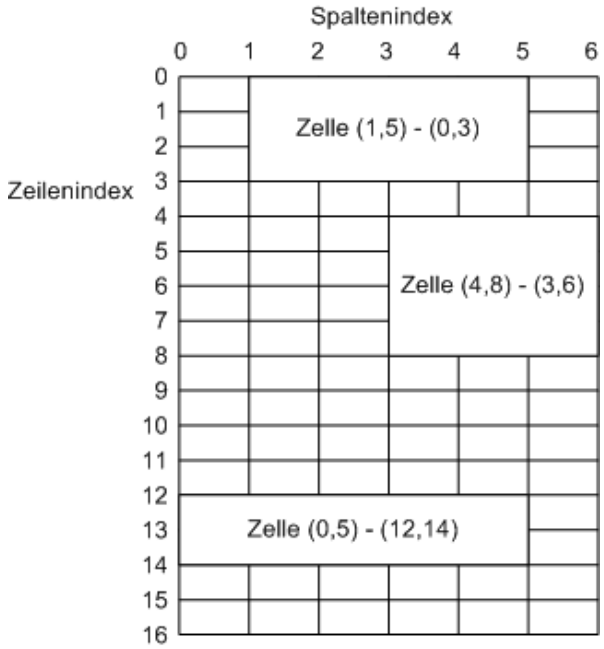

Abbildung 4.16: Die Indizierung in einem PlanLayout

Die Layout-Parameter einer Zelle beinhaltet die Klasse CellLayoutParams (Abb. 4.15). Das PlanLayout überprüft diese Parameter in `checkLayoutParams(LayoutParams layoutParams)`. Die Methode liefert einen Boolean zurück, der angibt, ob die View dargestellt werden kann (`true`) oder nicht (`false`). Falls die Höhe und Breite einer Zelle nicht mindestens 1 ist, oder die verwendeten Indizes außerhalb des Planbereiches liegen, wird sie nicht dargestellt. Um die Pixelbreite und -höhe der darstellbaren Child-Views zu bestimmen und ihnen ihre Abmessungen und ihre Positionen zuzuweisen (siehe 2.4.3), implementiert das PlanLayout zusätzlich die beiden von ViewGroup vererbten Methoden

`onMeasure(int widthMeasureSpec, int heightMeasureSpec)` und `onLayout(boolean changed, int l, int t, int r, int b)`.

Ein Tagesplan (Abb. 4.15) ist ein PlanLayout für einen bestimmten Wochentag, das sich aus zwei Arten von Views zusammensetzt: den einzelnen Tagesplan-Zeilen und den eingetragenen Veranstaltungen. Bei der Instanziierung werden sowohl die Zeilen, als auch die Veranstaltungen mit den Layout-Parametern einer PlanLayout-Zelle erzeugt. Die Positionen und Abmessungen der Veranstaltungen können damit von vornherein so festgelegt werden, dass alle Veranstaltungen, die sich zeitlich überschneiden, im Plan nebeneinander liegen. Die Tagesplan-Zeilen erstrecken sich immer vom ersten bis zum letzen Spaltenindex. Sie dienen insbesondere dazu, einem Kontextmenü die Informationen zum Wochentag des Plans und zur jeweiligen Uhrzeit zur Verfügung zu stellen. Im Folgenden wird die Konstruktion eines Tagesplans beschrieben.

Der Plan wird erzeugt für eine Start- und Endzeit, sowie eine zeitliche Auflösung in Minuten, die der Größe des Zeitintervalls für eine Tagesplan-Zeile entspricht. Die Methode `checkTimeParams(Time startTime, Time endTime, int timeframe)` prüft die im Konstruktor verwendeten Werte für diese Parameter. Zulässig sind die Auflösungen 5, 15, 30 und 60 Minuten (siehe Anforderung R20). Die Dauer muss mindestens fünf Stunden betragen, so dass der Plan auch bei der höchsten Auflösung noch einige Zeilen enthält. Anschließend wird in `countRowsFor(startTime, endTime, timeframe)` die Anzahl der benötigten Zeilen für das zugrunde liegende PlanLayout ermittelt.
Die Intervallgrenzen der Zeilen in den vier möglichen Auflösungen sind zur einfachen Orientierung durch ganzzahlige Faktoren festgelegt (z.B. bei Auflösung = 15 Minuten → Zeilenbeginn/-ende jeweils in Minute 0, 15, 30 oder 45). Um einheitliche Zeitintervalle darzustellen, wird die Startzeit des Plans auf die Minute des Intervalls, in das sie fällt, abgerundet und die Endzeit auf das entsprechende Intervallende aufgerundet.
Als Beschriftung enthält jede Zeile die Startzeit des jeweiligen Intervalls. Die Zeile, die am Index 0 beginnt (vgl. Abbildung 4.16) ist also mit der Startzeit des Tagesplans beschriftet. Bei den folgenden Zeilen ist die Zeit jeweils um die verwendete Auflösung inkrementiert. Die Endzeit markiert den letzten für den Plan verwendeten Zeilenindex und stellt die untere Begrenzung der letzten Tagesplan-Zeile dar.

Nach der Erstellung der TagesplanZeilen werden die Veranstaltungen eingetragen und positioniert. Sie liegen über den Zeilen, so dass in dem Plan-Bereich, über den sich eine Veranstaltung erstreckt, das Kontextmenü der dahinter liegenden Zeilen verdeckt wird, während das Kontextmenü der Veranstaltung dort aufrufbar ist.
Die Methode `trageVeranstaltungenEin(List<Veranstaltungsdaten> vdatenListe)` erzeugt die einzelnen Veranstaltungen zu den im Konstruktor übergebenen Veranstaltungsdaten. Das Layout der Views wird dabei aus einer XML-Ressource geladen, um eine einfache Anpassung der Darstellung zu erlauben. Als Layout-Parameter für die Zeilen verwendet die Methode die mit dem jeweiligen Veranstaltungsbeginn und -ende korrespondierenden Indizes. Für die Minute, in der sich zwei Tagesplan-Zeilen treffen, wird bei einem Veranstaltungsbeginn der Index der unteren Zeile gewählt, bei einem Veranstaltungsende der Index der oberen Zeile. Eine Veranstaltung belegt damit erst eine neue Zeile, wenn sie in der zweiten Minute des entsprechenden Intervalls endet. Veranstaltungen, die genau zu Beginn der ersten Zeile anfangen oder genau zum Ende der letzten Zeile enden, stellt der Plan noch dar, eine Minute oder mehr außerhalb des Plans beginnende/endende nicht mehr.
Als Layout-Parameter für die Spalten setzt die Methode zunächst die festen Werte 2 und das Spaltenmaximum des Tagesplans. So ist gewährleistet, dass der vorhandene Platz auf der rechten Seite voll ausgenutzt wird und die Uhrzeiten auf der linken Seite nicht verdeckt sind.

In der Methode `positioniereVeranstaltungen()`erfolgt eine nachträgliche Anpassung der Spalten-Parameter aller Veranstaltungen, die zeitlich parallel laufen, so dass die verfügbare Breite im Plan möglichst gleichmäßig aufgeteilt ist. Bei zwei nebeneinanderliegenden Veranstaltungen bekommt die erste den Bereich 2 bis 5 und die zweite den Bereich 5 bis 8. Bei drei Veranstaltungen sind es die Werte 2-4, 4-6 und 6-8. Für mehr als drei parallele Veranstaltungen muss der Tagesplan zunächst nicht ausgelegt sein (siehe Anforderung R5), daher reicht der Wert 8 als maximaler Spaltenwert aus.

Das Sequenzdiagramm in Abbildung 4.17 veranschaulicht die benötigten Systemoperationen zur Konstruktion eines Tagesplans. Die onDraw()-Methode der View-Klasse dient zum Zeichnen der Zeilenbeschriftung und der Zeilenbegrenzungen.

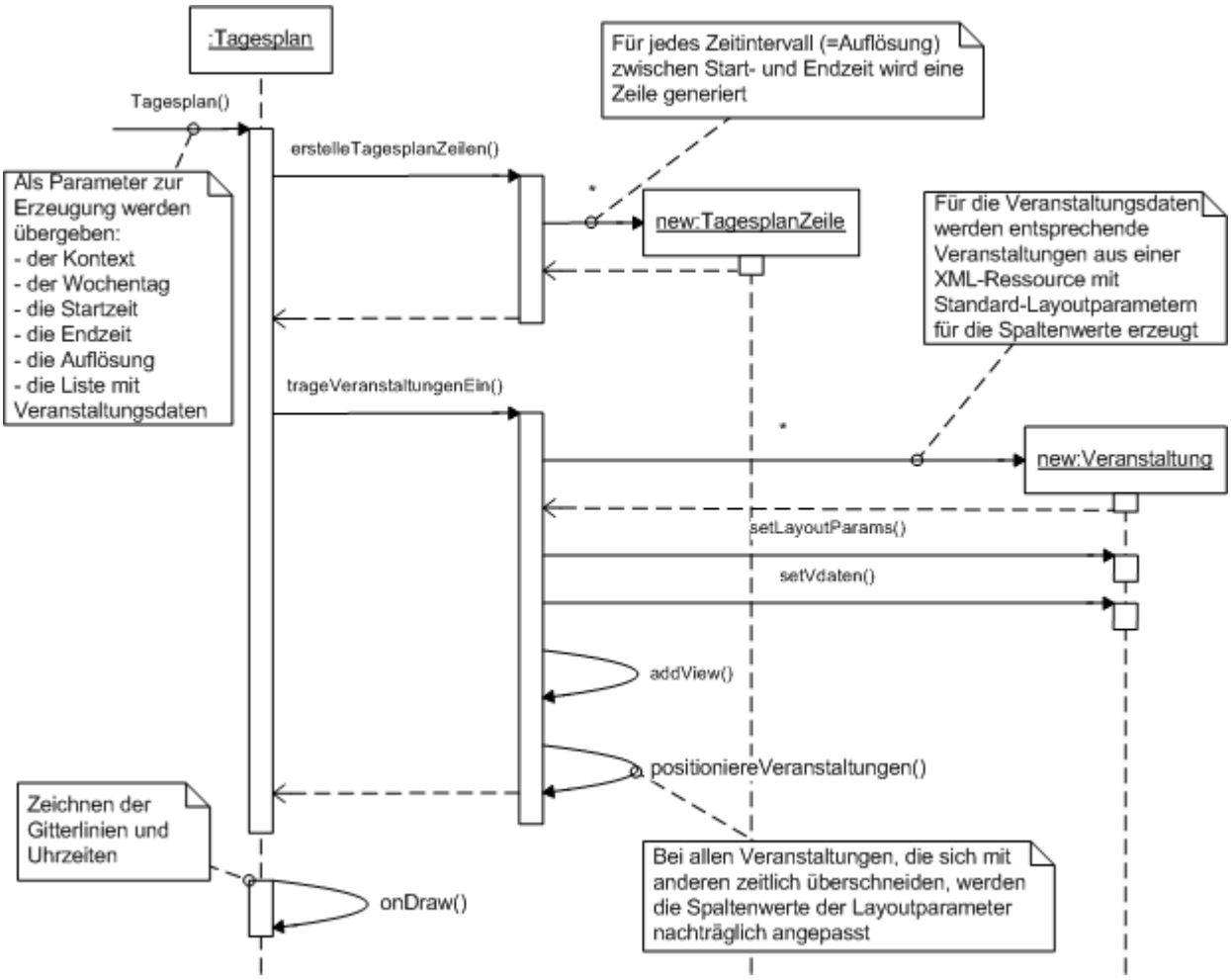

Abbildung 4.17: Sequenzdiagramm zum Erstellen eines Tagesplans

Formular

Die „Formular“-Komponente besteht aus einer Formular-Activity und einem Checkbox-Dialog zur Auswahl von Kalenderwochen. Die Formular-Activity enthält zur Erfassung der Veranstaltungsdaten die in der nachfolgenden Tabelle aufgeführten Formularelemente:

Erfassung von	Formularelement	Erlaubte Eingabewerte
Name	EditText	2-30 alphanumerische Zeichen
Wochentag	Spinner	Montag (Index 0) bis Samstag (Index 5)
Beginn	Button	Uhrzeit im 24h-Format
Ende	Button	Uhrzeit im 24h-Format
Kalenderwochen	EditText + Button	"," , "-" , Leerzeichen, Ziffern
Raum	EditText	0-20 alphanumerische Zeichen
Dozent	EditText	0-20 alphanumerische Zeichen

Tabelle 4.4: Die verwendeten Formularelemente zur Erfassung der Veranstaltungsdaten

Sie werden zeilenweise auf dem Bildschirm angeordnet, wobei sich die Views, die weniger Platz benötigen, eine Zeile teilen (Abb. 4.18). Alle Formularelemente sind in eine ScrollView eingebettet und werden mit dieser aus einer XML-Ressource geladen, um nachträgliche Layout-Änderungen zu erleichtern. Den Focus hat beim Start das oberste Eingabefeld; das Pflichtfeld „Name“ wird erwartungsgemäß als erstes ausgefüllt.

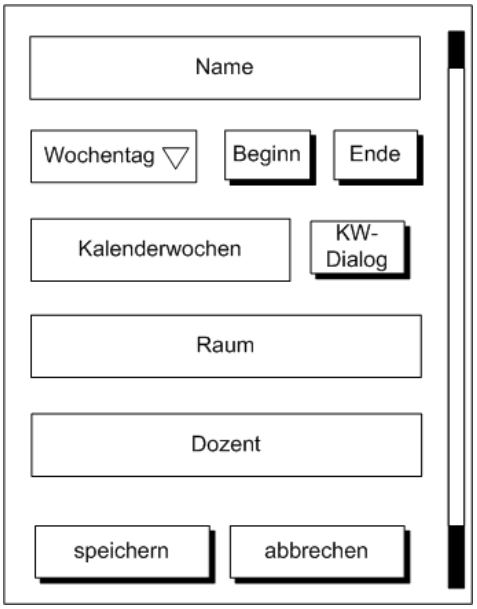

Abbildung 4.18: Anordnung der Formularelemente im Formular zur Erfassung von Veranstaltungsdaten

Die erlaubten Eingabewerte in Tabelle 4.4 sind Anforderung R10 entnommen. Die maximale Anzahl der erlaubten Eingabezeichen lässt sich jeweils für alle EditText-Felder bereits in einem XML-Attribut definieren, um falsche Eingaben zu verhindern und damit auf die entsprechende Überprüfung vor dem Speichern der Formulardaten verzichten zu können.

Die beiden Buttons für Veranstaltungsbeginn und -ende rufen einen TimePickerDialog auf, der zur Eingabe einer Uhrzeit im 24h-Format dient. Initialisiert wird der Dialog mit dem jeweiligen ButtonText. Die vom Benutzer ausgewählte Zeit wird beim Beenden des Dialoges als neuer Text auf dem entsprechenden Button gesetzt.
Einen dritten Button enthält das Formular zum Öffnen des Kalenderwochen-Checkbox-Dialoges. Dieser hat keine weitere Funktion. Das EditText-Feld unmittelbar links neben diesem Button (siehe Abb. 4.18) erlaubt die direkte Eingabe der Kalenderwochen. Ein spezieller KeyListeners lässt in diesem Feld nur die in Tabelle 4.4 angegebenen Eingabezeichen zu.
Zur Erfassung des Wochentages wird der Positionsindex des im Spinner ausgewählten Tages um 1 inkrementiert, so dass eine Übereinstimmung mit dem Datenbankschema (vgl. 4.2) erreicht ist (1=Montag bis 6=Samstag).

Das Sequenzdiagramm in Abbildung 4.19 beschreibt die durch die Formular-Komponente unterstützten Systemoperationen zum Anwendungsfall „Veranstaltung eintragen“. Das Formular ist auf zwei Arten aufrufbar, bei denen die Extras der zum Starten verwendeten Intents unterschiedlich sind:

Zum einen lässt sich das Formular zum Bearbeiten einer vorhandenen Veranstaltung starten. Der Start-Intent enthält dabei die Veranstaltungsdaten. Die Methode `initFormularZumBearbeiten(Veranstaltungsdaten vdaten)` initialisiert das Formular, indem sie den Formularelementen die entsprechenden Daten zuweist.

Zum anderen besteht die Möglichkeit, das Formular zum Erfassen einer neuen Veranstaltung aufzurufen. Dabei beinhaltet der Intent die Angaben des Tagesplans, aus dem es per Kontext- oder Optionsmenü aufgerufen wurde (siehe Anforderung R9). Die Methode `initFormularZumErstellen(Bundle extras)` übernimmt das Semester für den Kalenderwochen-Auswahldialog, den Tagesplan-Wochentag für den Wochentag-Spinner und die Uhrzeit der ausgewählten Tagesplan-Zeile für den Startzeit-Button. Der Endzeit-Button erhält die Zeit zwei Stunden nach der Startzeit. Wurde keine Uhrzeit ausgewählt (Start des

Formulars über das Optionsmenü), initialisiert das System die Buttons mit der Start- und Endzeit des Tagesplans.

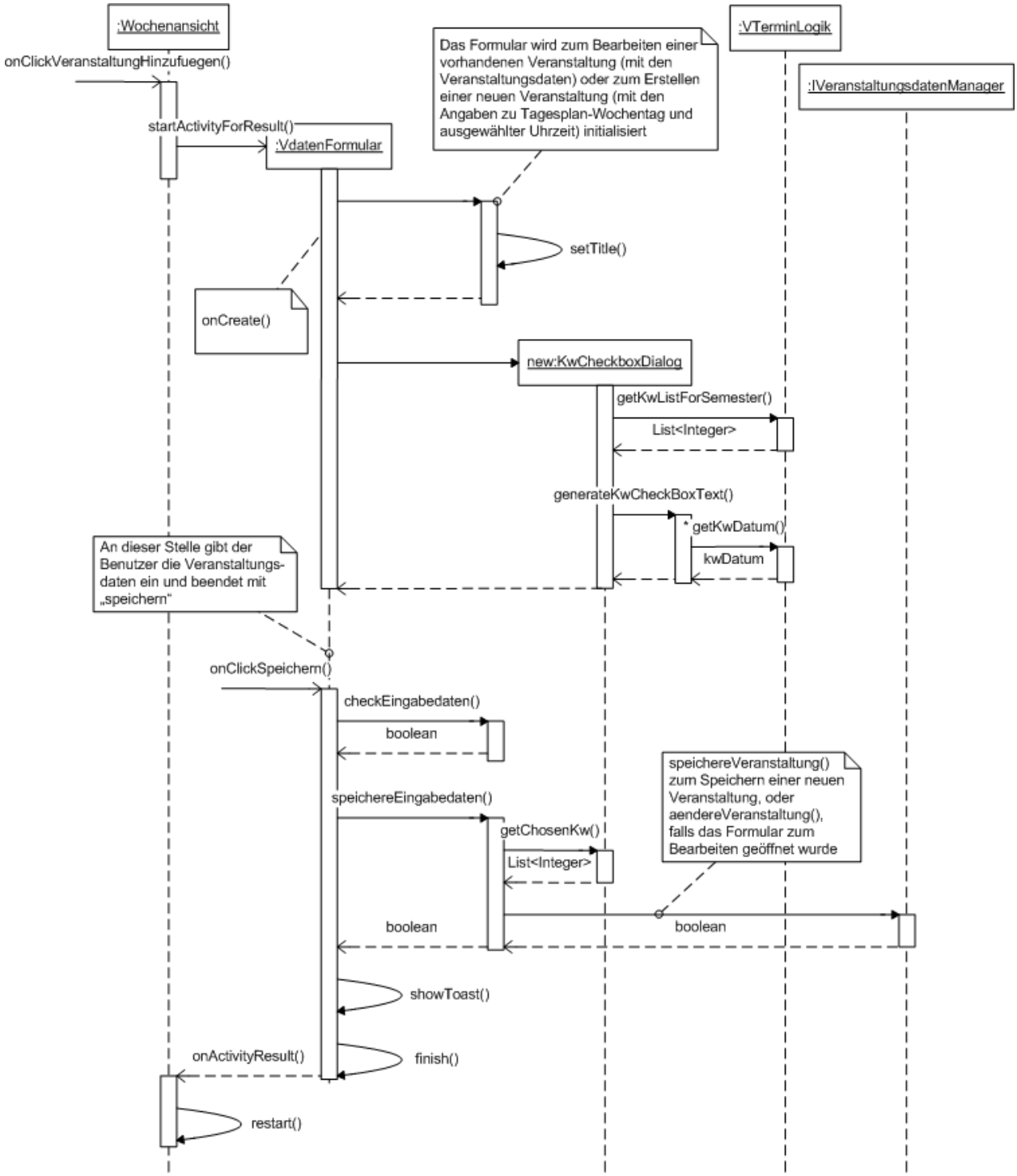

Abbildung 4.19: Sequenzdiagramm zum Anwendungsfall „Veranstaltung eintragen“

Die Schnittstelle zum Persistenz-Management umfasst die Methoden zum Speichern neuer und Ändern vorhandener Veranstaltungsdaten, sowie zum Öffnen der Datenbank beim Start und Schließen der Datenbank beim Beenden des Formulars:

`speichereVeranstaltung(`
`  String semester, String name, int wochentag, String beginn,`
`  String ende, List<Integer> kw, String raum, String dozent`
`  ) : boolean`

`aendereVeranstaltung(`
`  int vid, String name, int wochentag, String beginn, String ende,`
`  List<Integer> kw, String raum, String dozent`
`  ) : boolean`

`openDb() : void`
`closeDb() : void`

Tabelle 4.5: Die Schnittstelle des Formulars zum Persistenz-Management

Zur Auswahl der Kalenderwochen wird ein KwCheckboxDialog erstellt. Hierzu benötigt die Formular-Komponente eine Schnittstelle zur TerminLogik-Komponente:

`getKwListForSemester(String semester) : List<Integer>`
`getKwAnfangEndeDatum(String semester, int kw) : String`
`normalisiereKw(int kw, String semester) : int`
`isSommersemester(String semester) : boolean`

Tabelle 4.6: Die Schnittstelle des Formulars zur TerminLogik

Die Methode `getKwListForSemester(..)` liefert eine Liste aller Kalenderwochen, die in dem betreffenden Semester zur Planung zur Verfügung stehen. Für jede dieser Kalenderwochen wird eine Checkbox in den Auswahldialog eingebunden. Die Checkboxen erhalten als Beschriftung jeweils die Datumsgrenzen der entsprechenden Kalenderwoche. Über `getKwAnfangEndeDatum(..)` werden diese zu jeder Woche von der TerminLogik-Komponente abgerufen.

Die beiden übrigen Methoden der Schnittstelle sind für den Anwendungsfall „Kalenderwochen eintragen“ (Abb. 4.20) vorgesehen. Bei einem Klick auf den Kalenderwochen-Button wird der Dialog zunächst über `showDialog()` aus der Formular-Activity aufgerufen. Er enthält ein Boolean-Array mit den Checkbox-Zuständen aller auswählbaren Kalenderwochen, bei dem alle Zustandsfelder mit dem Wert `false` (für „nicht ausgewählt“) initialisiert sind. Aus der Eingabe im KwEditText wird beim Öffnen in `createKwSetFromEditTextString()` die Menge der Kalenderwochen ermittelt, die

im Array als `true` (ausgewählt) markiert werden müssen. Hierbei dient die TerminLogik-Komponente zur Normalisierung der Kalenderwochen. Anschließend zeigt der Dialog die Checkbox-Elemente mit den Zuständen entsprechend der Wertebelegung des Arrays an.

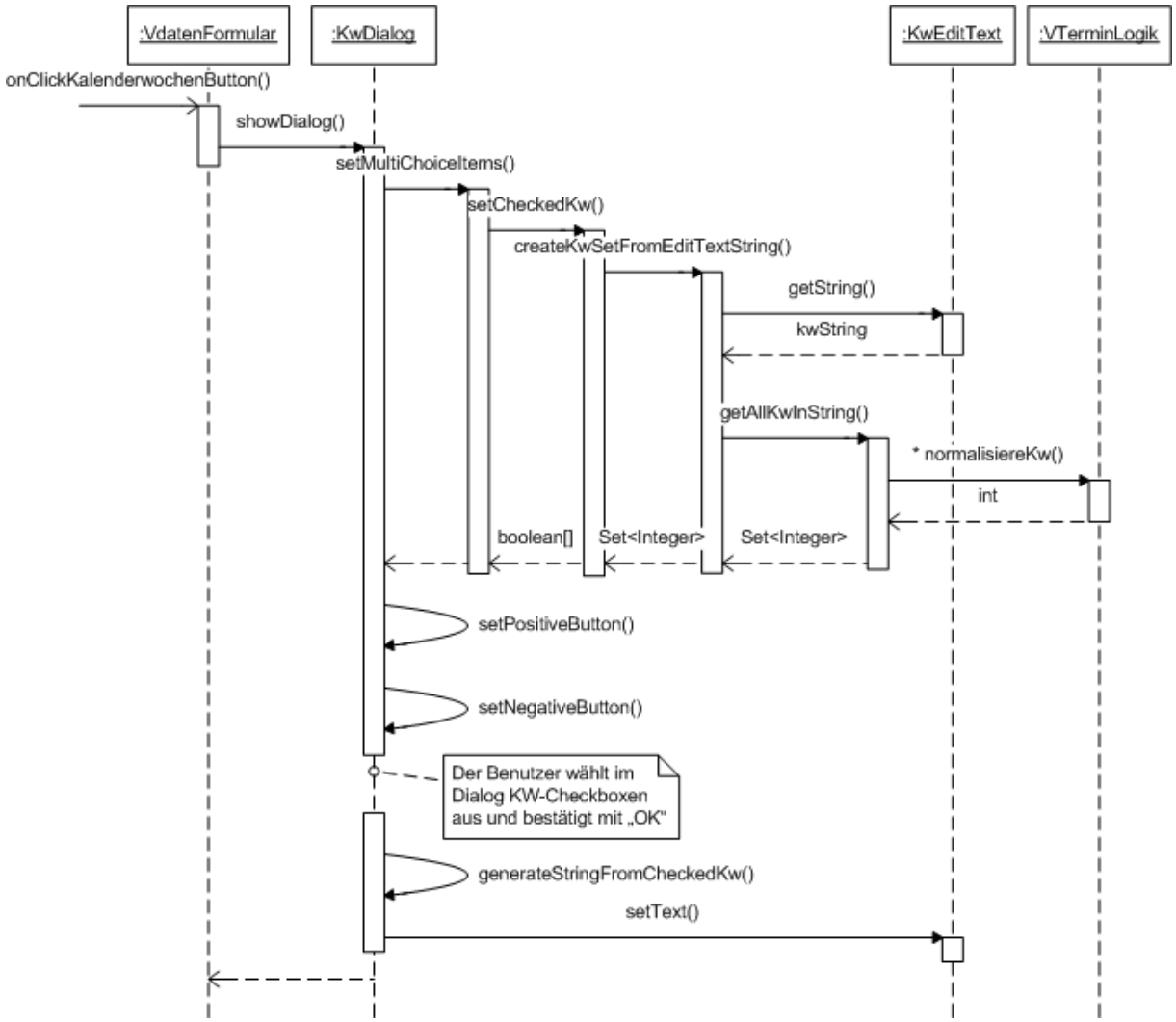

Abbildung 4.20: Sequenzdiagramm zum Anwendungsfall „Kalenderwochen eintragen“

Nachdem der Benutzer eine Auswahl getroffen und diese mit dem „OK“-Button bestätigt hat, erzeugt die Methode `generateStringFromCheckedKw()` aus den neuen Checkbox-Zuständen einen Kalenderwochen-String in der Kurzform nach Anforderung R13. Diesen String bekommt wiederum das EditText-Feld zur Direkteingabe der Kalenderwochen zugewiesen. Alle in dem Feld vorhandenen Zeichen werden dabei überschrieben.

Das nachfolgende Diagramm zeigt die beschriebenen Klassen der Formular-Komponente mit ihren Abhängigkeiten:

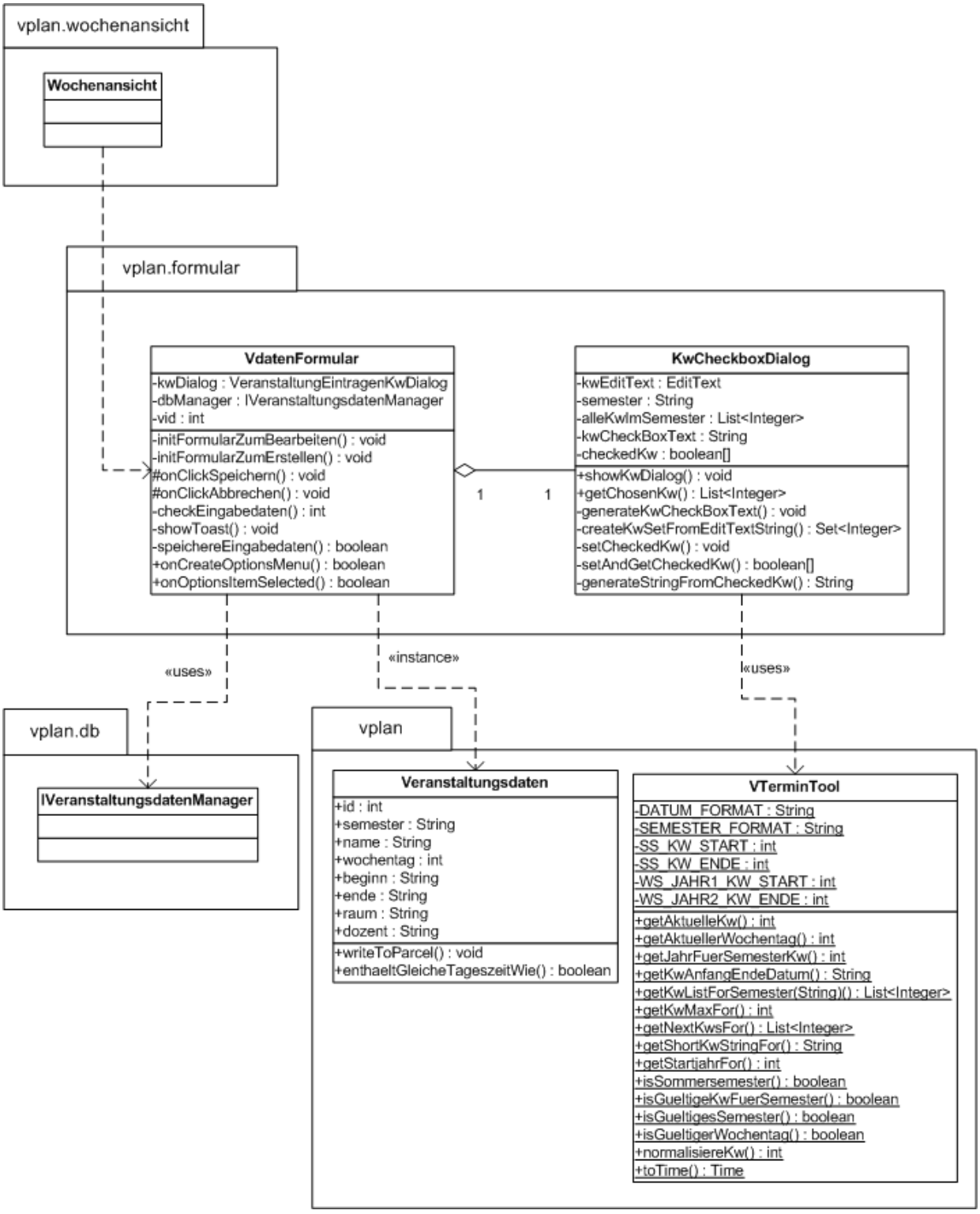

Abbildung 4.21: Die Veranstaltungsplan-Komponente „Formular“ mit ihren Abhängigkeiten

Lautlosfunktion

Die Klassen der Komponente „Lautlosfunktion“ erweitern die abstrakten Klassen des AlarmManagement. Die Klasse VLautlosModusService, leitet von der Klasse AlarmAnmeldeService ab, und der VLautlosModusWechsler erweitert die Funktionen eines RingerModeChangers:

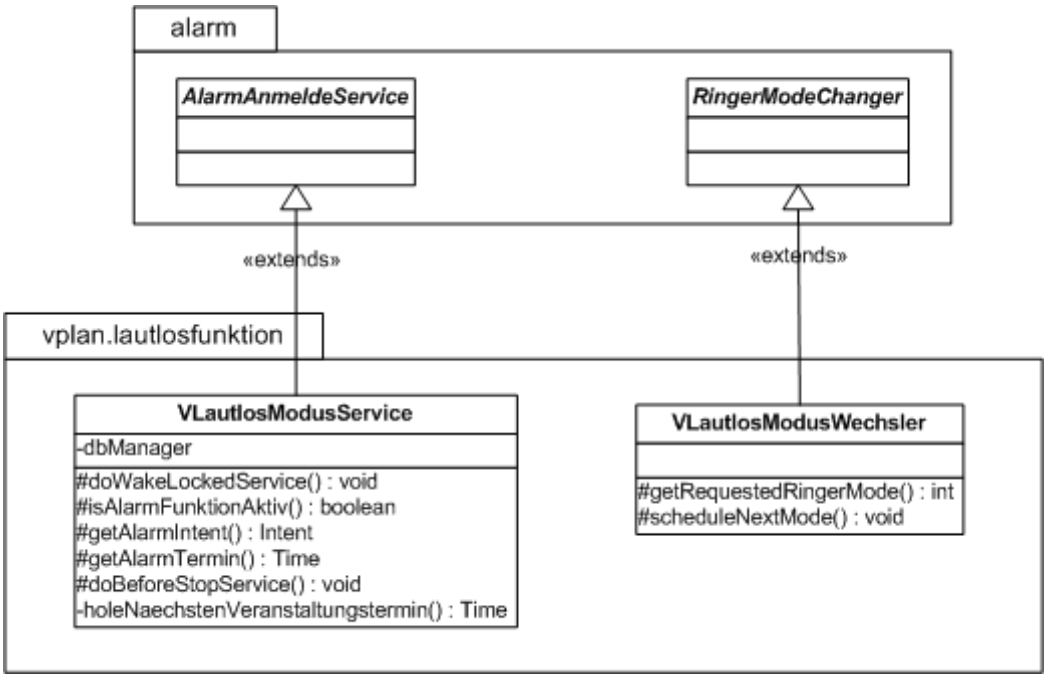

Abbildung 4.22: Klassendiagramm der Veranstaltungsplan-Komponente „Lautlosfunktion“

Mit Hilfe des VLautlosModusService lassen sich einzelne Veranstaltungstermine zur Aktivierung oder Deaktivierung des Lautlosmodus beim AlarmManager an- und abmelden. Der Start des Service erfolgt über drei verschiedene Wege:

- beim Start der Anwendung, falls die Lautlosfunktion aktiv ist (Abb. 4.23)
- über eine Checkbox-Preference in den Einstellungen, mit denen die Funktion de-/aktiviert werden kann (Abb. 4.24)
- durch den `VLautlosModusWechsler`

Die notwendigen Systemoperationen für den Anwendungsfall „Lautlostermin vormerken“ veranschaulicht Abbildung 4.25. Der Intent zum Starten des LautlosModusService enthält Informationen zum eingestellten Semester, sowie zum Zustand der Lautlosfunktion. Falls diese aktiviert ist, meldet der Service jeweils einen Lautlostermin an, andernfalls ab. Ein

Lautlostermin ist dabei entweder der Beginn der nächsten Veranstaltung, die noch nicht zu Ende ist (Aktivierung des Lautlosmodus), oder das Ende der laufenden Veranstaltung (Deaktivierung des Lautlosmodus).

Die Systemoperationen für den Anwendungsfall „Lautlostermin canceln“ sind in dem Sequenzdiagramm durch das Abmelden eines Termins beschrieben. Hierbei wird der Service aus den Einstellungen gestartet (vgl. Abb. 4.24), um die Lautlosfunktion über die Checkbox-Preference zu deaktivieren.

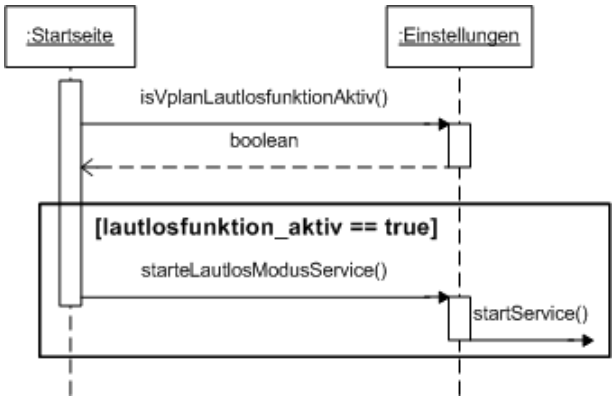

Abbildung 4.23: Sequenzdiagramm zum Start des LautlosModusService beim Start der Anwendung

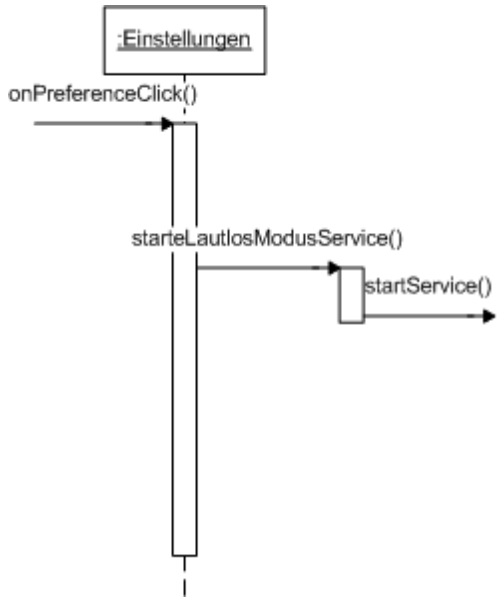

Abbildung 4.24: Sequenzdiagramm zum Start des LautlosModusService aus den Einstellungen

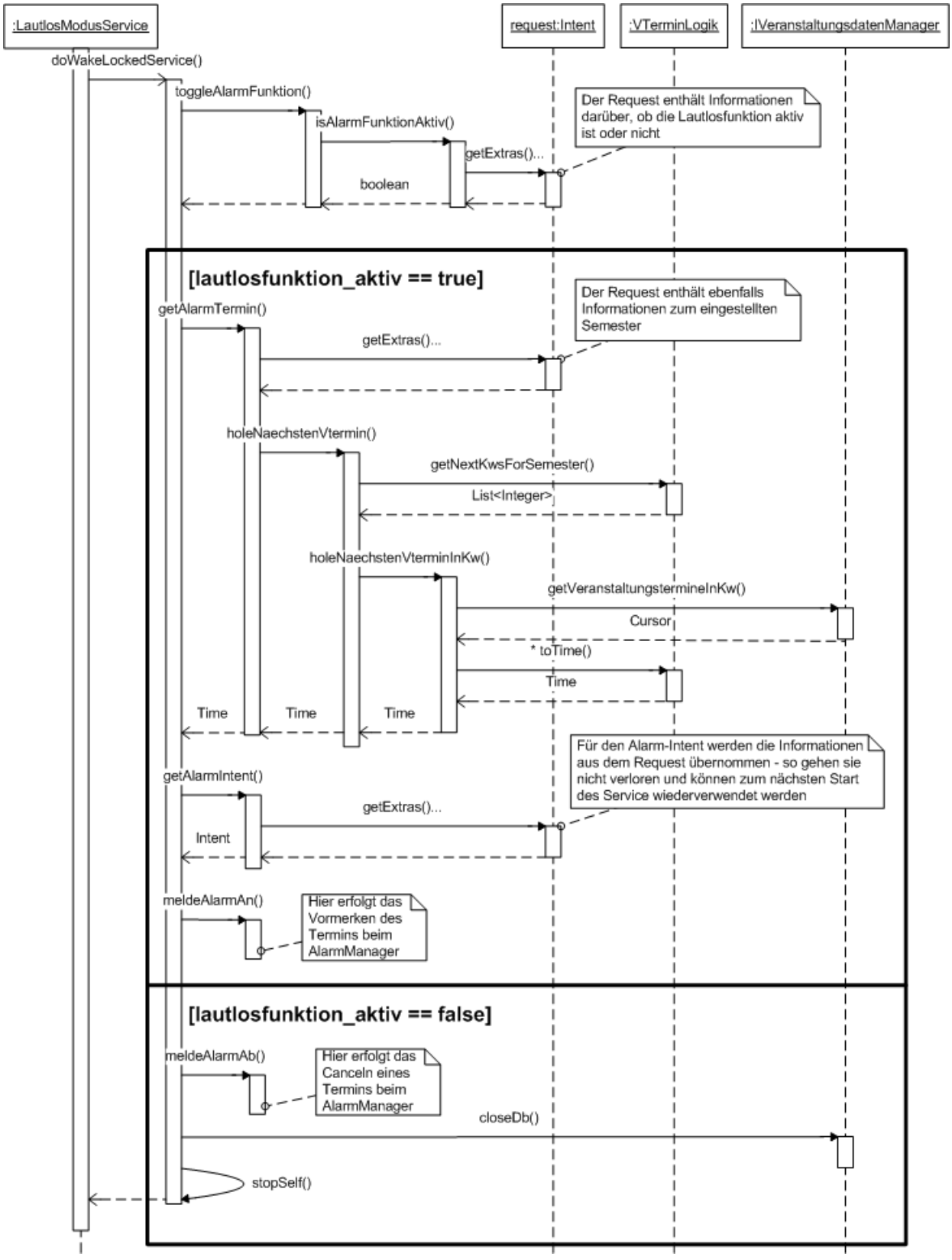

Abbildung 4.25: Sequenzdiagramm zum Anwendungsfall „Lautlostermin vormerken“

Zum Anmelden der Lautlostermine verwendet der VLautlosModusService die TerminLogik-Komponente (Tabelle 4.7), sowie eine Schnittstelle zur Persistenz (Tabelle 4.8):

`getNextKwsFor(String semester, int kwCount) : List<Integer>`
`toTime(String semester, int kw, int wochentag, String uhrzeit) : Time`

Tabelle 4.7: Die Schnittstelle der Lautlosfunktion zur TerminLogik

`getVeranstaltungstermineInKw(String semester, int kw) : Cursor`
`closeDb() : void`

Tabelle 4.8: Die Schnittstelle der Lautlosfunktion zum Persistenz-Management

Zunächst werden die nächsten drei Kalenderwochen des eingestellten Semesters über die Methode `getNextKwsFor(String semester, int kwCount)` von der TerminLogik abgefragt. Anschließend fordert der Service über die Schnittstelle zur Persistenz den nächsten Veranstaltungstermin innerhalb dieser drei Kalenderwochen an. Er ruft hierzu die Methode `getVeranstaltungstermineInKw(String semester, int kw)` falls erforderlich mit jeder der Kalenderwochen in chronologischer Reihenfolge auf. Sobald ein Termin gefunden wurde, wird die weitere Suche abgebrochen, und der Termin beim AlarmManager zum Aktivieren oder Deaktivieren des Lautlosmodus vorgemerkt. Wurde selbst in der dritten Kalenderwoche kein nächster Termin gefunden, wird auch keiner vorgemerkt. Es wird hier für jede der Wochen eine einzelne Datenbankabfrage durchgeführt, da sich eine Abfrage für mehrere Kalenderwochen aufgrund des Sprungs in den Wochennummern zum Jahreswechsel nur umständlich chronologisch sortieren ließe. Das Zeitfenster für die Abfrage von drei Wochen reicht aus, wenn davon ausgegangen werden kann, dass die Studienplaner-Anwendung einmal innerhalb von drei Wochen gestartet wird, da bei jedem Start der Anwendung mit aktivierter Lautlosfunktion erneut versucht wird, einen Termin zu ermitteln. Es ist darüber hinaus auch wenig sinnvoll, einen Termin zum Ändern des Klingelmodus vorzumerken, der über drei Wochen in der Zukunft liegt.

Das eigentliche Umstellen des Klingelmodus geschieht über den VLautlosModusWechsler. Er erhält und verarbeitet Anfragen in Form von Alarm-Intents, die vom VLautlosmodusService beim AlarmManager angemeldet, und von dort zum Alarm-Termin per Broadcast versendet wurden. Eine Anfrage enthält lediglich die Information, ob der Lautlosmodus aktiviert oder deaktiviert werden soll. Die entsprechende Auswertung eines empfangenen Intent nimmt der

VLautlosModusWechsler in der Methode `getRequestedRingerMode(Intent request)` vor. Nach erfolgter Umstellung des Klingelmodus veranlasst er mit dem Neustart des VLautlosmodusService, dass sofort der nächste Termin zum Wechsel vorgemerkt wird. Falls zuletzt in den Klingelmodus „normal“ gewechselt wurde, ist der nächste Modus „lautlos“, andernfalls ist der nächste Modus „normal“. In der Methode `scheduleNextMode(Context context, int actualRingerMode)` erhält der Intent zum Starten des VLautlosmodusService diese Information.

Um den Empfang aller Alarm-Intents unabhängig davon zu gewährleisten, ob die Studienplaner-Anwendung läuft oder nicht, wird der VLautlosModusWechsler als statischer BroadcastReceiver direkt im Manifest (vgl. 2.4.2) deklariert.

5 Implementierung und Test

Dieses Kapitel beschreibt die Realisierung der Anwendung nach dem Ansatz der testgetriebenen Entwicklung. Als Entwicklungsplattform wurde Eclipse v3.6 Helios SR 2 mit dem JDK 1.6, den Android SDK Tools r12 und dem Android Development Tools (ADT) PlugIn v12 verwendet.

5.1 Umfang der Implementierung

Alle in Kapitel 4 entworfenen Studienplaner-Komponenten werden implementiert. Auf die Implementierungsdetails geht dieses Kapitel aufgrund des Umfangs jedoch nicht ein. Zur näheren Betrachtung sei an dieser Stelle auf die beiliegende CD mit dem ausführlich dokumentierten Quellcode verwiesen. Größere Änderungen gegenüber der vorgestellten Konzeption sind nicht gegeben.

5.2 Testplan

Der Testplan zur Realisierung der Anwendung beinhaltet zwei Stufen. Die erste Stufe sieht eine entwicklungsnahe Durchführung von Komponententests vor, die in Form von automatisierten JUnit-Tests arrangiert werden. Ziel ist es dabei, die einzelnen Komponenten zunächst weitestgehend isoliert voneinander zu betrachten. Eine ergänzende Prüfung der Implementierung wird an einigen Stellen zusätzlich durch manuelle Tests erreicht.

Die zweite Stufe enthält ausschließlich nicht-automatisierte Tests, bei denen die Funktionalität des Gesamtsystems aus der Sicht des Anwenders im Vordergrund steht. Die Tests orientieren sich daher streng an den Anwendungsfällen. Mit einer Überprüfung der nichtfunktionalen Anforderungen schließen die Tests ab.

5.2.1 Zusammenfassung der Testfälle

Im Folgenden sind die Testfälle in der ersten Stufe für jeweils eine Komponente zusammenfassend beschrieben. Dabei wird unterschieden zwischen automatisiert und manuell durchzuführenden Tests. Die verwendeten Testdaten sind der detaillierteren Darstellung im Anhang zu entnehmen.

Zusammenfassung der Tests für die Komponente: PersistenzManagement **Package: de.studienplaner.test.db**		
Testziel/Testobjekt und Implementierung	**Beschreibung der Testfälle / Ablauf**	
	automatisiert	**manuell**
Datenbankschema, verwendete Statements zum Erstellen, Löschen und Modifizieren der Tabellen (Klassen `StudienplanerDatenbank`, `Tabelle_Veranstaltungen`, `Tabelle_V_in_Kw`)	Aufruf der in den Klassen als Konstanten definierten Statements in einer Testkette, Überprüfen des Datenbank-Inhaltes für jeden Testfall	Test, ob Constraint-verletzende DB-Anfragen scheitern (per sqlite-shell), Durchführen von „null-INSERTs“ für alle „NOT NULL“-Constraints
Schnittstelle IVeranstaltungsdatenManager (Klasse `DatenbankManager`)	Aufruf der Schnittstellen-Methoden und Überprüfen des Datenbank-Inhaltes	-

Tabelle 5.1: Test der Komponente „PersistenzManagement“

Zusammenfassung der Tests für die Komponente: Startmenü **Package: de.studienplaner.test**		
Testziel/Testobjekt und Implementierung	**Beschreibung der Testfälle / Ablauf**	
	automatisiert	**manuell**
Die Activity / Bildschirmseite mit dem Startmenü (Klasse `Startseite`)	Überprüfen der verwendeten Ressourcen, der Bildschirmelemente (Menü-Buttons) und des Optionsmenüs	-

Tabelle 5.2: Test der Komponente „Startmenü“

Zusammenfassung der Tests für die Komponente: Einstellungen **Package: de.studienplaner.test.prefs**		
Testziel/Testobjekt und Implementierung	**Beschreibung der Testfälle / Ablauf**	
	automatisiert	**manuell**
Activity / Bildschirmseite mit den Preferences, Zugriff auf die gemeinsamen Einstellungen der Anwendung (Klasse `Einstellungen`)	Testen, ob die Defaults beim ersten Start der Anwendung korrekt persistiert werden Testen, ob alle Einstellungen über die Schnittstelle korrekt ausgelesen werden	Laden und Speichern einiger zufälliger Werte mit Hilfe der Preference-Views Überprüfen, ob die Einstellungen nach einem Neustart der Anwendung korrekt geladen werden Testen der Lautlosfunktion-Preference (Anmelden der Lautlostermine durch VLautlosModusService) Testen der Semester-Preference (Abmelden der Lautlostermine durch VLautlosModusService)

Tabelle 5.3: Test der Komponente „Einstellungen“

Zusammenfassung der Tests für die Komponente: TerminLogik (Veranstaltungsplan) **Package: de.studienplaner.test.vplan**		
Testziel/Testobjekt und Implementierung	**Beschreibung der Testfälle / Ablauf**	
	automatisiert	**manuell**
TerminLogik (Klasse `VTerminTool`)	Aufruf der einzelnen Methoden mit unterschiedlichen Eingaben und Überprüfen der Rückgabewerte	-

Tabelle 5.4: Test der Komponente „TerminLogik“

Zusammenfassung der Tests für die Komponente: Tagesplan (Veranstaltungsplan) **Package: de.studienplaner.test.vplan.tagesplan**		
Testziel/Testobjekt und Implementierung	**Beschreibung der Testfälle / Ablauf**	
	automatisiert	**manuell**
Das Plan-Layout (Klassen: `PlanLayout` und `CellLayoutParams`)	Testen der Methoden zum Messen der Breite und Höhe des Plans, zum Positionieren der Child-Views, sowie zum Setzen und Validieren der Layout-Parameter.	-
Der Tagesplan (Klassen: `Tagesplan`, `TagesplanZeile`, `Veranstaltung` und `Veranstaltungsdaten`)	Testen der Konstruktion von Tagesplänen aus zulässigen und unzulässigen Werten, Überprüfen, ob die TagesplanZeilen jeweils mit der korrekten Uhrzeit erstellt und Veranstaltungen korrekt im Plan eingetragen werden	Eintragen unterschiedlicher Veranstaltungen, Überprüfen der Größen und Positionen im Tagesplan, Überprüfen von Gitterlinien und Text in den geforderten Geräteauflösungen

Tabelle 5.5: Test der Komponente „Tagesplan“

Zusammenfassung der Tests für die Komponente: Wochenansicht (Veranstaltungsplan) **Package: de.studienplaner.test.vplan.wochenansicht**		
Testziel/Testobjekt und Implementierung	**Beschreibung der Testfälle / Ablauf**	
	automatisiert	**manuell**
Die Bildschirmseite mit der Wochenansicht (Klasse `Wochenansicht`)	Testen der verwendeten XML-Ressourcen, des Bildschirmtitels, der Methoden zum Initialisieren der Tabs und der Tab-Inhalte, Testen des Optionsmenüs	-
Die Initialisierung der Wochenansicht (Klasse `WochenansichtInitTask`)	Testen, ob der Task startbar ist und die Wochenansicht mit den korrekten Testdaten aktualisiert	-

Tabelle 5.6: Test der Komponente „Wochenansicht“

Zusammenfassung der Tests für die Komponente: Formular (Veranstaltungsplan) **Package: de.studienplaner.test.vplan.formular**		
Testziel/Testobjekt und Implementierung	**Beschreibung der Testfälle / Ablauf**	
	automatisiert	**manuell**
Veranstaltungsdaten-Formular (Klasse `VdatenFormular`)	Starten des Formulars mit verschiedenen Intents und überprüfen, ob der Bildschirmtitel und die Formular-elemente korrekt initialisiert wurden	siehe Tabelle D.10 im Anhang
Kw-Checkbox-Dialog (Klasse `KwCheckboxDialog`)	Eintragen von verschiedenen Kalenderwochen-Strings in das KwEditText-Feld, öffnen des KwCheckbox-Dialoges und überprüfen, ob alle eingegebenen, für das Semester zulässigen Kalenderwochen ausgewählt sind	siehe Tabelle D.10 im Anhang

Tabelle 5.7: Test der Komponente „Formular“

Zusammenfassung der Tests für die Komponente: AlarmManagement **Package: de.studienplaner.test.alarm**	
Testziel/Testobjekt und Implementierung	**Beschreibung der Testfälle / Ablauf**
(Klassen `WakeLockedService`, `AlarmAnmeldeService`, `RingerModeChanger`)	Starten und Überprüfen jeweils einer konkreten Implementierung

Tabelle 5.8: Test der Komponente „AlarmManagement“

Zusammenfassung der Tests für die Komponente: Lautlosfunktion (Veranstaltungsplan) **Package: de.studienplaner.test.vplan.lautlosfunktion**		
Testziel/Testobjekt und Implementierung	**Beschreibung der Testfälle / Ablauf**	
	automatisiert	**manuell**
Klasse `VLautlosModusWechsler`	Das Einstellen und Vormerken aller möglichen Klingelmodi überprüfen	De-/Aktivieren der Lautlosfunktion für verschiedene Termine
Klasse `VLautlosModusService`	Überprüfen, ob die Lautlosfunktion korrekt aktiviert und deaktiviert wird	De-/Aktivieren der Lautlosfunktion für verschiedene Termine

Tabelle 5.9: Test der Komponente „Lautlosfunktion“

5.3 Testdurchführung und -auswertung

5.3.1 Erste Teststufe: Komponententests

Wie bereits im Abschnitt 5.1 erwähnt, werden die Tests der ersten Stufe entwicklungsnah durchgeführt und auswertet. Dabei werden die bestehenden Abhängigkeiten und Beziehungen zu Nachbarkomponenten durch den Einsatz geeigneter Mock-Objekte simuliert. Die Implementierung der in den Anforderungen festgelegten Funktionalitäten erfolgt nach der jeweils in einem Test aufgestellten Definition des erwarteten Ein-/Ausgabe-Verhaltens für eine oder mehrere Methoden. An einigen Stellen wird auf den Einsatz aufwendiger Mock-Objekte verzichtet und bereits getesteter Code verwendet. Auf diese Weise lässt sich auch die Integration der neuen Komponenten in das bestehende System schrittweise sicherstellen. Eine iterative Anpassung der Implementierung geschieht solange, bis alle im Testplan vorgesehenen Tests bestanden sind.

5.3.2 Zweite Teststufe: Systemtests

Bei den anwendungsfallorientierten Tests in der zweiten Stufe wird das Verhalten der Anwendung als Blackbox über mehrere Tage und auf mehreren verschiedenen Geräten untersucht. Dabei macht der Tester bevorzugt auch ungewöhnliche Eingaben, um Ausnahmesituationen herbeizuführen. Neben einer fehlerfreien Funktionalität sind zum Beispiel die Antwortzeiten, die Darstellung in verschiedenen Bildschirmauflösungen und -größen, sowie der Umgang mit asynchron auftretenden Ereignissen – wie eingehenden Anrufen oder Rotationen zwischen Portrait- und Landscape- Modus – von besonderer Bedeutung. Bei fehlerhaftem oder unerwünschtem Verhalten werden der Testkontext, der Status der Anwendung und die getätigten Eingaben protokolliert. Die gesammelten Ergebnisse fließen in weitere Implementierungsphasen ein, denen jeweils neue Tests folgen, bis eine weitestgehend fehlerfreie Funktionalität festzustellen ist.

6 Zusammenfassung

In dieser Arbeit wurde ein System vorgestellt, das eine strukturierte, studienbegleitende Terminplanung ermöglicht. Dabei wurde zunächst gezeigt, dass die Semesterzeiten an den deutschen Hochschulen nicht einheitlich sind und das System deshalb unterschiedliche Planungszeiträume berücksichtigen muss. Weiterhin ließ sich feststellen, dass eine Planung nach Kalenderwochen für Studienveranstaltungen sehr gut eignet ist, da es sich hier generell um Wiederholungstermine handelt. Eine solide Grundlage zur Planung von Serienterminen auf der Basis von Kalenderwochen ist sowohl durch die international gebräuchliche Kalenderspezifikation ISO 8601, als auch durch den iCalendar-Standard gegeben. Die sich darauf stützenden, weitverbreiteten Anwendungen wie der Outlook- oder Google- Kalender sind dazu jedoch nur bedingt geeignet, da sie nicht von vornherein die Möglichkeit bieten, Ausnahmen zu einer Terminserie festzulegen.

Aus dem gleichen Grund lässt sich auch der Android-Kalender nur umständlich zur Planung von Studienveranstaltungen einsetzen. Verschiedene Serientermine sind dort zudem nicht übersichtlich genug darstellbar. Es konnte jedoch gezeigt werden, dass die Kalenderfunktion einen hohen Stellenwert bei der Nutzung von Mobiltelefonen einnimmt. Dies lässt darauf schließen, dass eine App zur Studienplanung auf Android-Handys durchaus Verwendung finden kann.

Im weiteren Verlauf der Arbeit wurden die Entwicklungsplattform und einige verwendete Komponenten des Android-Frameworks beschrieben. Nach einer Anforderungsanalyse sind im Anschluss die notwendigen Funktionen und Eigenschaften des Systems katalogisiert und die Anwendungsfälle formuliert worden. Schließlich folgte der auf die Systemoperationen gestützte Entwurf. Mit der Veranstaltungsplan-Komponente ist dabei eine sinnvolle Alternative, aber auch eine Ergänzung zu vorhandenen Kalenderanwendungen entstanden. Im fünften Kapitel wurde aus allen entworfenen Komponenten erfolgreich ein testbarer Prototyp implementiert.

6.1 Erweiterungen

Das System bietet zahlreiche Erweiterungsmöglichkeiten. Eine bereits angesprochene, denkbare Zusatzfunktion ist der Import und Export von iCalendar-Daten. Jede Veranstaltung kann hierfür als ein iCalendar-VEvent behandelt werden, bei dem die wöchentliche Wiederholungsregel in der RRULE- und die Ausnahmen in der EXDATE- Eigenschaft (vgl. Abschnitt 2.2.2) gespeichert sind. Damit lassen sich die Veranstaltungsdaten bei Bedarf mit den Termindaten aus anderen Systemen wie dem Outlook- oder Google-Kalender synchronisieren. Für den Abgleich mit dem Android-Kalender steht aus dem Framework ein spezieller Content-Provider zur Verfügung, der unter anderem das Einfügen neuer Serientermine im iCalendar-Format erlaubt.

Eine andere Erweiterungsmöglichkeit ist eine Erinnerungsfunktion. Ähnlich wie die Lautlosfunktion würde diese auf die AlarmManagement-Komponente aufbauen, um Termine beim Android-AlarmManager an- und abzumelden.

Weiterhin kann die Anwendung zum Beispiel durch eine Druckfunktion für den Veranstaltungsplan, einen Speiseplan oder eine Umgebungskarte an zusätzlichem, praktischem Nutzwert gewinnen und damit als hilfreiches Allround-Tool für die allgemeine studentische Planung auf einer mobilen Plattform eingesetzt werden.

Die verwendeten Textelemente und große Teile des Layouts werden aus XML-Dateien geladen und sind daher nachträglich leicht anpassbar. Es kann zum Beispiel eine weitere Sprache hinzugefügt oder für die Veranstaltungsinformationen im Plan eine andere Darstellung gewählt werden.

Abgesehen von der Semesterbezeichnung lässt sich das System auf unterschiedliche Hochschulen und andere Anwendungsgebiete übertragen. Durch Anpassung der Kalenderwochengrenzen in der Terminlogik kann es beispielsweise auch zur Schulplanung eingesetzt werden.

Alle in den Anforderungen aufgeführten Bildschirmauflösungen und -größen werden grundsätzlich unterstützt, über die Gerätekompatibilität kann jedoch derzeit noch keine detaillierte Aussage getroffen werden. Tests mit dem SDK-Emulator, einem HTC Desire, einem Motorola Milestone und einem Samsung Galaxy SII verliefen bislang erfolgreich.

6.2 Fazit

Die Android-Plattform eignet sich hervorragend zur Entwicklung erweiterbarer Anwendungen wie dem Studienplaner. Das Intent-Konzept erleichtert die Realisierung einer losen Kopplung zwischen Komponenten, was sich in dieser Arbeit als sehr unterstützend herausgestellt hat. Entwickler für Android-Anwendungen stehen jedoch vor besonderen Herausforderungen. Durch eine immer größere Gerätevielfalt und kurze Zeitabstände zwischen den Veröffentlichungen neuer Betriebssystemversionen ergeben sich schnell Kompatibilitätsprobleme. Daneben muss schon bei der Implementierung einigen Aspekten spezielle Beachtung geschenkt werden. Die Intent-basierte Kommunikation macht es zum Beispiel erforderlich, dass die zwischen den Komponenten ausgetauschten Daten serialisierbar sind. Ein weiteres Beispiel ist der Lebenszyklus einer Activity. Bei einem asynchronen Ereignis, etwa bei Drehung des Bildschirmes, können Activities zerstört werden, was zu einem Datenverlust führen kann. Diese und weitere Gegebenheiten müssen bei der Entwicklung von Android-Anwendungen grundsätzlich immer berücksichtigt werden.

Umso hilfreicher ist es, dass die API sowohl ausführlich, als auch verständlich dokumentiert ist, und dass das im SDK integrierte Testframework mittlerweile vielfältige Möglichkeiten bietet, automatisierte Tests zu realisieren. Dazu gehören die Simulation von Touch-Events, das Abfangen versendeter Intents und der Einsatz nützlicher Mock-Klassen. Etwas schade ist, dass das Android-Testframework noch auf JUnit 3 aufbaut. Es werden hier zum Beispiel keine parametrisierten Tests unterstützt. Allerdings gibt es auch einige alternative, ausgereifte Frameworks, auf die sich zurückgreifen lässt. Im Allgemeinen ist der erhöhte Testaufwand mit den verfügbaren Mitteln durchaus beherrschbar.

Für den Nutzer erschließt der Entwickler durch den erhöhten Testaufwand natürlich auch vielseitigere Einsatzmöglichkeiten. So ist es denkbar, den Studienplaner ebenfalls für die Verwendung auf Tablets zur Verfügung zu stellen. Zusammenfassend lässt sich sagen, dass in dieser Arbeit eine zweckdienliche Anwendung entstanden ist, für die sich die Auswahl der Android-Plattform lohnt. Durch zusätzliche Erweiterungen lässt sie sich optimal in den Studienalltag integrieren.

A Literatur- und Quellenverzeichnis

[1] Google Inc. Android Developers – *The Developer's Guide* http://developer.android.com/guide/index.html, Letzter Zugriff: 19.09.2011

[2] Arno Becker, Marcus Pant: *Android 2: Grundlagen und Programmierung*, dPunkt-Verlag, Heidelberg, 2010

[3] Heiko Mosemann, Matthias Kose: *Android. Anwendungen für das Handy-Betriebssystem erfolgreich programmieren*, Hanser Fachbuch, München, 2009

[4] Andreas Spillner, Tilo Linz: *Basiswissen Softwaretest*, dPunkt-Verlag, Heidelberg, 2005

[5] Andreas Spillner, Thomas Roßner, Mario Winter, Tilo Linz: *Praxiswissen Softwaretest, Testmanagement*, dPunkt-Verlag, Heidelberg, 2008

[6] Hochschulrektorenkonferenz – *Empfehlung zur Harmonisierung der Semester- und Vorlesungszeiten an deutschen Hochschulen im Europäischen Hochschulraum* http://www.hrk.de/de/download/dateien/Beschluss_Semesterzeiten.pdf, Letzter Zugriff: 18.09.2011

[7] Uni Hamburg – *Semester- und Vorlesungszeiten* http://www.verwaltung.uni-hamburg.de/onTEAM_extern/0__99715661971459__9971 5661971461.html, Letzter Zugriff: 18.09.2011

[8] HAW Hamburg – *Semester- und Vorlesungszeiten* http://www.informatik.haw-hamburg.de/vorlesungszeiten.html, Letzter Zugriff: 18.09.2011

[9] Uni Hildesheim – *Semester- und Vorlesungszeiten* http://www.uni-hildesheim.de/index.php?id=77, Letzter Zugriff: 18.09.2011

[10] Uni Kiel – *Semester- und Vorlesungszeiten*
http://www.studservice.uni-kiel.de/termine.shtml, Letzter Zugriff: 08.03.2011

[11] Fachhochschule Kiel – *Semester- und Vorlesungszeiten*
http://fh-kiel.de/index.php?id=2277, Letzter Zugriff: 18.09.2011

[12] TU München – *Semester- und Vorlesungszeiten*
http://portal.mytum.de/studium/formalia/wintersemester_2011-12,
Letzter Zugriff: 18.09.2011

[13] Uni Köln – *Semestertermine*
http://www.wiso.uni-koeln.de/home/inhalt.asp?l=stud&m=s_termine,
Letzter Zugriff: 18.09.2011

[14] FH Köln – *Semestertermine*
http://www.studium.fh-koeln.de/service/u/01369.php, Letzter Zugriff: 18.09.2011

[15] TU Dresden – *Studienjahresablauf*
http://tu-dresden.de/studium/organisation/studienjahresablauf,
Letzter Zugriff: 18.09.2011

[16] Uni Mannheim – *Semester- und Vorlesungszeiten*
http://www.uni-mannheim.de/1/studium/aktuelles_termine/semesterzeiten/index.html,
Letzter Zugriff: 18.09.2011

[17] Wikipedia – *Semester*
http://de.wikipedia.org/wiki/Semester, Letzter Zugriff: 18.09.2011

[18] HAW Hamburg, Department Informatik – *Veranstaltungspläne*
http://www.informatik.haw-hamburg.de/veranstaltungsplaene.html,
Letzter Zugriff: 18.09.2011

[19] HAW Hamburg, Department Design – *Studienpläne*
http://www.design.haw-hamburg.de/studium/studienplaene.html,
Letzter Zugriff: 18.09.2011

[20] International Organization for Standardization – ISO 8601
http://www.iso.org/iso/support/faqs/faqs_widely_used_standards/widely_used_standards_other/date_and_time_format.htm, Letzter Zugriff: 18.09.2011

[21] RFC Editors – *Internet Calendaring and Scheduling Core Object Specification*
http://tools.ietf.org/html/rfc5545, Letzter Zugriff: 18.09.2011

[22] Bitkom – Bundesverband Informationswirtschaft, Telekommunikation und neue Medien e.V. – *Top10 der Handy-Funktionen*
http://www.bitkom.org/de/markt_statistik/64046_64889.aspx,
Letzter Zugriff: 18.09.2011

[23] Bitkom – Bundesverband Informationswirtschaft, Telekommunikation und neue Medien e.V. – *Presseinformation Smartphone Nutzer*
http://www.bitkom.org/files/documents/BITKOM_Presseinfo_Smartphones_14_10_2010.pdf, Letzter Zugriff: 18.09.2011

[24] Google Inc. – *Google Kalender Hilfe, Import und Export*
http://www.google.com/support/calendar/bin/topic.py?topic=15285,
Letzter Zugriff: 18.09.2011

[25] Google Inc. Android Developers – *The Developer's Guide, What is Android?*
http://developer.android.com/guide/basics/what-is-android.html,
Letzter Zugriff: 18.09.2011

[26] SQLite Developers – *SQLite Documentation, About SQLite*
http://www.sqlite.org/about.html, Letzter Zugriff: 18.09.2011

B Tabellen

C Abbildungen

D Anhang 1: Testfälle

Test der Datenbank

Die einzelnen Testfälle zum Überprüfen der Datenbank sind voneinander abhängig und bilden daher eine Testkette, die nach der Erzeugung einer Test-Datenbank in jedem Schritt für beide Tabellen durchlaufen wird. Vor jedem Testfall wird die Datenbank geöffnet, danach wieder geschlossen.

TF	Beschreibung	Voraussetzungen / Vorbedingungen	Soll- Ergebnis / Nachbedingungen
1	Erstellen der beiden Tabellen per CREATE	Die beiden Tabellen sind nicht in der Datenbank vorhanden (Löschen per DROP)	Die beiden Tabellen sind in der Datenbank vorhanden (SELECT auf sqlite_master für beide Tabellennamen)
2	Einfügen eines Testdatensatzes per INSERT	Testdatensatz ist nicht in der Datenbank vorhanden (Abfrage aller Datensätze per SELECT gibt 0)	Abfrage der vorhandenen Datensätze per SELECT gibt 1, eingefügte Werte sind in den entsprechenden Tabellenspalten vorhanden
3	Aktualisieren des Datensatzes per UPDATE	Testdatensatz ist in der Datenbank vorhanden (Abfrage aller Datensätze per SELECT gibt 1)	Der Datensatz liegt in der Datenbank verändert vor
4	Löschen des Datensatzes per DELETE	Testdatensatz ist in der Datenbank vorhanden (Abfrage aller Datensätze per SELECT gibt 1)	Abfrage der vorhandenen Datensätze per SELECT gibt 0
5	Löschen der beiden Tabellen per DROP	Die beiden Tabellen sind in der Datenbank vorhanden	Die beiden Tabellen sind nicht mehr in der Datenbank vorhanden

Tabelle D.1: Automatisierter Test der Datenbank

Test des Datenbank-Managers

Der Zugriff auf die Datenbank per DatenbankManager erfolgt ebenfalls in einer Testkette. Um die Abfragen zu prüfen, werden zuvor einige Testdatensätze in eine neu erzeugte Datenbank geschrieben. Vor jedem Testfall wird die Datenbank geöffnet, danach wieder geschlossen.

TF	Beschreibung	Voraussetzungen / Vorbedingungen	Soll- Ergebnis / Nachbedingungen
1	Speichern einer Veranstaltung über den DatenbankManager	-	Ein SELECT mit den Testdaten liefert die gleichen Veranstaltungsdaten
2	Ändern der Veranstaltung aus TF1 (alle Spaltenwerte) über den DatenbankManager	Der Testdatensatz aus TF1 existiert in der Datenbank	Ein SELECT mit der ID aus TF1 liefert einen korrekt geänderten Datensatz
3	Löschen einer Veranstaltung aus den Testdaten über den DatenbankManager	Die Testdatensätze aus den Testdaten existieren in der Datenbank	Ein SELECT auf die ID der gelöschten Veranstaltung liefert keinen Datensatz
4	Aufruf der Methode `getVeranstaltungenZu(semester,wochentag)`	Die Testdatensätze aus den Testdaten existieren in der Datenbank	Der Aufruf liefert die gleichen Veranstaltungen wie in den Testdaten

5	Aufruf der Methode `getAlleKwZuVeranstaltung(id)`	Der Testdatensatz aus TF1 existiert in der Datenbank	Ein SELECT mit der ID aus TF1 liefert die korrekten Kalenderwochen
6	Aufruf der Methode `getVeranstaltungsterminenInKw(semester,kw)`	Die Testdatensätze aus den Testdaten existieren in der Datenbank	Der Aufruf liefert die gleichen Termine wie in den Testdaten
7	Schließen der Datenbank	-	Ein (beliebiger) SELECT schlägt fehl

Tabelle D.2: Automatisierter Test des Datenbank-Managers

Test des Startmenüs

Zur Überprüfung des Optionsmenüs auf der Startseite werden Key-Events injiziert. Daher muss darauf geachtet werden, dass der Screen-Lock des Testgerätes bei der Ausführung deaktiviert ist. Andernfalls bricht der Test mit einer SecurityException ab.

TF	Beschreibung	Voraussetzungen / Vorbedingungen	Soll- Ergebnis / Nachbedingungen
1	Abfragen aller im Zielkontext zu verwendenden String-Ressourcen per `getString(ID)`	-	Alle Ressourcen sind vorhanden, es wird keine NotFoundException geworfen
2	Abfragen des Menü-Buttons zum Starten des Veranstaltungsplans per `findViewById(ID)`, Senden eines Key-Event an den Button und Überprüfen der gestarteten Activity per `ActivityMonitor` (Timeout: 5 Sekunden)	Die Activity `Startseite` wurde geladen	Über ihre Ressource-ID kann die Button-View gefunden werden, das Key-Event löst den Start der `Wochenansicht`-Activity aus
3	Aufruf des Optionsmenüs per Key-Event, Auswahl der Option „Einstellungen“ per Key-Event, Überprüfen der gestarteten Activity per `ActivityMonitor` (Timeout: 5 Sekunden)	Die Activity `Startseite` wurde geladen	Die Key-Events lösen den Start der Activity `Einstellungen` aus

Tabelle D.3: Automatisierter Test des Startmenüs

Test der Einstellungen

Zum Test der Einstellungen-Activity werden vor jedem Testfall alle persistenten, gemeinsamen Anwendungseinstellungen gelöscht. Nach Abschluss aller Tests werden sie ebenfalls wieder gelöscht.

TF	Beschreibung	Voraussetzungen / Vorbedingungen	Soll- Ergebnis / Nachbedingungen
1	Abfrage der verwendenden Preference-Ressourcen per `findViewById(ID)`	Die Activity `Einstellungen` wurde gestartet	Alle Preference-Ressourcen können mit ihrem jeweiligen Key gefunden werden
2	Speichern der Defaults und Auslesen der Werte über die Komponenten-Schnittstelle	Die Einstellungen sind noch nicht vorhanden	Alle Einstellungen wurden erfolgreich persistiert, die ausgelesenen Werte entsprechen den Defaults (als Konstanten in der Klasse `Einstellungen` definiert)

3	Simulieren eines Touch-Events zum Aktivieren/Deaktivieren der Lautlosfunktion	Die Activity `Einstellungen` wurde gestartet	Der `VLautlosModusService` wird ohne Exception bei Aktivierung gestartet und bei Deaktivierung gestoppt
4	Simulieren eines Touch-Events zum Ändern des Semesters	Die Activity `Einstellungen` wurde gestartet	Der `VLautlosModusService` wird ohne Exception gestoppt

Tabelle D.4: Automatisierter Test der Einstellungen

Test der TerminLogik

Alle Methoden werden mit unterschiedlichen Eingabewerten getestet. Um geeignete Werte festzulegen, ist es erforderlich, die zu jedem Parameter vorliegenden Äquivalenzklassen zu analysieren. Diese Analysen sind aufgrund des Umfangs hier nicht mit aufgeführt. Das Ergebnis ist jeweils eine Auswahl von Repräsentanten aller gültigen und ungültigen Äquivalenzklassen.

Test der Methode	Parameter	Eingabewerte	Soll- Ergebnis
checkSemester (String semester)	semester	zulässig: "WS99","SS99","SS13","WS14" unzulässig: "W11","WS09","HS12","SS100",“SS4“	IllegalArgumentException bei unzulässigem Semester
checkWochentag (int wochentag)	wochentag	zulässig: 1,2,3,4,5,6 unzulässig: 0,7	IllegalArgumentException bei unzulässigem Wochentag
checkUhrzeit (String uhrzeit)	uhrzeit	zulässig: "00:00","12:00","16:27","23:59" unzulässig: "24:00","1200","12:0","1:15","10:61"	IllegalArgumentException bei unzulässiger Uhrzeit
getAktuelleKw()	-	-	aktuelle Kalenderwoche nach ISO8601
getAktuellerWochentag()	-	-	aktueller Wochentag, mit 0=So bis 6=Sa
getJahrFuerSemesterKw (String semester, int kw)	semester	"SS12"	2012, 2012, 2012, 0, 0, 0
	kw	SS_KwStart, 30, SS_KwEnde, SS_KwStart-1, SS_KwEnde+1, 0	
getJahrFuerSemesterKw (String semester, int kw)	semester	"WS12"	2012, 2013, 2013, 0, 0, 0
	kw	WS_KwStart, 1, WS_KwEnde, WS_KwStart-1, 54, WS_KwEnde+1	
getJahrFuerSemesterKw (String semester, int kw)	semester	unzulässig: "","SS123","HS12"	IllegalArgumentException
	kw	15	
getKwAnfangEndeDatum (String semester, int kw)	semester	"SS12"	Die jeweilige Woche im Format "TT.MM.JJ - TT.MM.JJ", oder null bei unzulässigen Werten
	kw	zulässig: SS_KwStart, SS_KwEnde, 30 unzulässig: SS_KwStart-1, SS_KwEnde+1, 53, 0	
getKwAnfangEndeDatum (String semester, int kw)	semester	"WS12"	Die jeweilige Woche im Format "TT.MM.JJ - TT.MM.JJ", oder null bei unzulässigen Werten
	kw	zulässig: WS_KwStart, WS_KwEnde, 1, 52 unzulässig: WS_KwStart-1, WS_KwEnde+1, 53, 0	

getKwListForSemester (String semester)	semester	unzulässig: "","SS123","HS12"	IllegalArgumentException
getKwListForSemester (String semester)	semester	zulässig: "SS12","SS15","WS12","WS15"	Erste Kw: SS_KwStart / WS_KwStart Letzte Kw: SS_KwEnde / WS_KwEnde Größe der Kw-Liste: 41, 41, 43, 44
getKwMaxFor (int jahr)	jahr	Jahre mit 53 Kw: 2009, 2015, 2020, 2026 Jahre mit 52 Kw: 2011, 2012, 2013, 2014	53 bzw. 52
getNextKwsFor (String semester, int kwCount)	semester	unzulässig: "","SS123","HS12" weit in der Zukunft: "SS47" in der Vergangenheit: "SS09"	IllegalArgumentException bei unzulässigem Semester, leere Liste bei nicht aktuellem Semester oder kwCount < 1
	kwCount	-1, 0, 3	
getNextKwsFor (String semester, int kwCount)	semester	aktuelles Semester	(manuelle Überprüfung anhand von Ausgaben)
	kwCount	3, unterschiedliche Werte	
getShortKwStringFor (String semester, List<Integer> kalenderwochen)	semester	"SS12"	„[SS_ KwStart], [SS_ KwStart+1], [SS_KwEnde-1], [SS_ KwEnde]“
	kalenderwochen	SS_KwStart-1, SS_ KwStart, SS_ KwStart+1, SS_KwEnde-1, SS_ KwEnde, SS_ KwEnde+1, 0, -5, 300	
getShortKwStringFor (String semester, List<Integer> kalenderwochen)	semester	"SS12"	"10-13,22" "10-13,22" "50"
	kalenderwochen	0,9,10,11,12,13,22 0,9,10,11,13,22,12 1,2,50,51,52,3	
getShortKwStringFor (String semester, List<Integer> kalenderwochen)	semester	"WS12"	„[WS_ KwStart], [WS_ KwStart+1], [WS_KwEnde-1], [WS_ KwEnde]“
	kalenderwochen	WS_KwStart-1, WS_ KwStart, WS_ KwStart+1, WS_KwEnde-1, WS_ KwEnde, WS_ KwEnde+1, 0,-5, 300	
getShortKwStringFor (String semester, List<Integer> kalenderwochen)	semester	"WS12"	"1-3,6-10" "1-3,6,7,9,10" "50-52,1-3"
	kalenderwochen	1,2,3,6,7,8,9,10 1,2,3,6,7,9,10 1,2,50,51,52,3	
getStartjahrFor (String semester)	semester	zulässig: "SS12", "SS13", "SS26", "WS12", "WS13", "WS26" unzulässig: "", "SS123", "HS12"	2012, 2013, 2026, 2012, 2013, 2026, IllegalArgumentException bei unzulässigem Semester
isSommersemester (String semester)	semester	zulässig: "SS12", "SS13", "SS26", "WS12", "WS13", "WS26" unzulässig: "","SS123","HS12"	true, true, true, false, false, false, IllegalArgumentException bei unzulässigem Semester
isZulaessigeKwFuerSemester (int kw, String semester)	kw	SS_KwStart, SS_ KwStart+1, SS_KwEnde, SS_KwEnde-1, 31, SS_ KwStart-1, SS_KwEnde+1, 0 , 1, -1	true, true, true, true, true, false, false, false, false, false
	semester	"SS12"	
isZulaessigeKwFuerSemester (int kw, String semester)	kw	WS_KwStart, WS_ KwStart+1, WS_KwEnde, WS_KwEnde-1, 52, 1, WS_ KwStart-1, WS_KwEnde+1, 53, 0,-1	true, true, true, true, true, true false, false, false, false, false
	semester	"WS12"	
isZulaessigeKwFuerSemester (int kw, String semester)	kw	15	IllegalArgumentException
	semester	unzulässig: "","SS123","HS12"	
normalisiereKw (int kw, String semester)	kw	SS_KwStart, SS_KwStart+1, SS_KwEnde, SS_KwEnde-1, 31, SS_ KwStart-1, SS_KwEnde+1, -5, 0 , 60	SS_KwStart,SS_KwStart+1 SS_KwEnde,SS_KwEnde-1, 31 0, 0, 0 , 0, 0
	semester	"SS12"	

normalisiereKw (int kw, String semester)	kw	WS_KwStart, WS_ KwStart+1, WS_KwEnde, WS_KwEnde-1, 53, 60, WS_ KwStart-1, WS_KwEnde+1, -5, 150	WS_KwStart, WS_ KwStart+1, WS_KwEnde, WS_KwEnde-1, 1, 60, 0, 0, 0, 0
	semester	"WS12"	
normalisiereKw (int kw, String semester)	kw	15	IllegalArgumentException
	semester	unzulässig: "","SS123","HS12"	
toTime(String semester, int kw, int wochentag, String uhrzeit)	Semester	zulässig: "SS12", "WS12" unzulässig: "HS12"	null, da alle zu testenden Wertekombinationen unzulässig. Getestet werden: - Kombinationen mit nur einem unzulässigen Wert - die unzulässigen Semester-KW- und Semester-KW-Wochentag- Kombinationen mit sonst ausschließlich zulässigen Werten
	kw	zulässig zu "SS12": SS_ KwStart unzulässig zu "SS12": SS_ KwStart-1, SS_ KwEnde+1 unzulässig zu "WS12": WS_ KwStart-1, WS_ KwEnde+1 immer unzulässig: -1, 0	
	wochentag	zulässig: 2 unzulässig: -1, 0	
	uhrzeit	"24:45","7:10"	
toTime(String semester, int kw, int wochentag, String uhrzeit)	semester	"SS12", "WS12"	Ein Time-Objekt mit den jeweils entsprechenden Werten für Jahr, Kw, Wochentag, Stunde, Minute
	kw	zulässig zu "SS12": SS_KwStart, SS_KwEnde zulässig zu "WS12": WS_KwStart, WS_KwEnde	
	wochentag	zulässig: 2, 1, 6	
	uhrzeit	zulässig: "14:30","07:10"	

Tabelle D.5: Automatisierter Test der TerminLogik

Test des PlanLayout

Um das Plan-Layout zu implementieren, wird zunächst mit der Klasse `TestPlan` eine konkrete ViewGroup erstellt. Diese delegiert Methodenaufrufe an die Superklasse PlanLayout, so dass die zu implementierenden, geschützten Methoden `onMeasure(int widthMeasureSpec, int heightMeasureSpec)` zum Messen der Breite und Höhe, sowie `onlayout(int t, int l, int r, int b)` zum Positionieren der Child-Views mit Testwerten angesprochen werden können. Weiterhin dient der `TestPlan` zum Auswerten zulässiger und unzulässiger Layout-Parameter.

Test der Methode	Beschreibung	Eingabewerte	Soll- Ergebnis
`boolean checkLayoutParams (ViewGroup.LayoutParams layoutParams)`	Verwenden eines LayoutParam-Objektes mit (Startspalte, Endspalte, Startzeile, Endzeile) für den Parameter layoutParams	zulässig: (0,1,0,1), (0,10,0,10), (2,5,4,7) unzulässig: (0,0,0,5), (0,5,0,0), (0,11,0,5), (0,5,0,11), (-1,5,0,5), (0,5,-1,5)	true bei zulässigen Werten, false bei unzulässigen Werten
`LayoutParams generateDefaultLayoutParams()`	1. Direkter Aufruf der Methode und Überprüfen des LayoutParams-Objektes 2. Hinzufügen einer Child-View ohne Layout-Parameter und Überprüfen der in der ChildView gesetzten Defaults	-	1. und 2.: LayoutParams mit (0,1,0,1) - Startspalte=0, Endspalte=1, Startzeile=0, Endzeile=1

void onMeasure int widthMeasureSpec, int heightMeasureSpec)	widthMeasureSpec	1. (200, MeasureSpec.*EXACTLY*) 2. (1300, MeasureSpec.*AT_MOST*)	<u>Für den Plan:</u> getMeasuredWidth() liefert: 1. 200 (statt 10*30=300), 2. 300 getMeasuredHeight() liefert: 1. 400 (statt 10*30=300), 2. 300 <u>Für die Veranstaltung:</u> getMeasuredWidth() liefert: 1. 200 (statt 10*30=300), 2. 180 getMeasuredHeight() liefert: 1. 400 (statt 10*30=300), 2. 180
	heightMeasureSpec	1. (400, MeasureSpec.*EXACTLY*) 2. (1400, MeasureSpec.*AT_MOST*)	
void onLayout (boolean changed, int l, int t, int r, int b)	Den Testplan mit einer exakten Größe (Breite=400, Höhe=600) und einem Padding (left=14, top=15, right=16, bottom=17) erstellen, eine Layout-Anforderung simulieren durch Aufruf von: layout(0,3,0,0). Die gesetzten Child-Größen und -Positionen überprüfen (abgerundet auf volle Pixel)	(true, 0, 3, 0, 0)	v.getMeasuredWidth() liefert: 222 = ((400 - 14 - 16) / 10) * 6 Zellen v.getMeasuredHeight() liefert: 454 = ((600 - 15 - 17) / 10) * 8 Zellen v.getLeft() liefert: 14 (= PaddingLeft) v.getTop() liefert: 15 (= PaddingTop) v.getRight()liefert: 238 (= Breite + PaddingRight v.getBottom() liefert: 471 (= Höhe + PaddingBottom

Tabelle D.6: Automatisierter Test des PlanLayouts

Test des Tagesplans

In den Tests für den Tagesplan wird der Konstruktor

```
Tagesplan(Context context, int weekday, Time startTime, Time endTime,
int timeframe, List<Veranstaltungsdaten> vdatenListe)
```

jeweils mit dem Context des TestCase und den dargestellten Parametern aufgerufen. Als vdatenListe wird in TF1-14 null gewählt, in TF15 eine leere Liste.

TF	Beschreibung	Parameter	Eingabewerte (in dieser Reihenfolge)	Soll- Ergebnis / Nachbedingungen
1-8	Aufruf des Konstruktors mit <u>unzulässigen</u> Werten. 1.-3.: unzulässiger Tag, 4.: Dauer nicht über 5 Std., 5.: Ende um 00:00 Uhr, 6.-8.: unzulässige Auflösung	weekday	0, 7, -1, 3, 3, 3, 3, 3	IllegalArgument Exception
		startTime	7:15, 7:15, 7:15, 10:15, 16:15, 7:15, 7:15, 7:15	
		endTime	19:30, 19:30, 19:30, 15:15, 00:00, 19:30, 19:30, 19:30	
		timeframe	15, 15, 15, 15, 15, 4, 61, -10	
9-15	Aufruf des Konstruktors mit <u>zulässigen</u> Werten. Dabei werden erlaubte Grenzwerte für den Wochentag (1 und 6), die	weekday	1, 6, 3,3, 3, 3, 3, 3	Der Tagesplan wurde erfolgreich erzeugt (es wird keine Exception geworfen)
		startTime	07:15, 07:15, 00:00, 10:15, 07:15, 07:15, 07:15, 07:15	

	Start- und Endezeit (00:00), die Dauer (5Std + 1Min) und die Auflösung (5 und 60) überprüft	endTime	19:30, 19:30, 00:00, 15:16, 19:30, 19:30, 19:30, 19:30	
		timeframe	15, 15,15, 15, 5, 60, 15, 15	
16	Konstruktion von Tagesplänen für verschiedene Wochentage und Uhrzeiten, Überprüfen der generierten TagesplanZeilen: die Start- und Endezeiten, die Anzahl der TagesplanZeilen, die Wochentage und die ContextMenuInfo (siehe *)	(siehe *)	(siehe *)	Überprüfen, ob die Start- und Endezeiten korrekt gerundet werden, ob die Anzahl der TagesplanZeilen korrekt ist (siehe *)
17	Erzeugen des Plans mit einer Liste von Veranstaltungsdaten für verschiedene Start- und Endzeiten (siehe **)	weekday	3	Positionen und Größe der im Plan eingetragenen Veranstaltungen sind korrekt (siehe **)
		startTime	7:15	
		endTime	20:30	
		timeframe	15	

* Die Eingabe- und Erwartungswerte sind dem dokumentierten Quellcode zu entnehmen

** Die manuellen Tests stehen an dieser Stelle im Vordergrund: Die Veranstaltungsdaten-Liste gibt nur beispielhaft einige zulässige und unzulässige Beginn- und Ende-Zeiten für die Veranstaltungen vor.

Tabelle D.7: Automatisierter Test des Tagesplans

Test der Wochenansicht

Die `Wochenansicht-`Activity verwendet Voreinstellungen für Semester und Samstag-Funktion. Diese werden vor dem Start der Activity in den Anwendungseinstellungen gespeichert: Semester jeweils „SS12“, Anzahl sichtbarer Wochentage: 5 (und 6 zum Test der Tab-Anzahl und -Inhalte).

Bei der Ausführung der Tests muss darauf geachtet werden, dass der Screen-Lock des Testgerätes deaktiviert ist, damit das Options- und Kontextmenü über Key-Events angesteuert werden kann.

Test-Beschreibung	Soll- Ergebnis / Nachbedingungen
Abfragen aller im Zielkontext zu verwendenden String-Ressourcen per `getString(ID)`	Alle Ressourcen sind vorhanden, es wird keine NotFoundException geworfen
Testen, ob der Bildschirmtitel nach dem Start korrekt initialisiert wird; Aufruf per `getTitle()`	`getTitle()` liefert einen String mit dem Plan-Semester und der aktuellen Kalenderwoche im geforderten Format
Testen, ob die Anzahl der Tabs, die `getTabCount()` des TabWidget liefert, der Anzahl der eingestellten Wochentage entspricht, und ob sie mit der jeweiligen Wochentag-Bezeichnung aus dem Ressource-Array beschriftet sind	Für jeden Wochentag enthält das TabWidget einen Tab mit der jeweiligen Bezeichnung
Aufruf des Optionsmenüs per Key-Event, Auswahl der Option „Einstellungen“ per Key-Event, Überprüfen der gestarteten Activity per `ActivityMonitor` (Timeout: 5 Sekunden)	Die Key-Events lösen den Start der Activity `Einstellungen` aus

Aufruf des Optionsmenüs per Key-Event, Auswahl der Option „Neue Veranstaltung“ per Key-Event, Überprüfen der gestarteten Activity per `ActivityMonitor` (Timeout: 5 Sekunden)	Die Key-Events lösen den Start der Activity `VdatenFormular` aus
Aufruf der Callback-Methode `updateWochenansichtTabContent()` zum Aktualisieren der Tab-Inhalte mit einer Tagesplan-Liste, Überprüfen der TabContent-View	Die TabContent-View enthält für jeden Wochentag eine ScrollView mit jeweils einem Tagesplan, der Tab für den aktuellen Wochentag ist selektiert

Tabelle D.8: Automatisierter Test der Wochenansicht

Test des WochenansichtInitTask

Zum Test des WochenansichtInitTask wird zunächst eine Mock-Klasse für den DatenbankManager implementiert, sowie ein MockCursor, der Testdaten liefert. Dies ermöglicht den Abgleich der Daten, die der Task an die Wochenansicht zurückliefert. Die Framework-Klasse `MockCursor` wird erst ab API-Level 8 unterstützt, bei älteren Geräten muss daher auf die automatisierten Tests verzichtet werden. Ab Android v2.2 sind die in dieser Tabelle aufgeführten Tests jedoch durchführbar.

Zu Beginn der Tests wird als Context eine Wochenansicht-Activity erzeugt. Dann wird ein Task gestartet, der die Methode `onPostExecute(List<Tagesplan> result)` überschreibt. Dies ist notwendig, damit dieser nicht die Wochenansicht, sondern eine Variable in der TestCase-Klasse mit der Tagesplan-Liste aktualisiert, welche überprüft werden kann. Zudem kann er in einer letzten Anweisung in `onPostExecute` einen Semaphor freigeben, der zur Synchronisation mit der TestCase-Klasse dient. Die verwendeten Testdaten werden in der Tabelle nicht im Detail aufgeführt und sind dem Quellcode zu entnehmen.

Test-Beschreibung	**Soll- Ergebnis / Nachbedingungen**
Prüfen, ob der Task seine Arbeiten erledigt und eine Liste von Tagesplänen zurückgeliefert hat	Der Task ist im Status „finished“, die vom Task zurückgelieferte Tagesplan-Liste enthält für jeden Wochentag einen Tagesplan.
Die Anzahl erwarteter Tagesplan-Zeilen aus den Einstellungen ermitteln, die der Task verwendet hat, und mit der Anzahl an Zeilen der zurückgelieferten Tagespläne abgleichen	Die Anzahl an Zeilen ist für jeden vom Task erstellten Tagesplan gleich und entspricht der erwarteten Anzahl
Die Veranstaltungsdaten in den vom Task gelieferten Tagesplänen mit den Testdaten vergleichen	Die gelieferten Veranstaltungsdaten entsprechen den Testdaten
Testen, ob alle Zeilen und Veranstaltungen eines Tagesplans auf Touch-Events reagieren. Hierzu wird lediglich die Anzahl der Touchables eines Tagesplans per `getTouchables()` überprüft.	Die Anzahl der Zeilen und Veranstaltungen ergibt jeweils die Anzahl der Touchables eines Tagesplans

Tabelle D.9: Automatisierter Test des WochenansichtInitTask

Test des Formulars mit dem Kw-CheckboxDialog

Für das Formular und den KW-Checkbox-Dialog zur Erfassung der Veranstaltungsdaten wird durch die automatisierten Tests zur Implementierungszeit hauptsächlich eine korrekte Initialisierung der Formularelemente sichergestellt (vgl. Tabelle 5.7). Um die Komponente dennoch ausreichend zu testen, werden daher nach der Implementierung die hier beschriebenen manuellen Tests durchgeführt. Das Formular wird vor jedem Test aus der Wochenansicht-Activity heraus gestartet. Die Prüfung unzulässiger Eingaben ist ein Test gegen R10 des Anforderungskatalogs.

Test-Beschreibung	Soll- Ergebnis / Nachbedingungen
Überprüfen der max. Länge der Eingabezeichen – nicht möglich sind Eingaben mit den unzulässigen Zeichenlängen: Name>30 oder Dozent>20 oder Raum>20 Klick auf Button „Speichern“ nach Eingabe einer unzulässigen Anzahl an Zeichen für die EditTest-Felder: Name < 2 oder Kalenderwochen < 1 Klick auf Button „Speichern“ nach Eingabe einer unzulässigen Veranstaltungsdauer mit: Endezeit – Startzeit <= 15 Minuten	Ein Toast zur Korrektur der Daten wird angezeigt. Die Meldung enthält einen Hinweis zur Korrektur der ersten unzulässigen Eingabe in der Reihenfolge: Name zu kurz, Dauer zu kurz, Kalenderwochen-Feld leer
Klick auf Button „Abbrechen“ nach Eingabe zulässiger Daten	Die Formular-Activity wird beendet
Klick auf Button „Speichern“ nach Eingabe zulässiger Daten	Die Formular-Activity ist beendet und die Formulardaten sind in der Datenbank gespeichert worden, die Wochenansicht wird neu gestartet
Auswahl verschiedener Kalenderwochen im Kw-Checkbox-Dialog und überprüfen des Inhaltes im Kw-EditText-Feld nach dem Schliessen des Dialoges	Der EditText enthält einen String mit allen im Dialog ausgewählten Kalenderwochen in dem Format nach Anforderung R13

Tabelle D.10: Manueller Test des Formulars mit dem Kw-CheckboxDialog

Test des Alarm-Managements

Die Tests der abstrakten Klassen des Alarm-Managements werden für drei konkrete Implementierungen erstellt:

Der `ConcreteWakeLockedService` fordert nach dem Start einen Wakelock vom PowerManager an und wartet danach auf die Freigabe eines Semaphors, der ihm beim Start übermittelt wurde. Über einen Getter gewährt er Zugriff auf den gehaltenen Lock, der während der Wartezeit per `isHeld()` überprüft wird. Der Test ist erfolgreich, wenn die Methode `true` zurückliefert.

Mit Hilfe des `ConcreteRingerModeChanger` wird in einem weiteren Test die Verarbeitung von Anfragen zum Ändern des Klingelmodus geprüft. Durch einen direkten Aufruf der normalerweise vom System angesprochenen Methode `onReceive(Intent request)` wird der Empfang der Anfragen simuliert. Eine Synchronisation des Test-Thread ist damit nicht notwendig. Der

`ConcreteRingerModeChanger` speichert die in `onReceive(Intent request)` empfangenen Anfragen (Intents) in einer Liste und gewährt auf diese per Getter öffentlichen Zugriff. Für alle drei Klingelmodi „silent“, „normal“, und „vibrate“ wird jeweils eine Anfrage gestellt. Der Test ist erfolgreich, wenn die Intent-Liste nach jedem Aufruf von `onReceive` und um 1 größer ist, und der AudioManager per `getRingerMode()` den jeweils angefragten Klingelmodus zurückliefert.

Die dritte Klasse, der `ConcreteAlarmAnmeldeService`, nimmt mit dem Start-Intent drei Extras entgegen: Einen Klingelmodus, den er in `getAlarmIntent(Intent request)` an den anzumeldenden Alarm-Intent weitergibt, den Zustand der Alarmfunktion (aktiv / nicht aktiv), sowie einen Semaphor, den er in `doBeforeStopService()` freigibt, so dass der Test-Thread über den Zeitpunkt benachrichtigt werden kann, zu dem der Service einen Alarm an- oder abgemeldet hat.

Tabelle D.11: Automatisierter Test des Alarm-Managements

Test der Lautlosfunktion

Um die Tests für diese Komponente durchzuführen, wird zunächst ein MockContext erstellt, so dass der Start des `VLautlosModusService` simuliert werden kann: Bei einem Aufruf der Methode `startService(Intent intent)` speichert er den Start-Intent in einer Liste, auf die er per Getter öffentlichen Zugriff gewährt. Zusätzlich wird eine Erweiterungsklasse des `VLautlosModusWechsler` erstellt, die ausschließlich dazu dient, die innerhalb `onReceive` angesprochene Methode `scheduleNextMode` nach außen zugänglich (public) zu machen. So wird ein direkter Aufruf aus der TestCase-Klasse mit dem MockContext erlaubt und jeder Start des VLautlosModusService innerhalb der Methode `scheduleNextMode` durch einen zusätzlichen Intent in der Liste des MockContext bestätigt.

Der Test des `VLautlosModusWechsler` erfolgt in mehreren Durchläufen, in denen jeweils drei Klingelmodi definiert werden: Ein Modus, mit das Gerät initialisiert wird, einer, in den innerhalb `onReceive` gewechselt wird, sowie ein Modus, der innerhalb `scheduleNextMode` für den nächsten Wechsel vorgemerkt wird. Nach der Initialisierung werden `onReceive` und `scheduleNextMode` direkt nacheinander aufgerufen. Der Test ist erfolgreich, wenn für alle möglichen Kombinationen der Klingelmodi der angeforderte Modus eingestellt und eine Anforderung für den nächsten Modus in der Liste der gespeicherten Intents identifiziert werden kann.

Zur Implementierung der zweiten Klasse der Komponente wird ein ServiceTestCase erstellt, der eine Erweiterungsklasse des `VLautlosModusService` testet. Diese sorgt zum einen dafür, dass eine Anfrage beim Start des Service zunächst nicht verarbeitet wird (sie überschreibt `onHandleIntent`), und zum anderen dafür, dass alle geschützten Methoden, die im `VLautlosModusService` implementiert sind, aus dem TestCase heraus aufgerufen werden können. Auf diese Weise lässt sich der Start des Service einmal zum Deaktivieren und einmal zum Aktivieren der Lautlosfunktion simulieren:

Mit einem manuellen Test der Lautlosfunktion schließt die erste Teststufe ab. Während einer laufenden Veranstaltung muss der Klingelmodus bei De-/Aktivierung der Funktion sofort umgestellt werden, zukünftige Termine werden korrekt vorgemerkt. Überprüft wird dies anhand von Debug-

Ausgaben und des Statusleisten-Symbols für den aktiven Klingelmodus.	
Aufruf der Methode	**Soll- Ergebnis / Nachbedingungen**
void doBeforeService (Intent request)	Die Datenbank wurde geöffnet.
boolean isAlarmFunktionAktiv (Intent request)	Der Zustand der Lautlosfunktion wurde über den Key aus den String-Ressourcen `intent_key_lautlosfunktion_aktiv` korrekt aus dem Request ermittelt (Anmelden → true, Abmelden → false).
Intent getAlarmIntent (Intent request)	Die Methode liefert einen expliziten Alarm-Intent, der die Klasse `VLautlosModusWechsler` als Ziel, sowie das Semester und den angeforderten Klingelmodus enthält. Wird zum Abmelden nicht aufgerufen.
Time getAlarmTermin (Intent request)	Es wird null zurückgeliefert, da der DatenbankManagerMock keine Veranstaltungsdaten liefert. Zum Abmelden wird diese Methode nicht aufgerufen.
void doBeforeStopService (Intent request)	Die Datenbank wurde geschlossen. Beim Abmelden wurde der Klingelmodus auf „normal“ geändert.

Tabelle D.12: Automatisierter Test der Lautlosfunktion

E Anhang 2: Inhalt der CD

`Studienplaner.pdf` – dieses Dokument im PDF-Format

`Studienplaner.zip` – das Eclipse-Projekt (inkl. Test-Projekt)

`Studenplaner.apk` – eine signierte Installationsdatei

Erklärung

Hiermit versichere ich, dass ich die vorliegende Bachelorarbeit im Sinne der Prüfungsordnung nach §22(4) bzw. §24(4) ohne fremde Hilfe selbständig verfasst und nur die angegebenen Quellen verwendet habe. Sofern kein Literatur- oder Quellenverweis angegeben ist, sind Tabellen oder Abbildungen selbst erstellt.

Brügge, den 21.09.2011	**Benjamin Thom**
Ort, Datum	Unterschrift