Fachbereich Informatik und Medien

BACHELORARBEIT

Implementierung eines mobilen GoogleWave-Clients zur Unterstützung kooperativer Arbeitsorganisation unter Verwendung vom Android-SoftwareDevelopmentKit 2.0

Vorgelegt von: Christian Leube

am: 06.09.2010

zum

Erlangen des akademischen Grades

BACHELOR OF SCIENCE

(B.Sc.)

Erstbetreuer: Prof. Dr. rer. nat. Rolf Socher

Zweitbetreuer: Prof. Dr. Kai von Luck

Selbstständigkeitserklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit zum Thema

Implementierung eines mobilen GoogleWave-Clients zur Unterstützung kooperativer Arbeitsorganisation unter Verwendung vom Android-SoftwareDevelopmentKit 2.0

vollkommen selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt sowie Zitate kenntlich gemacht habe. Die Arbeit wurde in dieser oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegt.

Brandenburg/Havel, den 06.09.2010

Unterschrift

Zusammenfassung

Diese Bachelorarbeit beschäftigt sich mit der Umsetzung eines mobilen Clients für den Internetdienst GoogleWave, zur Unterstützung kooperativer Arbeitsorganisation. Entwickelt wird der Client als Applikation für ein Android-Betriebsystem ab Version 2.1 Update 1.
Das Hauptaugenmerk der Arbeit liegt auf der Entwicklung der Applikation im Sinne des Software Engineering. In diesem Kontext wird zunächst eine objektorientierte Analyse des Programmes durchgeführt. Die daraus resultierenden Ergebnisse werden genutzt um das objektorientierte Design der Applikation zu entwerfen. Schließlich wird die Applikation unter Berücksichtigung der erworbenen Erkenntnisse implementiert. In dem Dokument werden entschiedene Programmcodeausschnitte näher erläutert.

Der Nutzer der Applikation wird in die Lage versetzt, sich auf GoogleWave einzuloggen. So kann er von dem Smartphone aus seinen GoogleWave-Account verwalten.

Abstract

The bachelor thesis deals with the conversion of a mobile client for the internet service GoogleWave, to support co-operative working. The client is developed as an application for an android operatingsystem from version 2.1 Update 1.
This bachelor thesis focuses on the development of the application in terms of he software engineering. In this context, an object-oriented analysis of the program is carried out at first. The results are used to prepare an object-oriented design for the application. Finally the application will be implemented with regarding the acquired knowledge. Determined program code cuttings are specified more detailed in the document.

The user of the application is moved into the position to log in on GoogleWave. Thus he is able to manage his GoogleWave account by using the Smartphone.

Inhaltsverzeichnis

1 Einleitung

Die Teamarbeit ist ein wichtiger Faktor bei der erfolgreichen Bearbeitung von Problemstellungen. In einem Team werden unterschiedlich veranlagte Wissensquellen, Fähigkeiten und Persönlichkeiten der beteiligten Personen gebündelt. Auf diese Weise werden schneller und effektiver Lösungen für Probleme gefunden.
Gerade in der heutigen Zeit ist es wichtig, dass Teammitglieder so effizient wie möglich zusammenarbeiten. Die verfügbare Ressourcen sind in der Regel knapp und der vorgegebene Zeitrahmen für die Fertigstellung des Produktes ist oft minimalistisch vom Auftraggeber vorgegeben. Zudem arbeitet ein Team in vielen Fällen nicht allein an einem Projekt. Vielmehr wird ihm eine Teilaufgabe zugewiesen. Die Bearbeitung der Teilaufgabe ist wiederum für das Vorankommen anderer Teams von großer Wichtigkeit. Daraus resultiert eine unvermeidbare relative Abhängigkeit von einander.
Die richtige Koordination innerhalb eines Teams ist daher der Schlüssel, um eine möglichst hohe Effizienz zu erreichen. Dabei zählt nicht nur, dass der Teamleiter seine Anweisungen schnell und gezielt durchstellen kann, sondern dass auch die Teammitglieder regelmäßig Informationen austauschen. Regelmäßige Kommunikation ist eine wichtige Grundlage dafür, dass die Teilaufgaben innerhalb des gegebenen Zeitraumes rechtzeitig fertig gestellt.
Um die korrekte Entwicklung der Teilprodukte und somit des gesamten Projektes zu gewährleisten, muss die Organisation in gleicher Weise auch zwischen den Teams ermöglicht werden.

Dies kann durch den Einsatz von moderner IT-Technik erreicht werden. Deren rasante Entwicklung in den letzten Jahrzehnten ermöglicht den Einsatz von performanten und dazu bezahlbaren Informationstechnologien. Längst integrieren Firmen unterschiedlichster Branchen zunehmend Computer in ihre Arbeitsabläufe. Der Computer dient als Mehrzweck-Arbeitsgerät und sei es für die einfachste Büroarbeit. Den meisten Firmen ist es daher aus technischer Sicht möglich, Software zu nutzen, welche die kooperative Arbeitsorganisation unterstützt.
Die passende Software gibt es bereits in unterschiedlichster Ausführung im Closed- als auch im Open-Source Bereich. Als Closed-Source Vertreter seien [Tea10b] TeamSnap und

[Tea10a] TeamR genannt. Zu den Open-Source Vertretern zählen [Ope10] OpenGroupware und [Goo10a] GoogleWave.
Um die aufgeführten Software-Produkte verwenden zu können, muss eine Firma nicht unbedingt einer bestimmten Branche angehören. Entgegen des Vorurteils, nur Firmen aus dem IT-Bereich könnten Nutzen aus derartiger Software ziehen, bieten sie prinzipiell jeder Firma die selben Möglichkeiten. Die Software-Produkte stellen einen vielseitigen Funktionsumfang zu Verfügung und lassen sich intuitiv ohne besondere Computerkenntnisse benutzen.
Somit ist jede Firma mit vorhandener technischer Ausstattung in der Lage, die Effizienz ihrer Teams durch bessere Kommunikation und Organisation zu erhöhen.

1.1 Szenario

Im folgenden wird ein mögliches Szenario beschrieben. Es soll anschaulich den Diskursbereich für diese Bachelorarbeit umreißen. Das Szenario wird daher während der Arbeit immer wieder aufgegriffen, um als Beispiel zu dienen.

In Kapitel 1 wurde die breite Einsatzmöglichkeit von Software zur Unterstützung kooperativer Arbeitsorganisation erwähnt. Daher wird die Branche der Firma, bei der sich das Szenario abspielt, nicht weiter spezifiziert. Auch auf die Art des Projektes oder den Aufgabenbereich des Teams sowie den Inhalt des Dokumentes wird nicht weiter eingegangen.

Eine Firma betraut eines ihrer Teams mit einem Projekt, welches die Bearbeitung eines Dokumentes beinhaltet. Für das Projekt existiert ein strikter Zeitplan, den es einzuhalten gilt. Innerhalb des Teams hat sich, auf Grund von fachlichen Speziealisierungen der Teammitglieder, eine entsprechende Aufgabenteilung eingestellt.Zum Projektstart ist die Hälfte der Teammitglieder auf einer längeren Weiterbildung.
Es wird trotz Minimalbelegschaft mit der Bearbeitung des Projektes begonnen, um den Zeitplan einzuhalten. Bald stößt das verbliebene Team auf spezifische Problemstellungen. Für gewöhnlich werden derartige Problemstellungen von Teammitgliedern mit der entsprechenden Spezialisierung bearbeitet, jedoch sind eben diese auf der Weiterbildung.
Nun gibt es für das verbliebene Team mehrere Möglichkeiten mit den Mitarbeitern zu kommunizieren, welche sich zur Zeit auf Weiterbildung befinden.

1. Einberufen einer Telefonkonferenz
2. Nutzen des Smart Message Services (SMS)
3. Versenden einer E-Mail

4. Nutzung einer Software welche die kooperative Arbeitsorganisation unterstützt

Bei einer Telefonkonferenz könnten die Problemstellungen schnell an die entsprechenden Teammitglieder weitergegeben werden. Jedoch ist eine Telefonkonferenz angesichts einer Weiterbildung für alle Beteiligten störend und ineffizient. Auch das zu bearbeitende Dokument kann auf diese Weise nicht weiter gegeben werden.
Das Versenden einer SMS ist zwar wesentlich diskreter, kann aber nicht so viel Information auf einmal an die jeweilige Person übermitteln. Auch hier gibt es wieder das Problem, dass das Dokument nicht mitgeschickt werden kann.
Eine E-Mail könnte Abhilfe schaffen. Alles was der Mitarbeiter an Information benötigt, um dem verbliebenen Team weiterhelfen zu können, kann mit einem mal übermittelt werden. Auch das Dokument kann im Anhang mit versendet werden. Darüber hinaus ist eine E-Mail diskreter als ein störender Anruf. Allerdings ergeben sich auch bei der E-Mail einige Nachteile. Zwar wird eine E-Mail, genau wie eine SMS umgehend an den Empfänger verschickt, jedoch wird der Empfänger nicht wie bei einer SMS auf die neue Nachricht sofort hingewiesen. Das heißt, das Teammitglied muss erst einen Computer mit Internetverbindung aufsuchen und sein E-Mail-Account auf neue Nachrichten prüfen. Ob das jeweilige Teammitglied auf das Hilfegegesuch des verbliebenen Teams antwortet hängt also ganz von der Regelmäßigkeit ab, in dem der E-Mail-Account auf neue Nachrichten geprüft wird. In diesem Szenario wird davon ausgegangen, dass den Teammitgliedern auf der Weiterbildung keine Laptops mit einer Internetverbindung zur Verfügung steht.
Ein weiteres Problem beim Nutzen einer E-Mail, ergibt sich im Hinblick auf das mitgesendete Dokument. Das Dokument geht nicht ausschließlich an ein Teammitglied. Es gibt mehrere auf Weiterbildung befindliche Teammitglieder, welche bei der Problemlösung helfen können. Also wird auch an alle dieses Dokument versendet. Wenn jedes Teammitglied nun seine Lösung im Dokument verwirklicht, kommt es unweigerlich zu einem Versionskonflikt. Mühselig müssen alle Vorschläge ausgewertet und zu einem Dokument zusammen gefasst werden. Dabei stößt das verbliebene Team ständig auf Lösungsvorschläge, welche grob betrachtet die selbe Intention verfolgen, aber sich in winzigen Details unterscheiden.
Alle erläuterten Kommunikationsversuche bergen Nachteile in sich. Mit keinem ist es möglich, eine bestimmte Gruppe von Teammitgliedern auf der Weiterbildung, diskret und vollständig über eine Problemstellung aufzuklären, geschweige denn ein Dokument zu zusenden, welches ohne Versionskonflikte bearbeitet werden kann.
Die Firma ist sich solcher Situationen bewusst. Zu diesem Zweck verwendet sie eine Software, welche kooperative Arbeitsorganisation unterstützt. Im darzustellenden Szenario wird postuliert, dass die Firma den Internetdienst GoogleWave verwendet. Jeder Mitarbeiter der

Firma hat bei GoogleWave einen Account. Die entstehenden Daten werden auf einem Server gespeichert, welcher über das Internet zugänglich ist. Auf diese Weise kann jeder Mitarbeiter auch außerhalb der Firma, zum Beispiel von zu Hause aus oder während einer Weiterbildung, auf notwendige zugreifen Daten und Diskussionen einsehen. GoogleWave eignet sich zudem zum Weitersenden von Dokumenten jeglicher Art. Diese können von mehreren Personen gleichzeitig, ohne Versionskonflikte auszulösen, bearbeitet werden.
Auf diese Weise kann das Team in der Firma alle Teammitglieder auf der Weiterbildung diskret über die Problematik in Kenntnis setzten und das Dokument übermitteln. Die Teammitglieder auf der Weiterbildung ihrerseits können in dem Dokument ihre Bearbeitung vornehmen

Für weitere Betrachtungen wird festgelegt, dass die Firma ihren Mitarbeitern Smartphones als „Firmenhandys“ zur Verfügung stellt. Die Smartphones verwenden Android als Betriebssystem.

1.2 Motivation

Die eingesetzte Software zur Unterstützung kooperativer Arbeitsorganisation ,zum Beispiel GoogleWave, mag ihre Aufgabe erfüllen. Allerdings wird für ihre Verwendung immer ein Computer mit Internetverbindung benötigt. Ein gewöhnlicher Computer ist an seinen Standort gebunden und kann dem Nutzer daher nur lokalbedingt Zugriff auf die gewünschten Informationen bieten.
Es wird davon ausgeganen, dass Notebooks oder Netbooks nicht immer zur Verfügung stehen. Jedoch kommen häufig bei gut organisierten Firmen Smartphones als „Firmenhandy“ zum Einsatz. Wie es bei der Firma aus dem Szenario (s.a.1.1) der Fall ist.

Dank ihrer leistungsstarken Hardware und des umfangreichen Funktionsumfanges ihrer Betriebssysteme, ist die Entwicklung umfassender Programme möglich. Zum Beispiel können für Smartphones mit Windows Mobile oder Android als Betriebssystem bereits ganze Office-Suiten zum Einsatz kommen.[1].
Eine weitere Stärke der Smartphones ist ihre Fähigkeit von nahezu überall eine Verbindung mit dem Internet aufzubauen.

Daher ist es plausibel für Smartphones ein Programm zu entwickeln, welches kooperative Arbeitsorganisation unterstützt. Dieses Programm kann als eigenständige Software oder als eine mobile Erweiterungen von bereits bestehender Software wie GoogleWave oder TeamSnap

[1] Windows Mobil: Softmaker Office 2010
Android: Androffice V1.5

entwickelt werden.
In beiden Fällen stellen Smartphones eine geeignete Grundlage dar. Sie können komplexe Software ausführen und sind trotz ihrer Mobilität in der Lage, auf das Internet zuzugreifen. Teamorganisation wäre so, unabhängig vom Standort beziehungsweise Verfügbarkeit eines herkömmlichen Computers, nahezu immer möglich.
Bezogen auf das Szenario beitet sich die Entwicklung einer mobilen Erweiterung oder genauer eines mobilen Clients für den bereits verwendeten Internetservice GoogleWave und den bereits vorhandenen Smartphones an.

1.3 Zielsetzung

Im Hinblick auf die Motivation und unter Berücksichtigung des Szenarios ergibt sich für diese Bachelorarbeit die Zielsetzung, einen mobilen Client für GoogleWave zu implementieren.

Entwickelt wird die Applikation für ein mobiles Endgerät mit Android 2.1 Update-1 unter Verwendung des Android Software-Development-Kit 2.0.
Die Applikation soll so entwickelt werden, dass alle für einen Nutzer-Account verfügbaren Diskussionen mit den dazugehörigen Themengliederungen sowie den eigentlichen Nachrichten vom GoogleWave-Server geladen, angezeigt, erweitert oder im einzelnen manipuliert werden können. Das heißt, die Teilnahme an einer beliebigen, verfügbaren Diskussion wird in der elementarsten Form gewährleistet. Im Folgendem wird mehrfach auf die zu implementierende Applikation verwiesen. Sie wird dann kurz als „AndroidWave“ bezeichnet.
Die Arbeit befasst sich neben der Implementierung auch mit der Erfassung von grundlegenden Anforderungen und dem Entwurf der Software im Sinne des Software Engineering.

Somit ergibt sich folgende Agenda für die weiteren Kapitel. Zunächst werden wichtige Begriffe und benötigte Grundlagen für das bessere Verständnis der Arbeit im folgendem Kapitel erklärt. Alle spezifischen Anforderung der zu implementierenden Software werden im 3.1. Kapitel erfasst. Das darauf folgende Kapitel 3 wird die Applikation aus Sicht der OOA und des OODs erläutert. Im Kapitel 4 werden schließlich entscheidende Quellcode-Elemente genannt und eingehend erläutert. Im letzten Kapitel werden mögliche Entwicklungen von GoogleWave und *AndroidWave* aufgezeigt.

2 Grundlagen

2.1 Android

Android ist eine Open-Source-Software-Plattform für mobile Endgeräte wie Handys, Smartphones und Netbooks. In Kürze können in diese Liste so genannte Tablet-PCs aufgenommen werden. Die Open-Source-Software-Plattform umfasst das eigentliche Betriebssystem und das Software-Development-Kit.
Die Basis von Android bildet ein Linux-Kernel. Durch ihn werden die unterschiedlichen Gerätetreiber wie Bildschirm- , WiFi- und Tastatur-Treiber zur Verfügung gestellt. Des weiteren wird durch den Kernel die Energieverwaltung, Speicherverwaltung und die Prozessverwaltung gesteuert. Auf den Kernel bauen parallel zwei Schichten auf. Zum einen Bibliotheken beispielsweise für Grafik, SQLite und der Oberflächen-Manager und zum anderen die Android-Laufzeitumgebung mit den dazugehörigen Bibliotheken und der DVM (Dalvik Virtual Machine). Auf diese Schichten folgt der Anwendungsrahmen mit Managern für Activities, Oberflächenelemente, Window-Manager, Ressourcen-Manager und so weiter. Als letzte Schicht folgt die Anwendungsschicht. Hier sind die Android-Anwendungen und Drittanbieter-Anwendungen zu finden. Die Abbildung 2.1 auf Seite 7 veranschaulicht die Systemarchitektur von Android in einer Grafik (s.a. [uMP09a]). Programmiert wird in Android hauptsächlich mit Java. Für die Definition von Ressourcen und das Erstellen von Bildschirm-Layouts kommt die Auszeichnungssprache XML zum Einsatz.(s.a. [uMP09b])

Entwickelt wird Android durch die so genannte Oben Handset Alliance (OHA). Die OHA wurde am 5. November 2007 von 33 Unternehmen gegründet. Die Bekanntesten unter ihnen sind Google, HTC, LG Electronics, Motorola, Intel Corporation und Nvidia. Im Laufe der vergangenen Jahre ist die Anzahl der Mitglieder bis heute auf über 70 Firmen gestiegen.
Die OHA setzt auf bewährte Technologien. So wird beispielsweise Eclipse als Entwicklungsumgebung und, wie bereits erwähnt, Java als Programmiersprache verwendet.
Das Betriebssystem als auch das Software-Development-Kit wird stetig weiterentwickelt.

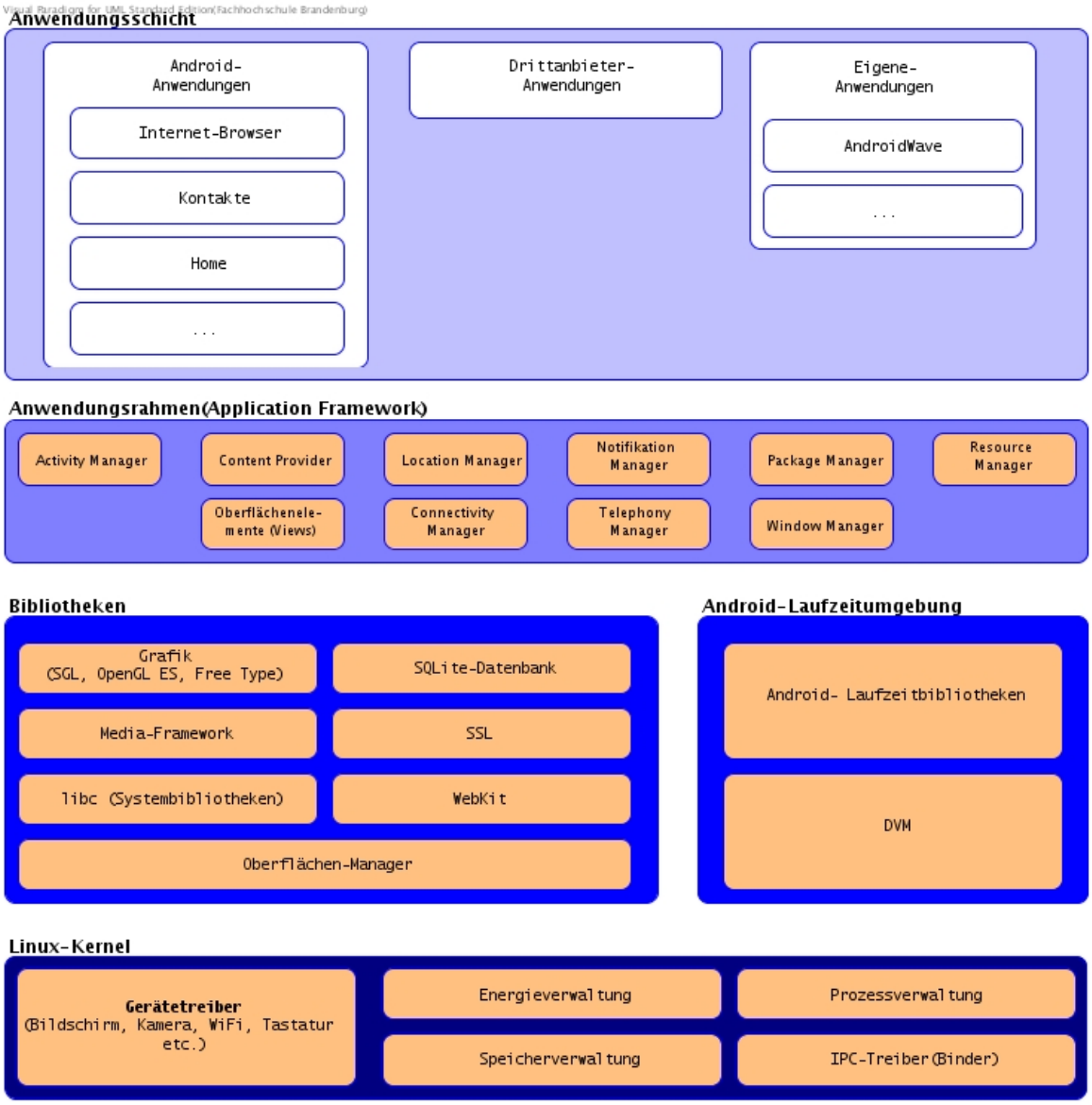

Abbildung 2.1: Die Android-Systemarchitektur

2.1.1 Dalvik Virtual Machine (DVM)

Die Dalvik Virtual Machine(DVM) ist, wie der Name bereits sagt, eine virtuelle Maschine und dient als Schnittstelle zwischen Android und der Hardware. Wie in der Abbildung 2.1 (s.a. [uMP09a]) zu sehen ist, bildet die DVM zusammen mit den Laufzeitbibliotheken von Android die Android-Laufzeitumgebung. Wird eine Android-Applikation gestartet, ist die Laufzeitumgebung für ihre Ausführung verantwortlich.

Entwickelt wurde die DVM von dem Google-Mitarbeiter Dan Bornstein. Er benannte die VM nach einem isländischen Dorf namens Dalvik, in dem seine Vorfahren lebten.

Die DVM basiert auf der quell-offenen Java VM Apache Harmony. Dies ist eine Open-Source-Version der Java-Technologie und wird von der Apache Software Foundation entwickelt.
Die DVM wurde im Aufbau und Funktionsumfang an mobilen Endgeräte angepasst. Im Gegensatz zu bisherigen JVMs besitzt die DVM keine Stack-basierende sondern ist eine Register-basierende Architektur.

Im Unterschied zu einer JVM kann eine DVM keinen Java-Bytecode ausführen. Android-Applikationen werden jedoch in der Programmiersprache Java geschrieben. Auf dem ersten Blick entsteht an dieser Stelle ein Widerspruch.
Tatsächlich verhält es sich so, dass Android-Applikationen in Java geschrieben werden. Beim compilieren entsteht auch ein Java-Bytecode. Damit dieser Java-Bytecode von einer DVM verwendet werden kann, wird er in ein Android-eigenen DEX-Bytecode umgewandelt. Für die Umwandlung wird das dx-Tool verwendet. Diese Vorgehensweise hat vor allem lizenzrechtliche Gründe.

Wenn eine Android-Anwendung ausgeführt wird, startet sie in einem eigenen Betriebssystem-Prozess. Die DVM ist klein und performant. So kann jede Anwendung in einer eigenen DVM gestartet werden. Dies kostet auf den ersten Blick mehr Ressourcen, wirkt sich jedoch positiv auf die Sicherheit und Verfügbarkeit aus, da die Anwendung einen eigenen Speicherbereich erhält. Stirbt ein Prozess, wird nur die dazugehörige Anwendung terminiert. Ein weiterer Sicherheitsmechanismus ist durch einen eigenen Betriebssystem-User pro Anwendung implementiert. Daraus folgt, dass eine Anwendung in einem Prozess unter einem User in einer eigenen DVM läuft. Hierdurch wird der Speicher einerseits durch die Berechtigung des Betriebssystems geschützt, andererseits durch die Sandbox, in der die Anwendung innerhalb der DVM ausgeführt wird. Das unerlaubte Zugreifen einer Anwendung auf die Daten einer anderen Anwendung ist daher unmöglich. (s.a. [uMP09c])

2.1.2 Datenübertragung via Internet

Mit Android werden prinzipiell zwei Lösungen realisiert, um Daten über das Internet auszutauschen:

1. Datenübertragung via Mobilfunknetz und
2. Datenübertragung via WLAN.

Ist eine Verbindung mit dem Internet hergestellt, kann der Nutzer auf dessen Inhalte zugreifen, um Daten auf das mobile Endgerät beziehungsweise auf einen Server im Internet zu laden. Neben dem Browser sind auch Applikationen in der Lage, derartige Aufgaben

zu übernehmen. Beispielsweise kann eine Applikation so implementiert werden, dass sie Daten von GoogleWave auf das mobile Endgerät lädt beziehungsweise Daten an GoogleWave schickt.
Im Nachfolgendem werden diese beiden benannten Varianten näher erläutert.

2.1.2.1 Datenübertragung via Mobilfunknetz

Der Zugriff auf das Internet mittels Datenübertragung via Mobilfunknetz wird durch so genannte Mobilfunkstandards ab der zweiten Generation realisiert. Zu den verfügbaren Mobilfunkstandards gehören:

1. GSM und
2. UMTS.

Wird eine Internetverbindung von Android unter Verwendung des Mobilfunktstandards GSM aufgebaut, wird gewöhnlich die GPRS-Erweiterung EDGE verwendet. GPRS ist ein nachträglich für GSM entwickelter Dienst zur Übertragung von Datenpaketen aus und in das Internet, mit einer Geschwindigkeit von 21,4 kbits/s pro Zeitschlitz. Durch die Erweiterung EDGE sind Geschwindigkeiten bei der Datenübertragung von 59,2 kbits/s pro Zeitschlitz erreichbar.
UMTS ist der aktuelle Mobilfunkstandard für mobile Endgeräte und erreicht eine Datenübertragungsrate von bis zu 384 kbits/s. Mit der UMTS Erweiterung HSDPA sind Raten von bis zu 14 Mbits/s möglich.

Mit der Verwendung der Mobilfunkstandards für den Zugriff auf das Internet ist der Nutzer von mobilen Endgeräten weitestgehend von herkömmlichen Netzwerken wie LAN oder WLAN unabhängig. Er kann von praktisch jedem Standort aus auf das Internet zugreifen. Verfügbarkeit und Geschwindigkeit der Datenübertragung sind ausschließlich von der Stärke des Empfangs und der Netzabdeckung des angebotenen Übertragungsstandarts abhängig.

2.1.2.2 Datenübertragung via WLAN

Der Zugriff auf das Internet mittels Datenübertragung via WLAN ist zunächst von der Hardware des Endgerätes und der Verfügbarkeit eines WLAN-Netzes abhängig.

Unter WLAN wird ein lokales Funknetzwerk verstanden, welches den IEEE-802-11-Standard verwendet. Im Gegensatz zu einem Mobilfunkstandard lassen sich mit einem WLAN nur 30 m bis 300 m Entfernung zu einem Accesspoint überbrücken. Im Bezug auf die Sicherheitsstandards in einem WLAN-Netzwerkes unterstützt Android Sicherheitsprotokolle wie WEP, WPA/WPA2-PSK und 802.1x Enterprise.
Die Verwendung eines WLAN-Netzwerkes bietet dem Nutzer eine Alternative zu einem Mobilfunkstandard wie GSM oder UMTS. Zwar ist der Nutzer in Bezug auf die Mobilität stärker eingeschränkt, jedoch fallen je nach Situation keine beziehungsweise vergleichbar geringere Kosten für die Nutzung an. Auch in einem WLAN-Netz ist die Geschwindigkeit und die bereits erwähnte Verfügbarkeit von den örtlich vorhandenen Feldstärken des WLAN-Netzes abhängig.

2.2 Google Wave

GoogleWave ist eine Internet-Plattform für Kommunikation und zur Unterstützung kooperativer Arbeitsorganisation. Der Internetdienst GoogleWave macht sich unterschiedliche Konzepte bereits bestehender Internetdienste zunutze. Wird eine neue Nachricht geschrieben, ähnelt der Ablauf sehr an das Schreiben einer E-Mail. Die Empfänger werden in einer entsprechenden Liste eingetragen und in einem großen Textfeld wird die Nachricht formuliert. Zudem können jede Art von Dateien an eine Wave angehangen werden.
Bei der Präsentation von neuen Nachrichten erinnert GoogleWave an einen Instant Messenger. Neue Nachrichten werden umgehen unter dem letzten Beitrag angezeigt. Von der Struktur der Präsentation erinnert GoogleWave wiederum an die eines Forums. Die Waves sind wie Forum-Threads angeordnet und können in einzelne Unterthemen gegliedert werden. Dabei wird jeder Beitrag und jedes Thema chronologisch in einer Wave geordnet. Der Vorteil von GoogleWave ist, dass Nachrichten mit dem Umfang einer E-Mail oder eines Forumbeitrages an alle Beteiligten versendet können und diese umgehend angezeigt werden. Würde hierfür eine Forum oder ein E-Mail-Client verwendet, erhält der Nutzer den neuen Beitrag nur durch Aktualisierung des Browsers beziehungsweise durch das Abrufen der neuen E-Mail. Außerdem existiert zwischen dem Versenden des Beitrages und dem Empfanger des Beitrages eine umbestimmt Zeitspanne. Eine Nachricht könnte innerhalb dieser Zeitspanne bereits an

relevanz verlohren haben oder für alle Beteiligte zu spät eintreffen. Ein abgesendeter Beitrag in GoogleWave wird umgehend für alle Beteiligten angezeigt, ohne zutun der Nutzer. So können flüssig Disskusionen mit einer vielzahl von Beteiligten gleichzeitig geführt werden. Zusätzlich ist der komplette Verlauf der Disskusion zu jeder Zeit, für jeden Beteiligten abrufbar. (s.a. [Goo10b])

GoogleWave besteht aus einem Webangebot und XAMPP-basierenden Protokollen, welche mit dem Open-Source-Framework Google Web Kit entwickelt wurden. GoogleWave kann mit einem HTML5- und AJAX-fähigen Browser verwendet werden. Exemplarisch seien FireFox (ab Version 3.5) und Safari (ab Version 4) genannt. Vielseitige Behauptungen, GoogleWave sei mit dem Android-eigenen Webbrowser nutzbar, können während der Bearbeitungszeit des Dokumentes vom 05.07.2010 bis 06.09.2010 nicht bestätigen werden.
Bei einem Login-Versuch mit dem Android-eigenen Webbrowser auf dem GoogleWave-Server, erhält der Nutzer auf dem Bildschirm folgende Fehlermeldung:

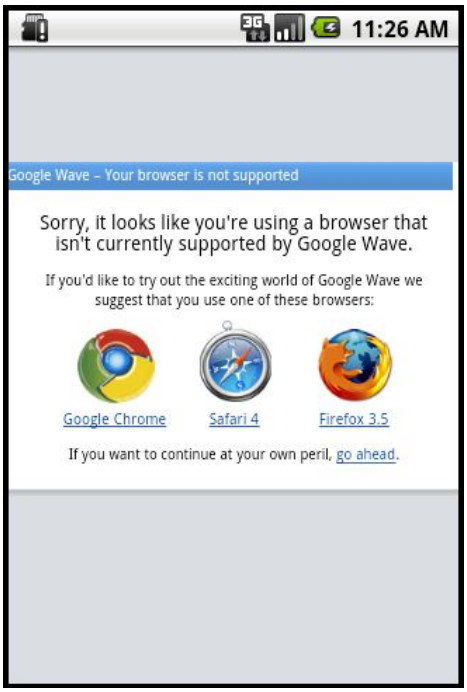

Abbildung 2.2: Bildschirmfoto des Emulators bei dem Versuch GoogleWave mit dem Android-Browser zu erreichen

Dasselbe Verhalten ist auf allen Versionen von Andriod unter Verwendung des Emulators und Android 2.1 Update 1 auf einem Motorola Milestone zu beobachten.

2.2.1 Anwendungsbeispiel von GoogleWave

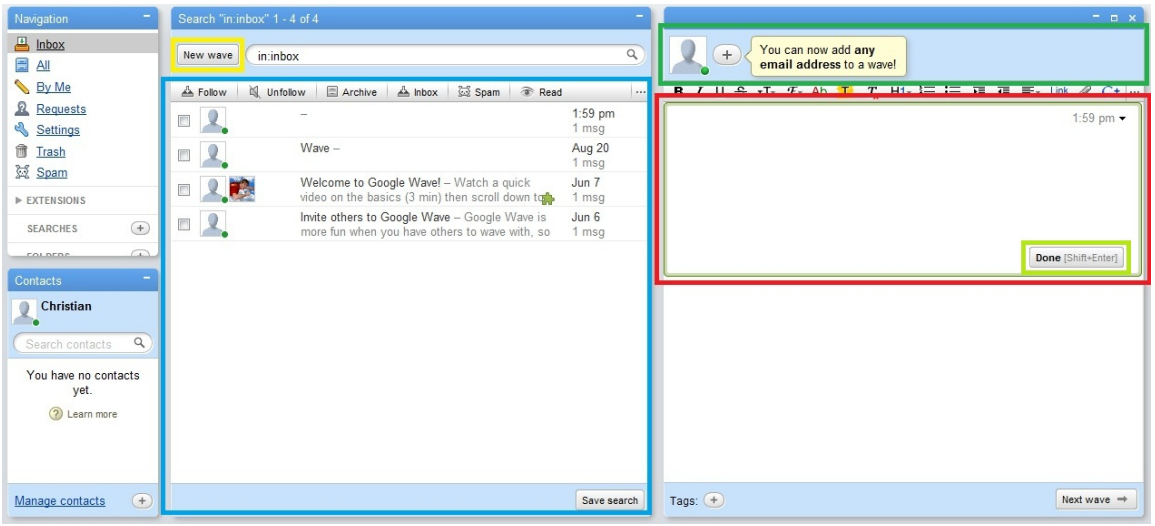

Abbildung 2.3: Startseite von GoogleWave

Im Folgendem wird am Beispiel des Szenarios (s.a. 1.1) die Anwendung von GoogleWave geschildert und näher beschrieben.
Die geschilderte Vorgehnsweise wird mit der Abbildung 2.3 grafisch unterstützt. Im Szenario wurde erwähnt, dass jeder Mitarbeiter der Firma bei GoogleWave einen Account besitzten. Mit Hilfe von GoogleWave kann das Team in der Firma die Teammitglieder auf der Weiterbildung benachrichtigen und ihnen das zu bearbeitende Dokument zusenden.

Zunächst soll eine Wave erstellt werden, um die Teammitglieder auf der Weiterbildung über die Problematik zu informieren.
Ein Teammitglied in der Firma meldet sich an einem gewöhnlichen Computer bei GoolgeWave an. Nach dem Login wird die Startseite von GoogleWave auf dem Bildschirm präsentiert (s.a. Abbildung 2.3). Hier werden dem Teammitglied, neben etlichen Optionen, zunächst alle verfügbaren Waves ausgegeben (blau markiert). Nach dem Betätigen der Schaltfläche „New wave“(gelb markiert) auf der Startseite, stehen dem Anwender zwei neue Eingabefelder zur Verfügung (dunkelgrün und rot markiert). Im oberen werden die Empfänger der Wave eingetragen (dunkelgrün markiert). Im Fall des Szenarios wären dies die Teammitglieder auf der Weiterbildung. Wahlweise können hier Benutzergruppen festgelegt werden, um die Sichtbarkeit der Wave zu definieren. Im größere, darunter liegenden Eingabefeld (rot markiert) wird der erste Beitrag formuliert. Im beschriebenen Szenario wird hier das Problem ausführlich geschildert. Dieser erste Beitrag bildet zugleich den Betreff der Wave und wird wie der Betreff einer E-Mail in der In-Box des GoogleWave-Accounts (siehe blau markierten Bereich) angezeigt.

An jede Wave können beliebig viele Dokumente angehangen werden. Diese Funktion wird nun auch im Szenario in Anspruch genommen. Mit der Betätigung der Schaltfläche „done“ (hellgrün markiert) oder Drücken der Tasten $Shift + Enter$ wird die Wave angelegt. Jedes Teammitglieder in der Empfängerliste wird nun die Wave in seiner In-Box angezeigt bekommen. In der Abbildung 2.3 ist in der In-Box (blau markiert) bereits die neue Wave sichtbar. Würde ein Empfänger hinzugefügt werden, so würde auch seine In-Box mit der neuen Wave aktualisiert.
Jeder Empfänger einer Wave kann Beiträge editieren oder neue anlegen. Ein auf der Weiterbildung befindliches Teammitglied kann so eine Antwort formulieren. Hierfür stehen ihm Textbearbeitungswerkzeuge zur Verfügung. Desweiteren können angehangene Dokumente bearbeitet und neue Dokumente erstellt werden. Die Bearbeitung eines angehangenen Dokumentes ist ohne das Verursachen von Versionskonflikten möglich.
Möchte ein eine Teammitglied eine weitere Person in die Wave einbeziehen, kann er dies durch die Erweiterung der Empfängerliste tun. Es können auch neue Benutzergruppen definiert werden.
Jede auf der Wave ausgeführte Aktion wird umgehen registriert und jedem an der Wave beteiligten Person angezeigt. Die Verarbeitung findet also in Echtzeit statt und erinnert an die Arbeitsweise eines Instant Messengers.
Innerhalb einer bestehenden Wave beziehungsweise Diskussion können neue Unterthemen angelegt werden. Diese Unterthemen besitzen eine eigene Empfängerliste.
Die erwähnten Dokumente können unterschiedlichste Formen annehmen. Es kann sich dabei um Videos, Programmcode, Karten von GoogleMaps, Textdokumente u.s.w. handeln.

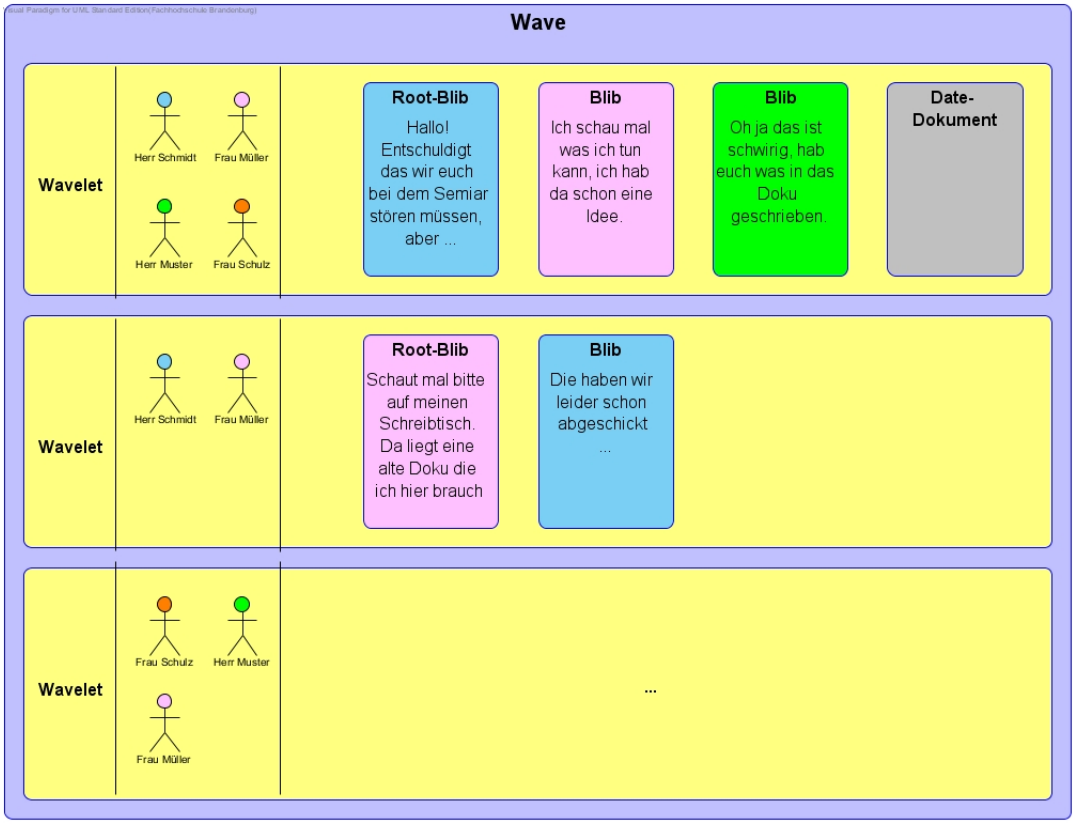

Abbildung 2.4: Aufbau einer Wave

2.2.2 Bestandteile einer Wave

Ergänzend zu den folgenden Erläuterungen zeigt die Abbildung 2.4 (s.a. [Goo10b]) den Aufbau einer Wave . Zusätzlich ist darin das Beispiel aus dem vorherigen Kapitel angedeutet.

Bei einer Wave handelt es sich um eine dynamische Einheit. Eine Wave besteht aus

1. Wavelets (Thema einer Diskusion)
2. Blibs (Nachricht).

Eine Wave beinhaltet zunächst nie selbst Nachrichten oder Dokumenten-Anhänge. Vielmehr ist sie eine Art Rahmen, welcher die Diskussionen thematisch zusammenhält. Auf der Entwicklerseite von GoogleWave [Goo10c] wird eine Wave auch als große Container-Einheit für Wavelets und Blibs, also für die Themen und deren Nachrichten bezeichnet.

Die Funktion einer Wavelet ist mit der eines Forum-Themas zu vergleichen. Hier sind die eigentlichen Nachrichten einer Diskussion zu finden. Jede Wavelet gehört genau zu einer Wave. Der Entwicklerseite [Goo10c] nach ist eine Wavelet die kleinste Container-Einheit einer Wave. Sie ist schlussendlich die Komponente eines Wave welche die Blibs, also Nachrichten

und Daten-Dokumente enthalten. Neben den Blibs und Daten-Dokumenten enthält jede Wavelet eine eigene Empfängerliste. Nur die Person der Empfängerliste können lesend und schreibend auf den Inhalt einer Wavelet zugreifen.
Bei der Arbeit mit einer API wird auf Wavelet-Ebene gearbeitet. Das heißt, es wird nie die Wave selbst angesprochen, sondern immer eine bestimmte Wavelet und deren Inhalt.

Die kleinsten Komponenten einer Wave sind ihre Nachrichten auch Blibs genannt und die Daten-Dokumente. Diese Komponenten sind immer genau einer Wavelet zugeordnen. Wie bereits in Kapitel 2.2.1 erwähnt, können Daten-Dokumente jedes beliebige Format besitzen. Der Entwicklerseite können diesbezüglich keine Einschränkungen entnommen werden.
In Kapitel 2.2.1 wurde das Anlegen einer neuen Wave beschrieben, dabei spielt sich auf dem GoogleWave-Server folgendes ab.

Als erstes wird die Wave, also der Rahmen angelegt. Die eigentliche Konversation wird erst mit der Erzeugung der Wavelet realisiert. Der Wavelet wird die Empfängerliste und die Blibs übergeben. Der erste Blib in einer Wavelet wird als Root-Blib bezeichnet
Im Übrigen existiert grundsätzlich zu jeder Wave mindestens eine Wavelet und ein Blib, der so genannte Root-Blib. Auch wenn eine Wave unabsichtlich erzeugt wird, hat sie immer eine Wavelet und den dazugehörigen Root-Blib. Die neue Wavelet hat dann eine leere Empfängerliste und der Root-Blib keinen Text. (s.a. [Goo10b])

2.2.3 Die GoogleWave APIs

Die GoogleWave-API unterteilt sich zur Zeit[1] in drei separaten APIs mit unterschiedlichen Funktionalitäten:

1. Extensions API (Robots und Gadgets)
    - Robots sind automatisierte Applikationen und werden als Servlets geschrieben. Das hat zur Folge, dass ein Robot auf einen Server geladen werden muss, damit er gestartet werden kann.
      Robots reagieren auf fest vorgegebene Ereignisse in einer Wave. Vom GoogleWave-Server werden sie als eigenständige Benutzer behandelt und müssen in eine Wave geladen werden, damit sie mit ihr interagieren kann.(s.a. [Goo10d])
    - Gadgets ermöglichen das Einfügen von nicht-Wave-Elementen in eine Wave. Beispielsweise kann eine Flash-Anwendung in einer Wave integriert und ausgeführt werden. (s.a. [Goo10e])

[1] Stand: 21.Juli 2010

2. Embedded Waves API
    - Diese API kann genutzt werden, um den Inhalt einer Wave auf einer beliebigen Internetseite anzuzeigen.
    - Alle Operationen dieser API haben ausschließlich Leserechte. Es kann der Inhalt einer Wave nur angezeigt, aber nicht verändert werden.(s.a. [Goo10f])
3. Wave Data API
    - Die Funktionen dieser API ermöglichen das Implementieren eines alternativen GoogleWave-Clients. Das heißt, es kann ein eigener Internetdienst entwickelt werden, der auf den GoogleWave-Server lesend und schreibend zugreifen kann.(s.a. [Goo10c])

2.3 Moderne Mehr-Schichtenarchitektur

Die moderne Mehr-Schichtenarchitektur ist ein Strukturierungsprinzip für Architekturen von Softwaresystemen. Die Mehr-Schichtenarchitektur ist nicht mit der ISO-Schichtenmodell zu verwechseln.
Durch die Anwendung des Strukturierungsprinzips werden die einzelnen Komponenten einer Software, nach Aufgabenfelder getrennt, Schichten beziehungsweise Layer zugewiesen. Zum Beispiel werden alle Komponenten eines Programmes, welche für die Bildschirmausgabe zuständig sind, dem *Presentation Layer* zugeordnet.
Jede Schicht ist mit ihren Komponenten von anderen Schichten getrennt. Es wird von einer sogenannten „Kapselung“ der Schichten gesprochen. Die Komponenten des *Presentation Layer* sind also von anderen Komponenten der Software, welche ihnen Daten bereitstellen gekapselt.
Damit die Komponenten des *Presentation Layer* die benötigten Daten für die Bildschirmausgabe erhalten, werden Schnittstellen zwischen den Schichten implementiert. Schnittstellen werden durch „Getter“- oder „Setter“-Klassen realisiert. Mit Hilfe dieser Klassen können die Schichten die jeweils benötigte Daten untereinander Austauschen.
In wie viele Schichten die Komponenten einer Software unterteilt werden ist prinzipiell nicht vorgeschrieben. Es hängt davon ab wie fein der Entwickler die Komponenten voneinander kapselt. Für gewöhnlich kommt die Drei-Schichtenarchitektur zum Einsatz. Demnach werden die Komponenten in die drei Schichten (siehe Abbildung 2.5, [And10]):

- Presentation Layer (Präsentationsschicht)
- Business Layer (Anwendungsschicht)

- Data Access Layer (Persistenzschicht)

untergliedert. Der *Presentation Layer* beinhaltet alle Komponenten, welche für die Bildschirmausgabe zuständig sind. Im *Business Layer* sind alle Komponenten der Geschäftslogik einer Software gekapselt. Mit dem *Data Access Layer* erhält die Software Zugriff auf Datenbanken, Funktionen anderer Programm oder Internetservices. Soll die Anwendung auf eine größere Datenbank oder auf einen Internetservice zugreifen, so kann eine vierte Schicht hinzugefügt werden, welche die jeweilige Datenbank beziehungsweise Internetservice repräsentiert. Diese zusätzliche Schicht wird *Data Store* (Data Layer) genannt. Sie ist zwar Teil des Gesamtkonzeptes aber nicht Bestandteil der eigentlichen implementierten Software. Während der Entwurfsphase einer Software können mit der zusätzlichen Schicht, Konzepte zum Beispiel für den Datenaustausch klarer Aufgezeigt werden.

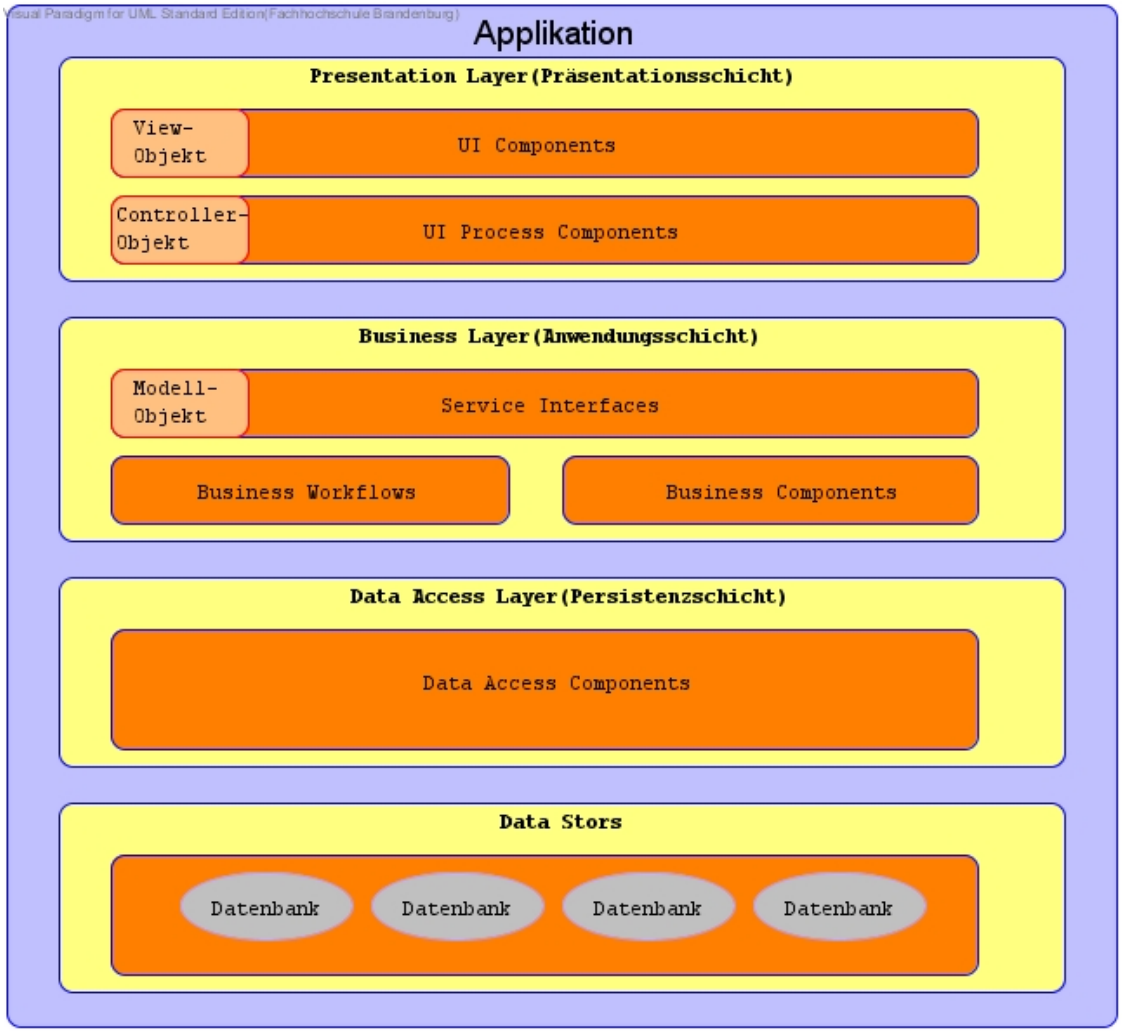

Abbildung 2.5: Moderne Mehr-Schichtenarchitektur

Für den Entwickler einer Software ergibt sich durch die Nutzung des Strukturierungsprinzips der Vorteil notwendige Änderungen am Programmcode effektiver und schneller durchzuführen. Das heißt, wenn sich zum Beispiel das Datenbankschema des *Data Stores* ändert, muss der Entwickler nur den *Data Access Layer* anpassen. Die darüber liegenden Schichten bleiben völlig unverändert. Ist es notwendig den gesamten *Data Access Layer* durch einen neuen zu ersetzten, muss der Entwickler nur darauf achten, dass er die bestehenden Schnittstellen zu den anderen Schichten beibehält, oder diese abändert. Auf die implementiert Logik der Schichten hat das Auswechseln des *Data Access Layer* keine weiteren Auswirkungen.
Ein weiterer Vorteil ergibt sich hinsichtlich der Wiederverwendbarkeit einer Schicht. Findet zum Beispiel der *Business Layer* einer Software auch bei andere Softwareprodukten Verwendung, kann dieser unter Berücksichtigung der Schnittstellen in anderen Softwareprodukten wieder verwendet werden.

Wenn die moderne Mehr-Schichtenarchitektur eingesetzt wird, ist auf folgende Regeln zu achten.

- Eine Schicht j stellt der Schicht $j+1$ Daten beziehungsweise Dienste zur Verfügung. Zum Beispiel stellt der *Business Layer* dem *Presentation Layer* die aus der Geschäftslogik resultierenden Anwendungsdaten zur Verfügung.
- Eine Schicht j „kennt“ die darüber liegende Schicht $j + 1$ und ihren Funktionsumfang nicht. Zum Beispiel „weiß“ der *Business Layer* nicht, dass er einem *Presentation Layer* die Anwendungsdaten übergibt. Für die Funktionsweise des *Business Layer* spielt dies auch keine Rolle.
- Eine Schicht j deligiert Teilaufgaben an die Schicht $j - 1$. Der *Business Layer* „kennt“ den *Data Access Layer* und weist ihn an, Daten für die Geschäftslogik aus einer Datenbank zu lesen.
- Eine Schicht j kann nicht direkt Daten von einer Schicht $j + 2$ oder $j - 2$ anfordern. Der *Presentation Layer* darf den *Data Access Layer* also nicht direkt anweisen Daten für die Bildschirmausgabe aus der Datenbank zu schreiben. In diesem Vorgang muss der *Business Layer* miteinbezogen werden.

2.3.1 MVC-Paradigma

Das MVC-Paradigma ist ein Architekturmuster. Ein Architekturmuster unterscheidet sich von einem Entwurfsmuster hinsichtlich seiner Aufgaben. Ein Entwurfsmuster stellt eine bewährt generische Lösung für ein immer wiederkehrendes Problem in der Softwarearchitektur oder Softwareentwicklung dar. Entwurfsmuster befassen sich also mit der konkreten Lösung eines Problemes. Ein Architekturmuster bestimmt die Organisation und Zusammenarbeit von Komponenten einer Software. Ursprünglich wurde es für Smaltalk-80 entwickelt. Die Umsetzung wurde hierfür mit drei Klassen realisiert. Heute findet es bei anderen Softwaresystemen ebenfalls Anwendung. Die Unterteilung in genau drei Klassen wird dabei nicht mehr verwendet. Vielmehr wird das Konzept von MVC für den Entwurf einer Software angewendet. Die Umsetzung kann daher in mehrere Module aufgeteilt werden. Das MVC-Konzept dient zur Konstruktion von Benutzerschnittstellen und wird durch drei Klassen realisiert:

1. Model-Klasse
2. View-Klasse
3. Controller-Klasse

Die *Modell*-Klassen dienen zur Erfassung von Anwendungsdaten und sind im *Business Layer* (Anwendungsschicht) zu finden. Die *View*-Klasse definiert das Layout einer Bildschirmpräsentation und werden im *Presentation Layer* (Presentationsschicht) untergebracht. Die *Controller*-Klassen dienen für die Kommunikation zwischen den *Modell*-Klassen und den *View*-Klassen. Auf diese Weise kann die Benutzerschnittstelle auf Benutzereingaben reagieren. Die *Controller*-Klassen sind ebenfalls im *Presentation Layer*(Präsentationsschicht) untergebracht.
Durch das MVC-Paradigma/Konzept werden die drei genannten Klassen entkoppelt. Dies bewirkt die Flexibilität und Wiederverwendbarkeit der Komponenten.

3 Analyse und Design

3.1 Anforderungen

3.1.1 Anwendungseinsatz

Die Applikation *AndroidWave* ist prinzipiell für jeden GoogleWave-Nutzer konzipiert, der im Besitz eines mobilen Endgerätes mit dem Betriebssystem Android ist. Weitere Auswahlkriterien müssen für diese Applikation im Bezug auf die Benutzergruppe nicht spezifiziert werden. Diese Applikation bietet dem Nutzer maximale Flexibilität und die Unabhängigkeit von herkömmlichen Computern im Hinblick auf die Nutzung ihres GoogleWave-Accounts. *AndroidWave* kann immer dann zum Einsatz kommen, wenn eine Verbindung zum Internet hergestellt werden kann, unabhängig davon, ob diese durch ein Mobilfunkstandart oder einem WLAN-Netz zustande gekommen ist.

3.1.2 Anwendungsfunktionen

Im Folgendem werden alle primären Funktionen vorgestellt, welche für die Applikation *AndroidWave* implementiert werden sollen:

1. Login bei GoogleWave
2. Erstellen einer neuen Wave
3. Abrufen und Anzeigen von verfügbaren Waves
4. Erweitern der Waves durch eigene Kommentare
5. Zugriff auf Kontaktdaten

Die Funktion „Login bei GoogleWave“ erlaubt dem Nutzer der Applikation den Zugriff auf dessen GoogleWave-Account. Diese Funktion wird nicht direkt vom Nutzer ausgeführt. Vielmehr wird sie ein Bestandteil jeder Funktion sein, welche den Zugriff auf den GoogleWave-Server benötigt. Das heißt, die Applikation wird nur dann auf den Internetdienst zugreifen

wenn dies erforderlich ist. Dieses Vorgehen hat, angesichts der begrenzten Hardware- und Energie-Ressourcen, eine positive Auswirkung auf die Lebensdauer des Akkus und die Auslastung des Systems. Die notwendigen Daten für den erfolgreichen Login sind Nutzername und Passwort. Sie werden persistent auf dem Gerät des Clients gespeichert.

Die Funktion zum „Erstellen einer Neuen Wave“ muss separat von der Funktion „Abrufen und Anzeigen von verfügbaren Waves“ zugänglich sein. Dies erfolgt vor allem aus Gründen der Performance und der damit einhergehenden Einsparung von Energie und zum anderen der Übersicht halber. Mit dieser Funktion wird eine neue Wave mit dazugehörigem Wavelet und Root-Blib auf dem GoogleWave-Server generiert.

Mit der Funktion „Abrufen und Anzeigen von verfügbaren Waves“ werden zunächst alle verfügbaren Waves aus der In-Box des jeweiligen GoogleWave-Accounts auf das mobile Endgerät heruntergeladen und angezeigt. Das Wechseln zwischen den einzelnen Waves und die Anzeige ihrer Inhalte, also die Auflistung in Wavelets und Blibs ist für die Flexibilität der Applikation und die Transparenz der Informationen erforderlich.

Die letzte wichtige Funktionalität der Applikation ist das Ergänzen beziehungsweise Erstellen von Blibs in einem bestehenden Wavelet und das Erzeugen eines neuen Wavelets mit dem dazugehörigen Root-Blib. Somit wird die Grundfunktionalität der Applikation komplettiert und eine vollwertige Konversation beziehungsweise Nutzung von GoogleWave gewährleistet.

Der Zugriff auf die Kontaktdaten der auf dem Endgerät gespeicherten Personen stellt ein Feature der Applikation dar. Die Empfänger eines Wavelets werden in einem entsprechenden Textfeld eingetragen. Jeder Empfänger wird durch seine Email-Adresse identifiziert. Mit Hilfe dieser Funktion wird das Ausfüllen der Empfängerliste vereinfacht. Diese Funktion listet die Email-Adressen auf und kopiert sie nach der Selektion in die Empfangsliste.

3.1.3 Abgrenzungskriterien

Da diese Applikation nur einen elementaren Funktionsumfang für die Nutzung von GoogleWave besitzt, werden folgende Funktionen nicht unterstützt. Sie werden nicht als elementar betrachtet und daher nicht umgesetzt.

- Datei anhängen:
  Das Durchsuchen des Systemverzeichnisses und Anhängen von Dateien als Daten-Dokumenten an bestehende oder zu erzeugende Wavelets wird in dieser Version der Applikation nicht realisiert.
- Herunterladen von angehängten Dateien:

Analog zum Anhängen von Dateien ist auch das Herunterladen von Daten aus Daten-Dokumenten nicht realisiert.

- GoogleWave-Account erstellen:
  Das Erstellen eines GoogleWave-Accounts soll mit dieser Applikation nicht unterstützt werden. Ein bereits erstellter GoogleWave-Account ist Voraussetzung für die Nutzung der Applikation *AndroidWave*.
- Aufbauen einer Internetverbindung:
  Einstellungen für eine Internetverbindung beziehungsweise das Erzeugen einer Internetverbindung sollen nicht Gegenstand der Implementation werden. Die Verbindung mit dem Internet sollte bereits vollzogen sein. Entsprechende Applikationen zum Einrichten von Internetverbindungen sind Teil des Android-Grundsystems. Diese Applikation soll sich daher nur bestehende Verbindungen zu nutze machen.

3.1.4 Anwendungsdaten

Alle Daten einer verfügbaren Wave sind auf dem GoogleWave-Server persistent gespeichert. Diese werden ausschließlich für die Präsentation und Bearbeitung auf den Client herunter geladen. Wie bereits im Kapitel 2.2 beschrieben sind Wavelets mit ihren Blibs Bestandteile einer Wave. Es wird nun geklärt welche Informationen nun genau zu einer Wave beziehungsweise Wavelet gespeichert werden. Jedes Wavelet besitzt eine Referenz zu dem übergeordneten Container-Element der Wave. Des weiteren wird für jede Wavelet eine Empfängerliste gespeichert. In dieser sind alle GoogleWave-Nutzer mit ihren Email-Adressen aufgelistet, welche Zugriff auf diese Wavelet besitzen sollen.

Ein Blib setzt sich aus einer Referenz zu dem übergeordneten Container-Element Wavelet, dem Namen des Verfassers, der eigentlichen Nachricht, dem Erstellungsdatum und dem Datum der letzten Änderungen zusammen.

Für den Login auf dem GoogleWave-Server werden Angaben zum Nutzernamen und dem dazugehörigen Passwort benötigt. Damit die Informationen nicht bei jedem Start des Programms erneut eingetragen werden müssen, bietet Android die Möglichkeit so genannte Preference-Daten in eine eigens dafür existierende Datenbank persistent zu speichern. Diese Datenbank ist bereits auf einem Android-System vorhanden und muss nicht eigenhändig erstellt werden [uMP09d]. Der Nutzername und das Passwort werden nach der Installation des Programms einmal eingetragen und steht jederzeit für den Zugriff auf den GoogleWave-Server zur Verfügung. Bei Bedarf können diese Angaben geändert werden.

Informationen zu Kontakten können der Kontakt-Datenbank entnommen werden. Die Datenbank ist fester Bestandteil eines Android-Systems. Die Informationen in der Kontakt-Datenbank bestehen aus dem Vor- und Zunamen eines Kontaktes, sowie der Telefonnummer und der Email-Adresse. Für die Vollständigkeit der Informationen ist der Nutzer des Android-Systems verantwortlich. Mit entsprechenden Funktionen kann auf die Datenbank zugegriffen und entsprechende Email-Adressen extrahiert werden.

3.1.5 Benutzeroberfläche

Die Bedienung der Benutzeroberfläche erfolgt hauptsächlich mit dem Touchscreen. Für Text-Eingaben kann optional die Hardware-Tastatur, sofern vorhanden, verwendet werden. Andernfalls steht die Android-Touchscreen-Tastatur für diese Eingaben zur Verfügung.

Die Bedienelemente sind groß und übersichtlich angeordnet. Mögliche Fehleingaben sollen verhindert werden. Ein übersichtliches Design trägt darüber hinaus zur schnellen Erfassung der notwendigen Informationen bei.

3.1.6 Anwendung der Drei-Schichtenarchitektur und des MVC-Paradigmas

Die Applikation ist nach dem Prinzip der Drei-Schichtenarchitektur und dem MVC-Paradigma zu implementiert. Da sich die APIs von Andorid und GoogleWave nach wie vor in der Entwicklung befinden, sind Änderungen an verwendeten Funktionen absehbar. Im Hinblick auf die Wartung, also die Anpassung der Applikation an einer veränderten API, beziehungsweise die Erweiterung von *AndroidWave* durch neue Funktionen ist ein derartiges Vorgehen unerlässlich.

3.2 Architektur

In den folgenden Kapiteln werden grundlegende Aspekte der Softwarearchitektur von *AndroidWave*, im Sinne der OOA[1] und der OOD[2] aufgezeigt.
Somit wird eine entscheidende Phase der Implementierung von *AndroidWave* dokumentiert.

Ein Smartphone ist zwar ein performantes mobiles Endgerät, dennoch verfügt es über begrenzte System-Ressourcen. Ist eine Applikation ohne entsprechendes Konzept programmiert, werden die physikalischen Grenzen eines Smartphones relativ schnell ausgereizt.

Bei der Programmierung von *AndroidWave* ist also darauf zu achten, dass sie so effektiv, das heißt so ressourcenschonend wie nur möglich entwickelt wird. Dadurch ist eine sichere beziehungsweise stabile Programmlaufzeit des Betriebssystems garantiert. Selbstverständlich darf die Qualität der Applikation nicht unter dieser Motivation leiden. Das heißt, der realisierte Funktionsumfang von Android muss trotz der ressourcenschonenden Programmierung, den Anforderungen aus Kapitel 3.1 gerecht werden.

Da ein Android-Betriebssystem Multitasking fähig ist, kann davon ausgegangen werden, das neben *AndroidWave* noch andere Applikationen zeitgleich gestartet sind. Das heißt, es muss nicht nur auf die stabile Laufzeit des Betriebssystems Rücksicht genommen werden, sondern auch auf die Laufzeit der bereits ausgeführten Applikationen.
Aus diesem Grund muss ein intensives Software-Engineering für *AndroidWave* betrieben werden.

Als Resultat wird ein entsprechendes Konzept für *AndroidWave* entstehen. Der Vorteil dieses Verfahrens liegt darin, dass alle Entscheidungen bezüglich der Architektur von *AndroidWave* getroffen sind, bevor die eigentliche Programmierung beginnt. Dadurch gestaltet sich die Realisierung wesentlich einfacher und übersichtlicher und die termingerechte Fertigstellung der Applikation wird nicht, durch wiederholte Umstrukturierungsmaßnahmen auf Grund eines fehlenden Gesamtkonzeptes, gefährdet.

3.2.1 Entwicklungsstand der Wave Data API

Die Wave-Data-API ist eine der drei Teil-APIs von GoogleWave und für die Implementierung von externen Clients konzipiert. Unter externen Clients wird im Zusammenhang mit GoogleWave ein Internet-Service verstanden, welche auf den GoogleWave-Server zugreifen

[1] objektorientierte Analyse
[2] objektorientiertes Design

kann, aber nicht von GoogleWave entwickelt wurde. Es ist eine API für die Entwicklung von GoolgeWave-Clients im zusammenspiel mit dem GoolgeWave-Server.
Für die Umsetzung von *AndroidWave* ist diese API daher, neben dem Android Software-Development-Kit, die wichtigste Komponente für eine erfolgreiche Implementierung.
Tatsächlich aber ist der Umfang der API zum Zeitpunkt der Erstellung dieser Bachelorarbeit[3] für die Realisierung von *AndroidWave* ungeeignet. Die API beinhaltete drei Funktionen:

1. Auslesen der In-Box, das heißt Abrufen aller verfügbaren Waves
2. Ändern des Zustandes einer Wave auf „gelesen“, „ungelesen“, „ignorieren“ und „verwerfen“
3. Suche nach einer bestimmten Wave.

GoogleWave kündigt [Goo10c] an, dass die betreffende API in absehbarer Zeit nicht erweitert wird. Stattdessen verweist der Entwickler auf die Extensions-API als einzige Lösung für einen externen Client.

Das Nutzen eines Robots als Proxy zwischen dem GoogleWave-Server und dem Client erweist sich bereits im Vorfeld einer Betrachtung als aussichtslos. Zum einen fehlen dem Andorid Software-Development-Kit Funktionen für die Kommunikatiomit einem Sevlet, zum anderen wird ein Robot vom GoogleWave-Server immer als eigenständiger Account behandelt. Dem Robot ist es nicht möglich die Identität eines „echten“ GoogleWave-Accounts anzunehmen. Ein Robot ist also für diese Zweck nicht konzipiert und stellt daher in keiner Weise einen Ersatz dar.
Angenommen die Implementation von *AndroidWave* würde, ungeachtet der genannten Kommunikationsschwierigkeiten mit der Extensions-API erfolgen, so müsste jeder Nutzer von *AndroidWave* einen „echten“ GoogleWave-Account und einen „Robot-Account“ besitzen. Mit dem „echten“ GoogleWave-Account ist die Arbeit an einer Wave von einem gewöhnlichen Computer möglich. Der „Robot-Account“ würde zum Einsatz kommen, wenn der Nutzer *AndroidWave* verwenden muss. Pro Nutzer ergäben sich jeweils zwei Account. Für die Empfängerliste bedeutet dies im ungünstigsten Falle eine Verdoppelung der Einträge. Analog dazu ergäbe sich folgendes Problem:
Wird beispielsweise die Einladung des Robot-Accounts eines Nutzers versäumt, könnte er nicht mit *AndroidWave* an einer Wave teilnehmen. Der betreffende Nutzer bliebe solange von der Wave „ausgesperrt“, bis die Einladung nachgeholt wird. Hinzu käme eine Unübersichtlichkeit innerhalb einer Wave, wenn ein Nutzer beide Arten der Accounts abwechselnd nutzen muss. Angesichts der unbrauchbaren API und der genannten Probleme bei der Verwendung

[3] 05.07.2010

von Robots als Alternative ist eine Umsetzung von AndroidWave in dem Zeitrahmen der Bachelorarbeit nicht möglich. Dem folgenden Kapitel ist das weitere Vorgehen für eine erfolgreiche Umsetzung zu entnehmen.

3.2.1.1 Der GoogleWave-Dummy

Da die API Wave-Data-API-APP von GoogleWave bis auf weiteres für die Implementierung ungeeignet ist, wird der Zugriff auf GoogleWave für diese Arbeit simuliert. Auf diese Weise können die Grundfunktionen des Clients umgesetzt werden, ohne auf die Fertigstellung der API warten zu müssen. Die serverseitigen Aufgaben werden stellvertretend von einem lokalen XAMPP-Server übernommen. Auf dem Server wird eine MySQL-Datenbank angelegt. Das Schema ist der Struktur des „reellen“ GoogleWave mit allen Komponenten nachempfunden. So kann die grundlegende Funktionalität von *AndroidWave* implementiert und präsentiert werden. Dabei werden IT-Sicherheitsrelevante Aspekte hinsichtlich der Entwicklung des GoogleWave-Dummys nicht betrachtet. Zum Beispiel erfolgt der Datenaustausch zwischen *AndroidWave* und dem GoogleWave-Dummy im Klartext.
Wird die betreffende API erweitert und die notwendigen Funktionen bereitgestellt, kann dank der verwendeten Drei-Schichtenarchitektur und der Nutzung des MVC-Paradigmas mit relativ geringem Aufwand die Persistenzschicht ausgetauscht werden. Der Austausch reduziert sich deshalb auf diese eine Schicht, da die Persistenzschicht den eigentlichen Zugriff auf den GoogleWave-Server realisiert. Das heißt, nur in dieser Schicht würde die GoogleWave API Wave-Data-API ihre Anwendung finden. Alle darüber liegenden Schichten werden sorgfältig gekapselt. Da der GoogleWave-Dummy dem Original nachempfunden ist, kommt es nur zu minimalen Anpassungsarbeiten an den Schnittstellen.

3.2.2 Analyse

3.2.2.1 Anwendungsfälle

Ein Use-Case-Diagramm stellt einen groben Umriss des zu entwickelnden Programmes dar. Die Diagramme werden dabei aus Sicht des Anwenders modelliert. Sie zeigen also nicht den gesamten Funktionsumfang einer Anwendung in voller Breite.
Use-Case-Diagramme werden vorwiegend zu Beginn eines Softwareprojektes erstellt und bestehen aus einem Diagramm und einer Tabelle. Die Tabelle beschreibt den Anwendungsfall und ist das wichtigste Element dieser Modellierung.
Use-Case-Diagramme geben dem Entwickler erst Aufschlüsse über konzeptionellen Klassen und bieten einen ersten Blick auf den Ablauf des Programms. Des weiteren entstehen, auf Grund der Diagramme, erste Überlegungen bezüglich des Grafikinterfaces.

Die Abbildung 3.1 zeigt in einem Use-Case-Diagramm typische Anwendungsfälle, die für den Nutzer bei der Verwendung von *AndroidWave* eintreten.

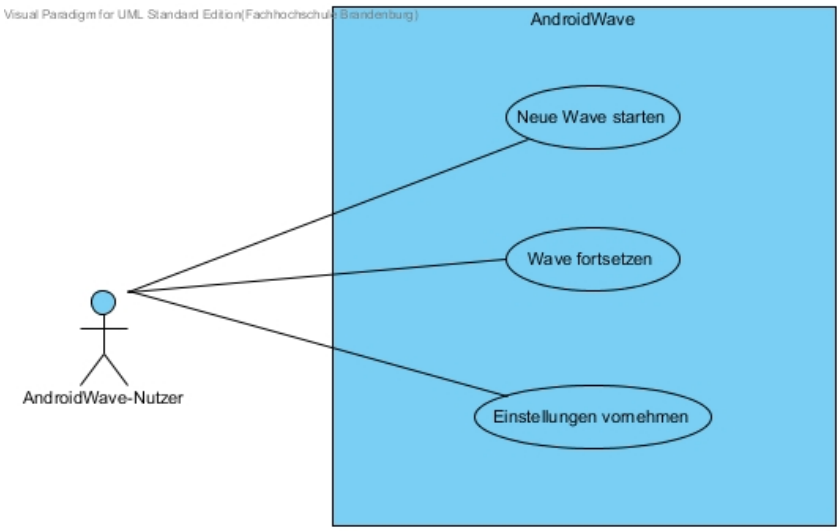

Abbildung 3.1: Use-Case-Diagramm: Allgemeine Funktionsübersicht

Der einzige Akteur des *AndroidWave*-Systems ist der Nutzer selbst. Gemäß der Anforderungen aus Kapitel 3.1.2 kann der Nutzer von *AndroidWave* folgende Anwendungsfälle auslösen:

1. „Neue Wave starten“
2. „Einstellungen vornehmen“
3. „Wave fortsetzen“

In den folgenden Abschnitten werden die genannten Anwendungsfälle im Einzelnen beschrieben.

Einstellungen vornehmen Mit Abbildung 3.2 wird der Anwendungsfall „Einstellung vornehmen“ genauer dargestellt.

In dem Anwendungsfall nimmt der Nutzer Einstellungen für die Applikation vor. Der Auslöser des Anwendungsfalles findet sich im Hauptmenü der Anwendung.
Während des Anwendungsfalles, kann der Nutzer aus drei Optionen wählen. Ihm stehen folgende Optionen zur Wahl:

1. Nutzername eingeben
2. Passwort eingeben
3. Serveradresse eingeben

Nutzername, Passwort und Serveradress werden für den Login auf der Datenbank des GoogleWave-Dummys benötigt. Die Daten werden persistent auf einer Datenbank des Betriebssystems gespeichert. Wird die Applikation neu gestartet werden die Optionen aus der Datenbank für die Applikation bereitgestellt.

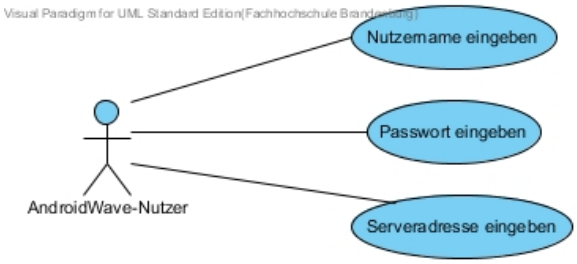

Abbildung 3.2: Use-Case-Diagramm: Einstellungen vornehmen

Tabelle 3.1: Use-Case Einstellungen vornehmen

Name	Einstellung vornehmen
Kurzbeschreibung	Der Nutzer trägt seine Verbindungsdaten für den GoogleWave-Dummy ein.
Akteure	AndroidWave-Nutzer
Auslöser	Betätigung einer entsprechenden Schaltfläche im Hauptmenü.
Vorbedingung	Der Datenbank-Account des GoogleWave-Dummys ist vorhanden.
Ergebnis	Nutzername, Passwort und Serveradresse werden persistent gespeichert. Die Login-Daten werden von der Anwendung zum Anmelden auf dem GoogleWave-Dummy genutzt.
Nachbedingung	Login-Daten sind persistent gespeichert. Änderungen können vorgenommen werden, wenn der Nutzer nicht auf dem GoogleWave-Dummy angemeldet ist.
Essenzieller Ablauf	1. Schaltfläche im Hauptmenü getätigt 2. Eingabe des Nutzernamens 3. Eingabe des Passwortes 4. Eingabe der Serveradresse 5. Bestätigung des Änderung

Neue Wave starten Mit der Abbildung 3.3 wird der Anwendungsfall „Neue Wave starten“ näher beschreiben.

Voraussetzung für das Eintreten des Anwendungsfalles ist das Vorhandensein einer Internetverbindung.
Nach dem Auslösen des Anwendungsfalles durch den Nutzer, können Empfänger eingetragen und ein Beitrag formuliert werden. Der geschriebene Beitrag dient der Wave als Betreffzeile. Die Empfänger können, manuell oder durch die Nutzung der Kontaktdaten, in die Empfängerliste eingetragen werden. Auf die Kontaktdaten kann mit Hilfe einer Funktion von AndriodWave zugegriffen werden. Die Funktion lädt die Daten aus der Kontaktdatenbank des Systems, stellt sie dem Nutzer zur Wahl und kopiert das selektierte Element in die Empfängerliste. Zur Wahl stehen nur die E-Mail-Adressen. Ist für ein Kontakt keine E-Mail-Adresse gespeichert, so wird der Kontakt auch nicht angezeigt.
Sind alle Einträge vorgenommen, wird die neue Wave an den GoogleWave-Dummy gesendet.

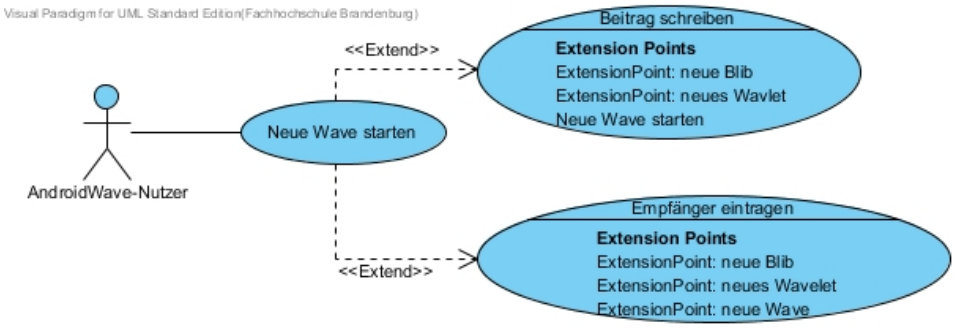

Abbildung 3.3: Use-Case-Diagramm: Neue Wave starten

Tabelle 3.2: Use-Case Neue Wave starten

Name	Neue Wave starten
Kurzbeschreibung	Der Nutzer kann eine neue Wave erzeugen und auf die Datenbank des GoogleWave-Dummys laden
Akteure	AndroidWave-Nutzer
Auslöser	Der Nutzer betätigt die Schaltfläche „neue Wave starten“
Vorbedingung	Eine Internetverbindung ist hergestellt. Der Nutzer hat ein Benutzerkonto beim GoogleWave-Dummy.
Ergebnis	Eine neue Wave wird auf den GoogleWave-Dummy geladen. Die Empfänger bekommen eine Bestätigung zugeschickt.
Nachbedingung	Der Nutzer gelangt zum Startbildschirm zurück.
Essenzieller Ablauf	1. Schaltfläche „Neue Wave starten“ wird betätigt 2. Empfänger werden eingetragen 3. der erste Beitrag der Wave wird formuliert 4. Bestätigung der erzeugten Wave 5. neue Wave wird auf den Server geladen 6. der Akteur erhält eine Benachrichtigung über Miss-/Erfolg.

Wave fortsetzten Die Abbildung 3.4 beschreibt den Anwendungsfall „Wave fortsetzten“.

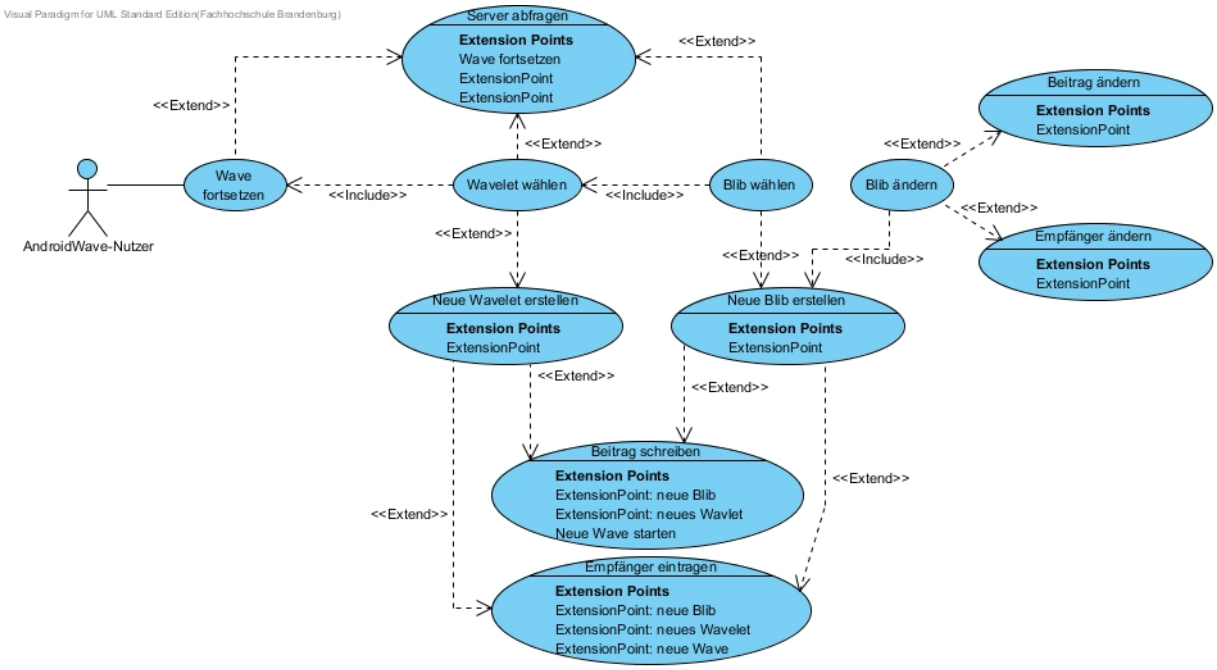

Abbildung 3.4: Use-Case-Diagramm: Wave fortsetzten

Voraussetzung für das Eintreten des Anwendungsfalles ist das Vorhandensein einer Internetverbindung.

Nachdem der Nutzer den Anwendungsfall ausgelöst hat, generiert die Anwendung eine Liste der verfügbaren Waves vom GoogleWave-Dummy. Der Nutzer kann nun zwischen ihnen wählen.

Nach der Selektion einer Wave läd *AndroidWave* alle dazugehörigen Wavelets vom GoogleWave-Dummy. Nun kann der Nutzer entweder eine Wavelet wählen um zu seinen Blibs zu gelangen oder einen neuen Wavelet erzeugen. Dies erzeugt zu dem neuen Wavelet einen neuen Blib, dem Root-Blib der Wavelets. Es wird wie beim UseCase „neue Wave starten“ die Empfängerliste ausgefüllt und ein Beitrag geschrieben. Nach Senden des Wavelets wird dieser mit seinem Root-Blib in die Datenbank des Dummys geladen.

Wählt der Nutzer stattdessen ein bestehendes Wavelet aus, gelangt er zu dessen Blibs. Er kann nun entweder einen neuen Blib in diesem Wavelet erstellen oder einen bestehenden Blib ändern. Ändert er einen Blib werden ihm die Empfängerliste und der Text der Blib angezeigt. Ist die Blib editiert, wird sie zum GoogleWave-Dummy zurück geschickt. Soll ein neuer Empfänger eingeladen werden, muss nur eine der Blibs ausgewählt und anschließrnd die Empfängerliste geändert werden. Wird ein neuer Blib erstellt, ist die Empfängerliste bereits ausgefüllt. Sie kann dennoch durch neue Teilnehmer ergänzt werden. Abschließend

wird das Textfeld für den Beitrag ausgefüllt. Nach dem Absenden der Blib wird sie in die Datenbank des GoogleWave-Dummys geschrieben.

Tabelle 3.3: Use-Case Wave fortsetzten

Name	Wave fortsetzen
Kurzbeschreibung	Der Nutzer kann aus vorhandenen Waves, Wavelets und Blibs auswählen und diese mit Kontakte und/oder Beiträge erweitern.
Akteure	AndroidWave-Nutzer
Auslöser	Der Nutzer hat die Schaltfläche „Wave fortsetzen“ betätigt
Vorbedingung	Eine Internetverbindung ist hergestellt. Der Nutzer hat ein Benutzerkonto beim GoogleWave-Dummy.
Ergebnis	Der Nutzer hat den Inhalt der Wave lesen können und bei Bedarf erweitert.
Nachbedingung	Änderungen sind auf den GoogleWave-Dummy geladen. Der Nutzer bleibt in dem UseCase und kann erneut aus den vorhanden Waves auswählen.
Essenzieller Ablauf	1. Schaltfläche „Wave fortsetzen“ wird betätigt 2. Auswahl einer Wave 3. Wavelets der selektierten Wave werden angezeigt 4. eventuell neue Wavelet erzeugen 5.1 eventuell Wavelet selektieren 5.2 Blibs des selektierten Wavelets werden angezeigt 5.3 eventuell bestehende Blib editieren 5.4 eventuell neue Blib erstellen 6. Änderungen werden auf den GoogleWave-Dummy geladen

3.2.2.2 Domänenmodell

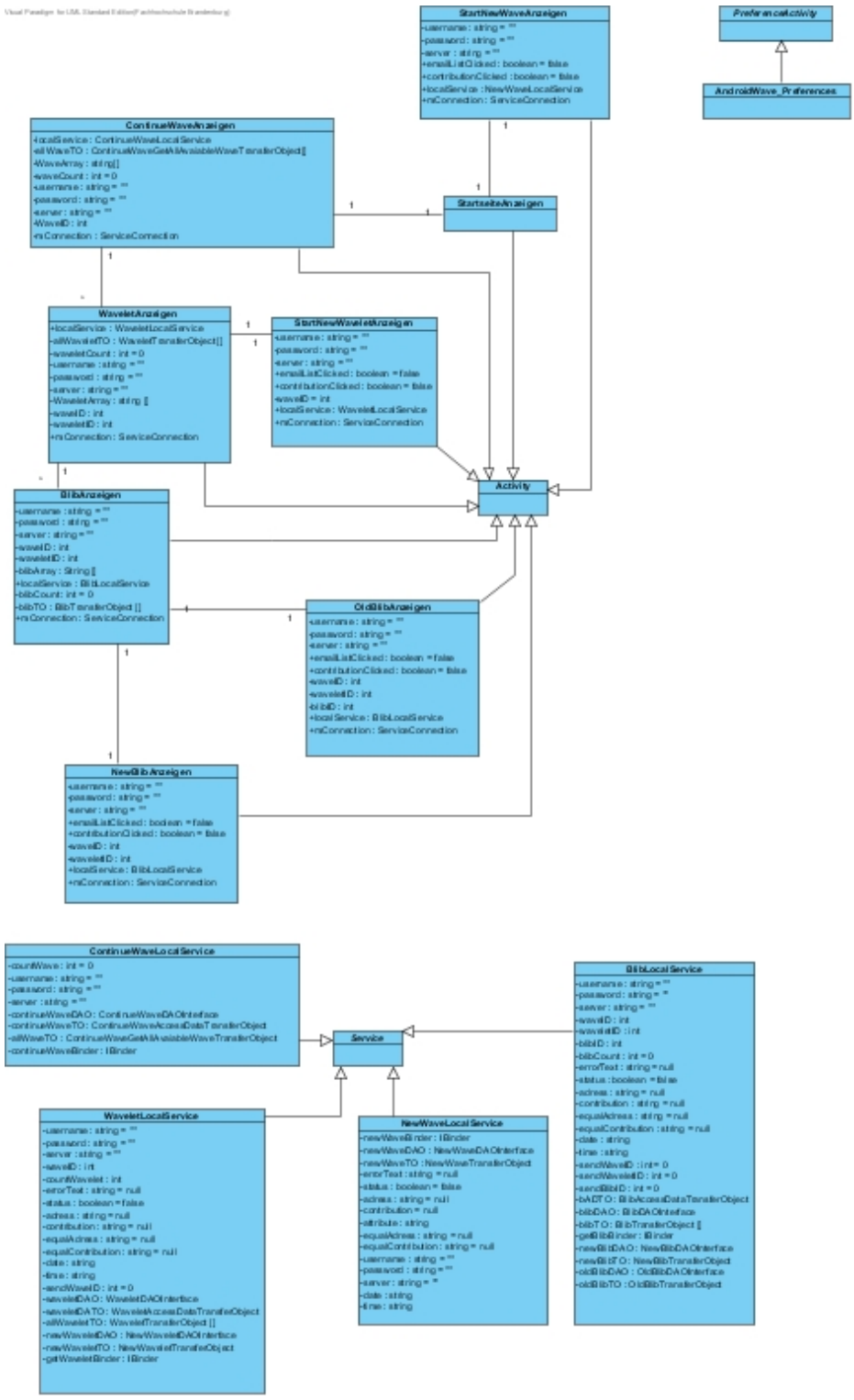

Abbildung 3.5: Das Domänenmodell zeigt alle konzeptionelle Klassen der Applikation AndroidWave

In einem Domänenmodell werden Klassen dargestellt, welche Konzepte aus dem Problemraum wiedergeben. Daher werden sie auch konzeptionelle Klassen genannt. Für die Ermittlung der konzeptionellen Klassen sind Use-Case-Diagramme wie jene im Kapitel 3.2.2.1 äußerst hilfreich.

Für *AndroidWave* ergibt sich ein Domänmodell, wie in Abbildung 3.5 dargestellt. Es ist auffällig, dass einige der Klassen in Gruppen zusammen gefasst und nicht mit den anderen Gruppen verbunden sind. Dies dient einerseits einer besseren Übersichtlichkeit, andererseits sind diese Gruppierungen aus konzeptionellen Gründen gewollt. Die Aufteilung zeigt unterschiedliche Aufgabenbereiche der Klassen auf. Für die im späteren Verlauf behandelten Komponentendiagramme ist diese Aufteilung eine hilfreiche Vorarbeit.
Demnach ergeben sich für die Applikation *AndroidWave* folgende konzeptionelle Klassen.

1. Activity Klassen
2. AndroidWave_Preferences Klasse
3. LocalService Klassen

Transferobjekt-Klassen oder DAO-Klassen beziehungsweise DAO-Interfaces werden im Domänenmodell nicht dargestellt. Auch die Darstellung des GoogleWave-Dummys ist für ein Domänenmoddell nicht relevant. All diese nicht dargestellten Komponenten werden aus dem Grund nicht modelliert, da sie Aufgaben für die konzeptionellen Klassen übernehmen. Ihre Existenz ist also eine direkte Auswirkung der Modellierung einer konzeptionellen Klasse und muss daher nicht extra angegeben werden. In späteren Kapiteln der Arbeit wird auf Komponenten eingegangen, welche nicht im Domänenmodell aufgezeigt werden. Die konzeptionellen Klassen werden nun genauer beschrieben.

Activity Klassen Diese Klassen sind in der Abbildung 3.5 zu einer Gruppe zusammen gefasst, da sie alle von der selben Klasse aus dem Android-SDK abgeleitet werden und schlussendlich die selbe Aufgabe beziehungsweise das selbe Konzept besitzen. Bei der besagten Klasse aus dem Android-SDK handelt es sich um die Activity-Klasse. Sie enthält einen sehr komplexen Funktionsumfang, deren genaue Erläuterung den Rahmen dieser Arbeit sprengen würde. Im allgemeinen sorgt eine Activity-Klasse für die Bildschirmausgabe, Aufruf von anderen Activities, Aufruf von Services, Zugriff auf Ressourcen und so weiter. Auf das MVC-Paradigma bezogen, entspricht die Activty-Klasse der Definition einer Controller-Klasse. Jede Klasse der Applikation *AndroidWave*, welche von der Activity-Klasse abgeleitet wird, erbt den besagten Funktionsumfang und die damit zusammenhängenden Aufgaben einer Controller-Klasse. Dementsprechend definiert sich das Konzept der damit zusammenhängenden Klassen.

Die Activity-Klassen von Android unterliegen innerhalb der Gruppe einer definierten Gliederung.
Als erste Activity wird die StartseiteAnzeigen-Klasse aufgerufen. Dies geschieht beim Starten der Applikation *AndroidWave*. Von dieser Activity aus können die Klassen beziehungsweise Activities

- AndroidWave_Preferences
- StartNewWaveAnzeigen
- ContinueWaveAnzeigen

aufgerufen werden. Die Klasse AndroidWave_Preferences gehört nicht zu der Gruppierung und wird im nachfolgenden Abschnitt beschrieben.
Mit der Klasse StartNewWaveAnzeigen wird eine Activity bereitgestellt, welche das Anlegen einer neuen Wave ermöglicht. Für die Ermittlung dieser Klasse liegt der Anwendungsfall aus Kapitel 3.2.2.1 zu Grunde.

Analog dazu wird die Klasse ContinueWaveAnzeigen als konzeptionelle Klasse durch den Anwendungsfall aus Kapitel 3.2.2.1 Abbildung 3.1 ausfindig gemacht. Dieser Anwendungsfall wird noch einmal in Kapitel 3.2.2.1 näher beschrieben. Der Nutzer hat dem Use-Case-Diagramm zufolge mehrere Möglichkeiten und „Stationen“, welche er innerhalb einer bestehenden Wave durchlaufen kann. Die Klasse ContinueWaveAnzeigen, realisiert dabei die erste „Station“, die Ausgabe aller verfügbaren Waves. Des weiteren ruft diese Klasse beziehungsweise Activity, nach einer entsprechenden Reaktion des Nutzers eine Klasse beziehungsweise Activity namens WaveletAnzeigen auf. Zunächst zeigt diese Klasse alle Wavelets der gewählten Wave an. Je nach Benutzereingabe werden aus dieser Activity herraus zwei weitere Activities, dem Anwendungsfall entsprechend, aufgerufen. Entweder die Activity NewWaveletAnzeigen, mit der neue Wavelets innerhalb einer Wave erzeugt werden könne, oder die Activity BlibAnzeigen. Die Klasse BlibAnzeigen entstand ebenfalls, wie alle bisher genannten Klassen, aus dem Anwendungsfall aus Kapitel 3.2.2.1. Sie zeigt zunächst alle Blibs einer Wavelet an. Entsprechend dem Anwendungsfall werden die Activities NewBlibAnzeigen, zum Erstellen einer neuen Blib, oder OldBlibAnzeigen, zum Manipulieren einer bestehenden Blib, dem Domänenmodell als konzeptionelle Klassen hinzugefügt. Diese werden ebenfalls durch eine entsprechenden Aktion des Nutzers aus der Klasse beziehungsweise BlibAnzeigen ausgeführt.

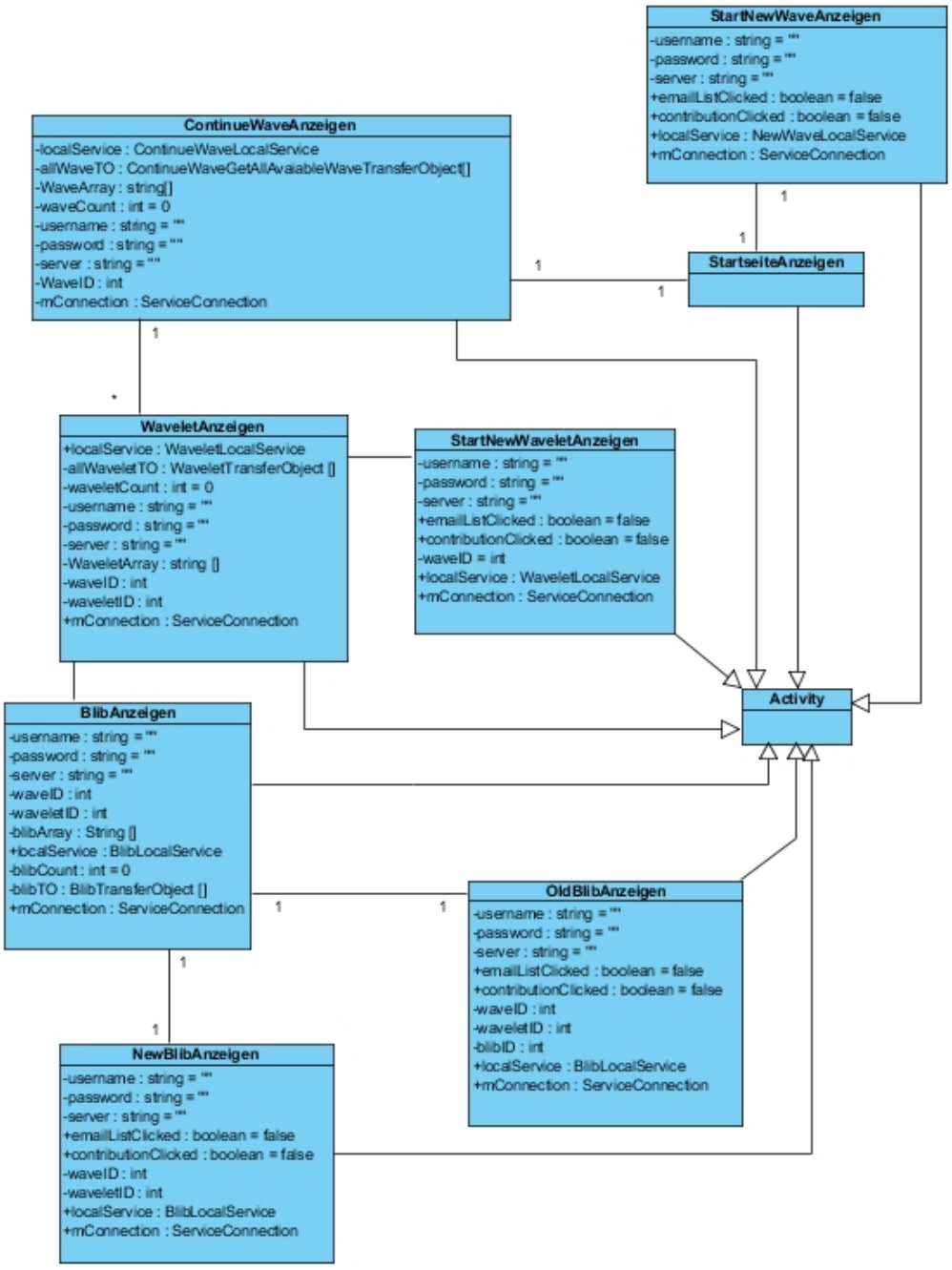

Abbildung 3.6: Ausschnitt des Domänenmodells: konzeptionelle Activity-Klassen der Applikation AndroidWave

Zusammenfassend wird also das gesamte Use-Case-Diagramm aus Kapitel 3.2.2.1 zum Erfassen von konzeptionellen Klassen für das Domänenmodell verwendet. Dementsprechend ergeben sich die folgenden konzeptionellen Klassen.

- ContinueWaveAnzeigen

- NewWaveletAnzeigen
- WaveletAnzeigen
  * BlibAnzeigen
    · NewBlibAnzeigen
    · OldBlibAnzeigen

AndroidWave_Preferences Klasse Diese Klasse ist ein direktes Resultat des Use-Case-Diagramms *Einstellungen vornehmen* (s.a. 3.2.2.1). Sie wird von der abstrakten Klasse des Android-SDKs PreferenceActivity abgleiten. Diese abstrakte Klasse ist in Android eine spezielle Form der Activity. Neben dem für eine gewöhnliche Activity üblichen Funktionsumfang, bietet eine PreferenceActivity zusätzlich Funktionen zur Speicherung von Einstellungsdaten. In der implementierten Applikation wird diese Klasse später über die Startseite im Hauptmenü erreichbar sein. Der Nutzer ist dann in der Lage, gemäß dem Anwendungsfall aus Kapitel 3.2.2.1, Einstellungen für den Nutzernamen, Passwort und Serveradresse vorzunehmen.

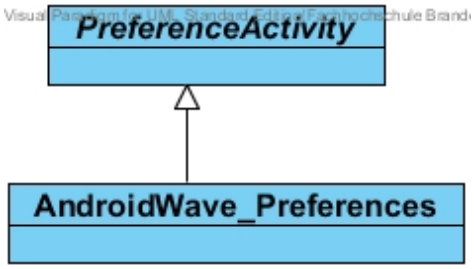

Abbildung 3.7: Ausschnitt des Domänenmodells: konzeptionelle AndroidWave_Preferences Klasse der Applikation AndroidWave

LocalService Klassen Die LocalService-Klassen können im Gegensatz zu den bisher genannten konzeptionellen Klassen, nicht aus einem Use-Case-Diagramm ermittelt werden. Sie stellen jedoch ein wichtiges Konzept für eine stabile Programmlaufzeit von *AndroidWave* dar.

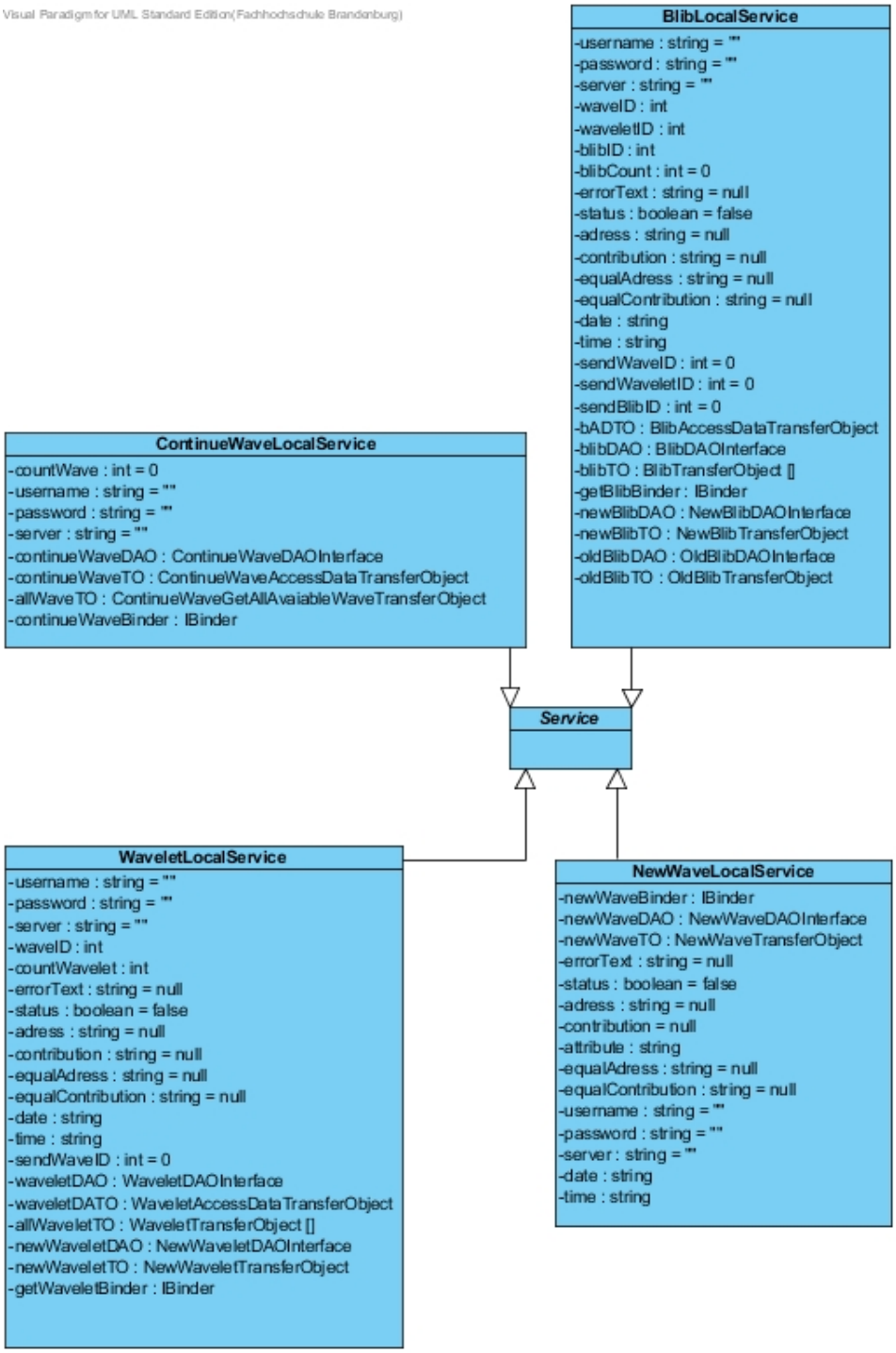

Abbildung 3.8: Ausschnitt des Domänenmodells: konzeptionelle LocalService Klassen der Applikation AndroidWave

Zu jeder Activity wird ein Thread in einem Prozess gestartet. In diesem Thread werden alle Anweisungen und Dienste der Activity durchgeführt. Der Thread ist bereits mit der Darstellung der Benutzeroberfläche und der Auswertung von Benutzereingaben relativ stark ausgelastet. Startet in einer Activity eine rechenintensive Anweisung zum Beispiel ein Internetdienst, wird der Thread zusätzlich belastet. Bei rechenintensiven Anweisungen handelt es sich meist um die Geschäftslogik der Applikation. Solange diese Anweisung ausgeführt wird, kann die Activity auf keine weiteren Aktionen reagieren, weder von Seiten des Systems noch von Seiten des Nutzers. Erst wenn die Anweisung beendet ist, kann die Activity wieder antworten. Im schlimmsten Fall führt eine rechenintensive Anweisung zu einem Absturz der ganzen Applikation.

In der Android-SDK existieren zwei Arten von Services, der *local Service* und der *remote Service*. Dabei handelt es sich um Software-Lösungen zum Betreiben von Hintergrundprozessen.
Beide Arten werden aus Activitiey heraus gestartet. Ein *remote Service* startet in einem eigenen Prozess mit einem eigenen Thread. Sie werden bevorzugt für die Kommunikation zwischen Applikationen verwendet, da sie über die Laufzeit ihrerer aufrufenden Activity hinaus existieren können. Ein *local Service* startet hingegen im selben Prozess wie die aufrufende Activity, jedoch in einem eigenen Thread. Der *local Service* kann daher nicht über die Laufzeit der Applikation hinweg weiter existieren, sondern nur so lange, wie die Applikation ausgeführt wird.
Für *AndroidWave* werden die *local Services* verwendet. Da mehrere Services gestartet werden sollen, müssen sie nach Beendigung der Applikation wieder beendet werden. Da ein *local Service* nach Beendigung der Applikation in jedem Fall gestoppt wird, wird die Freigabe von benötigten Systemressourcen garantiert. Sollte *AndroidWave* also unerwartet beendet werden, existiert die Gefahr nicht, unzählige verwaiste Prozesse im Speicher des Smartphones zurück zu lassen. Zudem ist die laufzeitübergreifende Existenz der Prozesse nicht notwendig. Der Vorteil aller Services besteht darin, dass sie mit einer höherer Priorität als die ausführende Activity gestartet werden. Das bedeutet, dass Services für Anweisung mit hohen Rechenaufwand besser geeignet sind als Actvities. Ihnen stehen mehr RAM-Speicher und mehr CPU-Ressourcen zur Verfügung. Sollte ein Service auf Grund mangelnder freier Ressourcen vorzeitig beendet werden, so wird er nicht vollständig gestoppt. Sobald die benötigten Systemressourcen wieder verfügbar sind, startet dieser automatisch neu, um die ihm übertragenen Aufgaben zu bearbeiten. Die Activity, die den gestoppten Service aufgerufen hat, ist die ganze Zeit verfügbar geblieben und konnte ohne Einschränkungen benutzt werden.

Für *AndroidWave* sind vier LocalService-Klassen vorgesehen. Jede Activity-Klasse wird

Zugriff auf eine der LacalService-Klassen erhalten. So kann sie Aufgaben mit Geschäftslogik-Character an die jeweilige LocalService-Klasse weiter geben. Die Aufteilung der Activity-Klassen auf die LocalService-Klassen, sieht wie folgt aus.

- NewWaveLocalService
  - StartNewWaveAnzeigen
- ContinueWaveLocalService
  - ContinueWaveAnzeigen
- WaveletLocalService
  - WaveletAnzeigen
  - NewWaveletAnzeigen
- BlibLocalService
  - BlibAnzeigen
  - NewBlibAnzeigen
  - OldBlibAnzeigen

3.2.2.3 Komponentendiagramm

Abbildung 3.9: Komponentendiagramm: Systemarchitektur von AndroidWave

Die Applikation *AndroidWave* soll, wie in dem Kapitel 3.1.6 gefordert, in einer Mehrschichtenarchitektur umgesetzte werden, unter zusätzlicher Berücksichtigung des MVC-Paradigmas.

AndroidWave wird in vier Schichten oder auch Layer entwickelt.

1. Presentation Layer
2. Business Layer
3. Data Access Layer

4. Data Store

Data Store Der *Data Store* ist keine Schicht der Applikation an sich. Er stellt vielmehr den GoogleWave-Dummy als Schicht im Hinblick auf das Gesamtprojekt dar. Diese Schicht ist also nicht auf dem Client implementiert sondern serverseitig zu finden. Hier werden die Schnittstelle, das heißt die PHP-Scripte für den Zugriff auf die Datenbank des GoogleWave-Servers und die Datenbank selbst angelegt. Die Erstellung der besagten Schnittstellen notwendig, weil es sich bei der Datenbank um eine MySQL-Datenbank handeln wird. Das Android-SDK besitzt keinerlei Möglichkeiten auf die MySQL-Datenbank eines Servers zuzugreifen.

Data Access Layer Der *Data Access Layer* wird Klassen beinhalten welche eine Verbindung zum GoogleWave-Dummy aufbauen. Sie nehmen Zugriff auf die Schnittstellen des Servers, also auf die PHP-Scripte. Jeder Schnittstelle wird mindestens die Login-Daten: Nutzername, Passwort und die Serveradresse übergeben.
In dieser Schicht werden des weiteren Klassen für den Austausch von Daten angelegt.

Business Layer Im *Business Layer* werden alle Klassen untergebracht, welche Anweisungen, das heißt Methoden der Geschäftslogik für *AndroidWave* enthalten. Dabei kann der *Business Layer* selbst noch einmal in einzelne Komponenten unterteilt werden.
Für *AndroidWave* ist eine spezielle Unterteilung des Layers nicht nötig. Die einzigen Komponenten im *Business Layer* von *AndroidWave* sind die Modell-Klassen. Sie werden für die Umsetzung des MVC-Paradigmas benötigt und gehören laut MVC in den *Business Layer*.

Presentation Layer Für *AndroidWave* wird der *Presentation Layer* zunächst in zwei Komponenten aufgeteilt. Die Aufteilung erfolgt nach dem MVC-Paradigma. Die Modell-Klassen wurden bereits dem *Business Layer* zugeteilt. Die beiden fehlenden, View-Klassen und Controller-Klassen, werden im *Presentation Layer* zugeordnet und bilden jeweils eine Komponente des Layers.

Zuordnung der konzeptionellen Klassen Das Domänenmodell weist drei Arten von konzeptionellen Klassen auf (s.a. 3.2.2.2). Diese Klassen werden nun jeweils einer Schicht, je nach Aufgabenbereich, zugeordnet.

Die Activity-Klassen werden dem *Presentation Layer* zuzuordnen. Von ihrer Konzeption beziehungsweise ihren Aufgaben her, entsprechen sie Controller-Klassen des MVC-Paradigmas.

Daher wird die Zuweisung dieser Klassen präzisiert und der *Presentations-Layer-Komponente* Controller-Klassen zugewiesen.
Für die AndroidWave_Preferences Klasse gilt dieselbe Einordnung, da sie trotz ihres speziellen Funktionsumfanges nach wie vor eine Activity ist.

Die LocalService-Klassen erfüllen nach dem MVC-Paradigma, eindeutig die Kriterien von Modell-Klassen und werden dem *Business Layer* zugeteilt. Sie beinhalten die Geschäftslogik von *AndroidWave* und stellen den Controller-Klassen Anwendungsdaten zur Verfügung. Die Anwendungsdaten wiederum beinhalten Informationen über Waves, Waveles oder Blibs, welche ihnen vom *Data Access Layer* breit gestellt werden.

3.2.2.4 Systemsequenzdiagramme

Das letzte Kapitel der OOA stellt Systemsequenzdiagramme dar. Ein Systemsequenzdiagramm bezieht sich immer ergänzend auf ein Use-Case-Diagramm. Es zeigt den Nachrichtenaustausch zwischen dem Akteur mit dem System. Der Informationsfluss ist rein konzeptioneller Natur. Die gezeigten Nachrichten sind nicht mit Funktionsaufrufen zu verwechseln. In späteren Phasen der OOD können diese Nachrichten helfen, für Klassendiagramme benötigte Funktionsaufrufe zu ermitteln. Des weiteren können Rückschlüsse über die zu modellierenden Benutzerschnittstellen getroffen werden.

Für *AndroidWave* ist das Systemsequenzdiagramme kurz SSD modifiziert worden. Üblicherweise wird in einem SSD nur der Akteur und das System, welches er benutzt, modelliert. Doch SSDs dürfen bei Bedarf durch weitere Komponenten ergänzt werden.
Das SSD für *AndroidWave* wird um die Komponente „GoogleWave-Dummy“ erweitert. Mit dem modifizierten SSD ist es nun möglich, die Integration von *AndroidWave* als Client zwischen dem Nutzer und GoogleWave beziehungsweise dem GoogleWave-Dummy darzustellen. Es werden Rückschlüsse, bezüglich der Arbeitsweise des Clients getroffen. Bisher existieren nur die konzeptionellen Klassen. Diese sind zudem in Schichten untergebracht, welche unabhängig von der Vorgehensweise des Clients, in jedem Fall implementiert werden müssen. Ihre Funktionsweisen werden nicht von der Analyse der SSDs beeinflusst. Im Bezug auf die noch zu entwickelnden Funktionsweisen der DAO-Klassen, Transferobjekten und Serverschnittstellen, ist eine Aussage über den Kommunikationsfluss vom Nutzer über *AndroidWave* zum GoogleWave-Dummy unerlässlich. Von dieser Erkenntnis hängt das weitere Software-Engineering für *AndroidWave* und dem GoogleWave-Dummy im Sinne der OOD im Kapitel 3.2.3 ab.

SSD Einstellung vornehmen Abbildung 3.10 zeigt das Systemsequenzdiagramm für den Anwendungsfall „Einstellung vornehmen“ aus Kapitel 3.2.2.1.

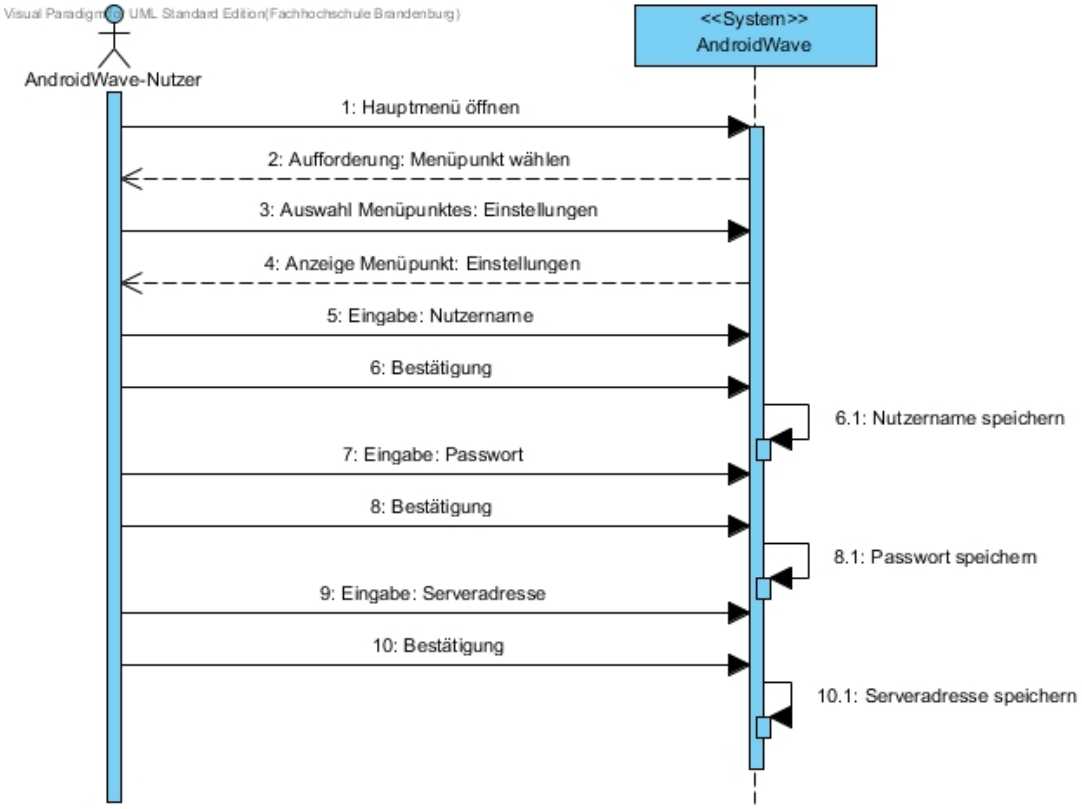

Abbildung 3.10: Systemsequenzdiagramm: Einstellungen vornehmen

Der Nutzer öffnet das Hauptmenü und wählt die Option „Einstellungen“ auf. Daraufhin werden ihm die Einstellungsoptionen angezeigt. Der Nachrichtenfluss in diesem SSD ist eher einseitig. Sie gehen vor allem vom Nutzer an das System. Es handelt sich immerhin um reine Eingaben des Nutzers.
Nach jeder Eingabe wird die jeweilige Option persistent in der Einstellungsdatenbank gespeichert. Wenn die Login-Daten benötigt werden, können sie jederzeit aus der Datenbank entnommen werden. Wie bereits erwähnt, bleiben die Daten nach Beendigung der Applikation weiterhin in der Datenbank gespeichert.

SSD Neue Wave starten Abbildung 3.11 zeigt das Systemsequenzdiagramm für den Anwendungsfall „Neue Wave starten“ aus Kapitel 3.2.2.1.

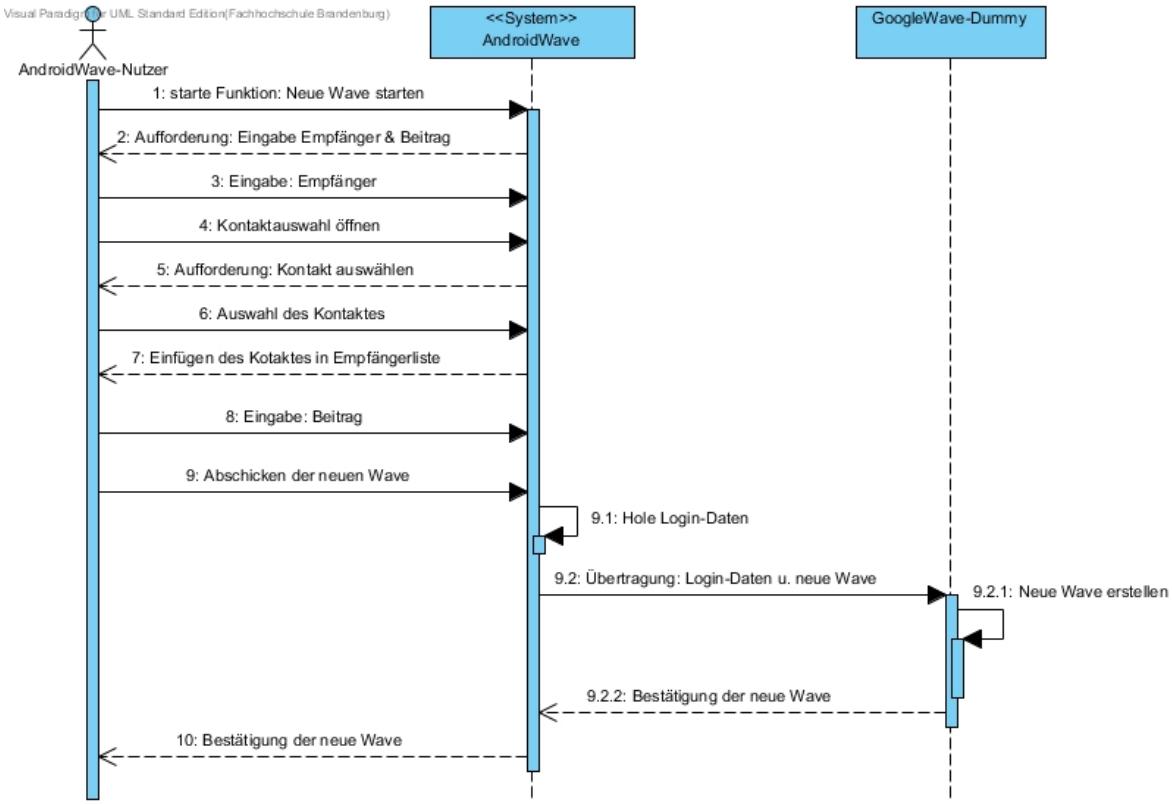

Abbildung 3.11: Systemsequenzdiagramm: Neue Wave starten

Wird der Anwendungsfall ausgeführt, geht die Kommunikation zwischen Akteur und System zu Beginn vor allem vom Akteur aus. Das System reagiert frühestens bei dem Aufruf der Kontaktdaten. Solange der Akteur mit der Eingabe der Empfänger und der Formulierung des Beitrages beschäftigt ist, bleibt die genannte Aktion die einzige des Systems.
Erst wenn der Akteur die neue Wave abschickt, liegt die Hauptlast beim System. Nach dem Laden der Login-Daten findet die Kommunikation nicht mehr allein zwischen Akteur und System statt. Von nun an kommuniziert das System mit dem GoogleWave-Dummy, um die neue Wave auf die Datenbank des Dummys zu übertragen. Ist die Authentifizierung bestätigt, wird die eigentliche Wave an die Datenbank übermittelt. Erst mit der Benachrichtigung über Erfolg beziehungsweise Misserfolg der Transaktion, wird wieder eine Nachricht an den Akteur gesendet.

SSD Wave fortsetzten Abbildung 3.12 zeigt das Systemsequenzdiagramm für den Anwendungsfall „Wave fortsetzten“ aus Kapitel 3.2.2.1. Der Anwendungsfall wird weiter spezifiziert. Geschildert wird das Anzeigen eines Blibs und die Erzeugung eines neue Blibs.

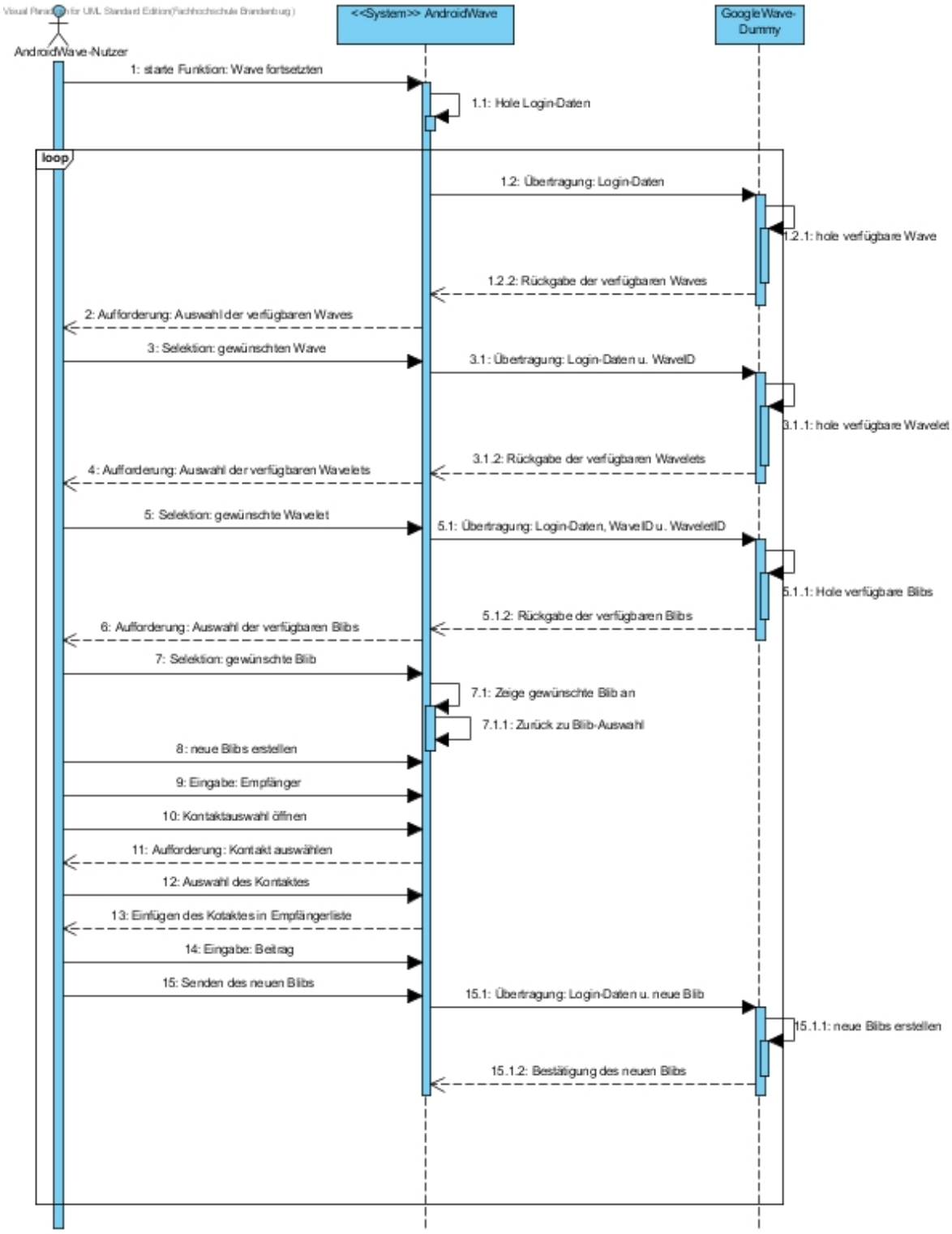

Abbildung 3.12: Systemsequenzdiagramm: Wave fortsetzten

Mit dem Beginn der Sequenz werden die Login-Daten geladen. Diese übermittelt *AndroidWave* an den GoogleWave-Dummy um sich zu authentifizieren. Die Kommunikation findet beim Anwendungsfall „Wave fortsetzten“ verstärkt zwischen dem System und dem GoogleWave-Dummy statt, da sich *AndroidWave* bei jedem Zugriff auf die Datenbank authentifizieren muss. Des weiteren werden benötigte Daten zwischen *AndroidWave* und dem Dummy ausgetauscht, von Variablen bis hin zum gewünschten Ergebnis. Der Nachrichtenaustausch zwischen Akteur und System findet immer dann statt, wenn der Akteur aus einer Wave, Wavelet oder Blib wählen muss. Die Selektion einer Auswahl verursacht erneut eine Kommunikation zwischen System und dem Dummy.
Erst wenn ein Blib, wie in der Abbildung 3.12 gezeigt, oder ein neues Wavelet angelegt wird, ist der Akteur wieder stärker in das Geschehen involviert. Dabei unterscheidet sich der Ablauf der Erstellung einer Blib oder eines Wavelets nicht sonderlich von dem, eine neue Wave anzulegen.
Der komplette Anwendungsfall kann beliebig oft wiederholt werden, bis das Programm beendet wird.

Auswertung der Systemsequenzdiagramme Bei der Betrachtung der drei SSDs ist auffällig, dass *AndroidWave* benötigte Information sammelt und entweder an den GoogleWave-Dummy oder an den Nutzer weiter gibt. Auf dem Client an sich werden keine weiteren Berechnungen angestellt. Die Informationen werden lediglich für den HTTP-Request aufbereitet, das heißt Sonderzeichen oder Umlaute werden codiert. Des weiteren wird die Systemzeit für eine eindeutige Zeitangabe erfasst, wenn eine Wave, Wavelet oder Blib erstellt wird.
Auf dem Client werden keinerlei Informationen, abgesehen von den Login-Einstellungen, gespeichert.
Die einzige Aufgabe des Client besteht also darin, entsprechende Informationen vom Nutzer für eine HTTP-Anfrage auf den GoogleWave-Dummy zu sammeln und abzuschicken. Zu den genannten Informationen gehören in jedem Fall die Daten der Login-Einstellungen und je nach Funktion Empfängerlisten, Nachrichten, Datum, Uhrzeit, ID der Wave, ID des Wavelets und/oder ID einer Blib. Die eigentlich Arbeit wird serverseitig ausgeführt. Das heißt, der Server stellt die Verbindung zur Datenbank her und führt entsprechende Datenbankanfragen aus. Die Ergebnisse der Datenbankanfrage werden zurück an den Client gesendet. Dieser bereitet die Ergebnisse für den Nutzer grafisch auf und gibt sie auf dem Bildschirm aus.
Während der Server, gesendete HTTP-Requests vom Client bearbeitet, kann der Nutzer den Client weiter uneingeschränkt nutzen und nach Bedarf weitere HTTP-Requests auslösen.

Auf Grund dieser Erkenntnisse, werden nun im Kapitel 3.2.3 die DAO-Klassen und Transfer-

objekte von *AndroidWave* und ferner die Schnittstellen des GoogleWave-Dummys wie folgt entworfen.

3.2.3 Design

3.2.3.1 Komponentendiagramm (erweitert)

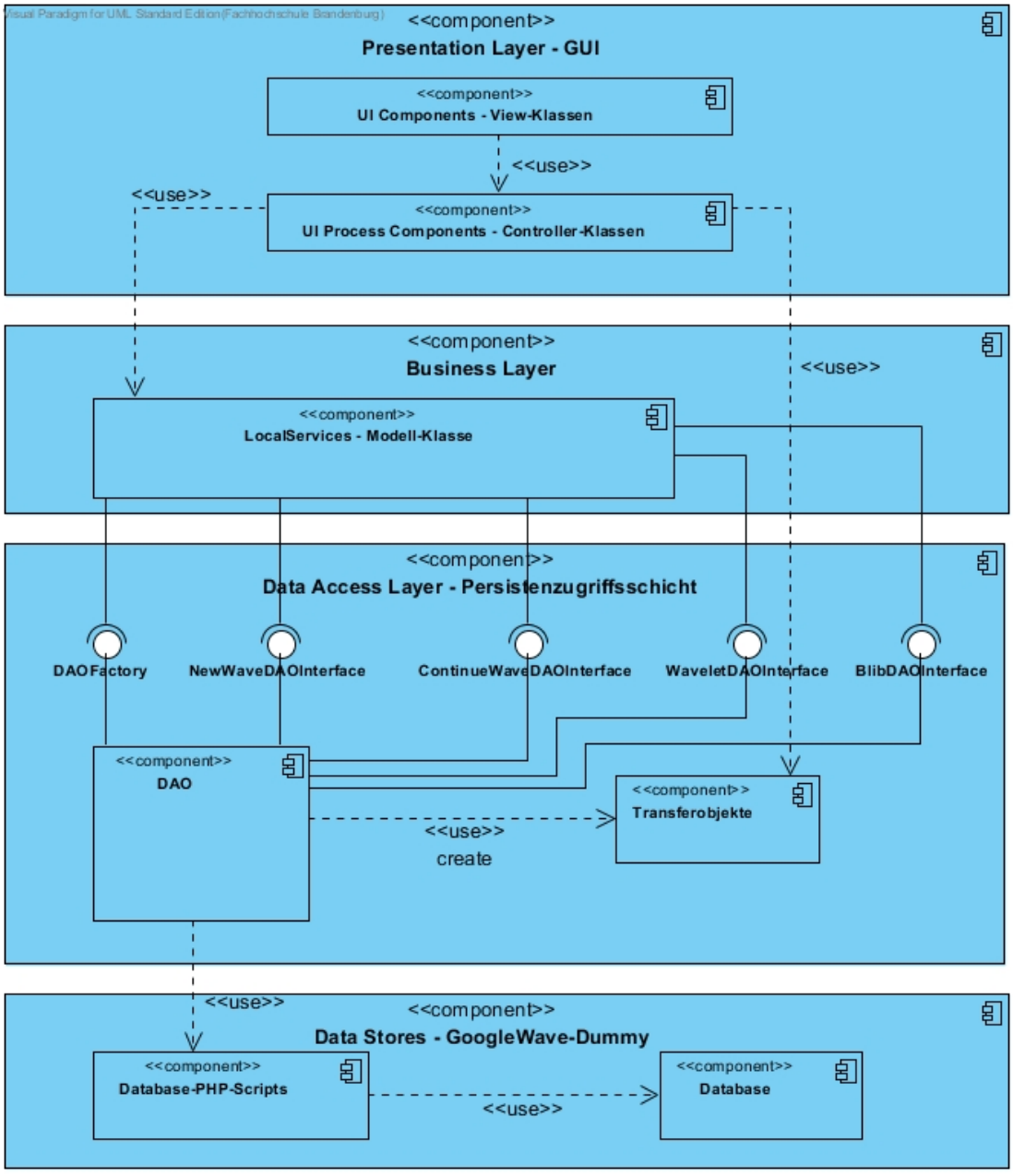

Abbildung 3.13: erweitertes Komponentendiagramm: Systemarchitektur von AndroidWave

Das Komponentendiagramm aus Abbildung 3.9 wird nun um die Komponenten

- View-Klassen
- DAO-Klassen und DAO-Interfaces mit dazugehöriger DAO-Factory
- Transferobjekte

erweitert. Das Resultat ist in der Abbildung 3.13 veranschaulicht. Des weiteren wird die Schnittstelle des GoogleWave-Dummys und die Datenbank eingetragen.
Alle neuen Komponenten sind ihren Aufgaben entsprechend den passenden Schichten zugeordnet.

View-Klasse In dem ersten Komponentendiagramm (s.a. Abbildung 3.9) wurden die konzeptionellen Klassen Acitivity, AndroidWave_Preferences und LocalServices den jeweiligen Schichten zugeteilt. Des weiteren ist ihre Bedeutung im Bezug auf das MVC-Paradigma geklärt. Um das MVC-Paradigma zu komplettieren, müssen nun die View-Klassen spezifiziert werden. Die Zuteilung in einer Schicht ist bereits vorgenommen. View-Klassen gehören dem Presentation-Layer an.

In Android werden Komponenten der View-Klasse nicht durch Javacode umgesetzt. Android verwendet in dieser Hinsicht XML-Dateien, um ein Bildschirm-Layout zu definieren.
Für *AndroidWave* werden während der Realisierung entsprechende XML-Dateien erstellt. Jede Activity wird ihr Bildschirm-Layout aus einer der XML-Dateien beziehen. Für die Ermittlung des passenden Bildschirm-Layouts spielen die Systemsequenzdiagramme eine wichtige Rolle. Der Nachrichtenaustausch zwischen Nutzer und System, also *AndroidWave*, bietet Aufschluss darüber, welche Eingabemöglichkeiten dem Nutzer zur Verfügung gestellt werden müssen, um *AndroidWave* bedienen zu können.

DAO-Klassen und DAO-Interfaces mit dazugehöriger DAO-Factory Diese Komponenten sind typischer Weise in dem *Data Access Layer* oder auch Persistenzschicht genannt, zuzuordnen.

Die Konstellation von DAO-Klassen,DAO-Interfaces mit dazugehöriger DAO-Factory stellt die Anwendung des Entwurfsmusters *Factory* dar.
Die Interfaces bestimmen dabei in einer abstrakten Form das Verhalten der später ausimplementierten DAO-Klassen. Das heißt sie geben die, für den Zugriff auf die Schnittstellen des GoogleWave-Dummys, zu implementierenden Methoden vor, ohne konkret auf die Logik der Methoden einzugehen. In Android, das heißt in Java, geschieht dies, indem nur der Rumpf einer Methode angegeben wird. Die eigentliche Implementierung findet in der konkreten Klasse, also den DAO-Klassen statt. Die konkrete DAO-Klasse erbt zunächst das Interface

und muss alle durch das Interface vordefinierten Methoden übernehmen. In den Methoden wird nun die eigentliche Logik implementiert. Die Factory verknüpft die Interfaces und die konkreten Klassen miteinander. Die Verknüpfung geschieht durch die Erstellung einer Methode. Die Methode ist **static** und vom Datentyp eines Interfaces. Als Rückgabewert wird eine Instanz der konkreten Klasse des Interfaces zurückgegeben.
Wird nun in einem LocalService eine DAO benötigt, so ist zunächst eine Variable vom Typ der gewünschten DAO in Form des DAO-Interfaces zu deklarieren. Dieser Variablen wird nun eine Instanz der konkrete DAO-Klasse durch den Aufruf der entsprechenden Methode aus der DAO-Factory übergeben.

Der Vorteil dieses Entwurfsmusters liegt in der Austauschbarkeit der konkreten DAO-Klassen. Das Interface definiert nur die zu verwendenden Methoden in Form des Methoden-Rumpfes. Die letztendlich implementierte Logik kann von einer konkreten DAO-Klasse zur anderen bei der Verwendung desselben Interfaces als Oberklasse unterschiedlich ausfallen.

Angenommen für den Datenaustausch zwischen *AndroidWave* und dem GoogleWave-Dummy existiert zu einem DAO-Interface eine konkrete DAO-Klasse. Die konkrete Klasse hat in ihren Methoden spezifische Zugriffslogiken für die Schnittstellen des Dummys implementiert. Auf dem *AndroidWave*-Dummy werden neue Schnittstellen installiert.
Die neuen Schnittstellen sind vollkommen anders konzipiert als die alten und *AndroidWave* kann mit den aktuellen DAO-Klassen nicht mehr auf den Dummy zugreifen.
Jetzt kommt der Vorteil des verwendeten Entwurfsmusters zum Tragen. Es wird eine neue DAO-Klasse implementiert. Durch das ‚Erben der Interfaces besitzen sie dieselben Methodenaufrufe mit entsprechender Parameterübergabe wie die alten DAO-Klasse. Ausschließlich die Logik für den Zugriff auf die Schnittstellen des Dummys sind innerhalb der Methoden angepasst. Nun wird die konkrete DAO-Klasse ausgetauscht, indem in der DAO-Factory an Stelle der Instanziierung der alten DAO-Klasse die Neue eingetragen wird. Für das restliche Programm stehen keine Änderungen des Programmcodes an. Sie greifen nach wie vor auf dieselben Methoden mit dem selben Methodenaufruf zu mit dem Unterschied, dass innerhalb der DAO-Klasse eine neue Logik zum Einsatz kommt.

Für *AndroidWave* wird eine DAO-Factory angelegt. Zu jeder Activity gehört ein DAO-Interface und eine konkrete DAO-Klasse. Da jede Activity eine bestimmte Aufgabe besitzt, müssen die zugehörigen DAO-Komponenten mit der entsprechenden Logik für den Zugriff implementiert werden. Die DAOs werden in dem, für die Activity zuständigen, LocalService-Objekt, wie zuvor beschrieben, während der Laufzeit erzeugt und verwendet.

Transferobjekte Die Transferobjekte sind Klassen welche eine bestimmte Anzahl an unterschiedlichen Datentypen und dazugehörigen Variablen aufnehmen können. Sie dienen ausschließlich der Weitergabe von inhaltlich zusammenhängenden Variablen.
Diese Klassen werden, wie bereits die DAO-Komponenten, dem *Data Accesss Layer* zugeordnet.

Es können in *AndroidWave* grundlegend drei verschiedene Typen von Transferobjekten benannt werden. Zum einen Transferobjekte welche alle Daten enthalten, um eine neue Wave, Wavelet oder Blib anzulegen. Diese Transferobjekte beinhalten grundsätzlich folgende Variablen

- Nutzername
- Passwort
- Serveradresse
- Nachricht
- Uhrzeit
- Datum

Je nach Bedarf werden die IDs von Wave, Wavelet und/oder Blib mitgegeben.

Die zweite Art von Transferobjekten enthalten lediglich die Login-Daten für die Schnittstellen des Dummys. Dies wird erforderlich, wenn die Datenbank auf verfügbare Waves, Wavelets oder Blibs abgefragt werden soll. Je nach Bedarf werden auch hier die benötigten IDs mitgegeben.
Zusammen mit der zuvor genannten Variante der Transferobjekte werden diese in den LocalServices als Typ einer Variable deklariert. Die LocalServices erhalten die entsprechenden Daten von ihren jeweiligen Activities. Die Daten werden von Methoden des LocalServices für den HTTP-Request in den DAOs angepasst und den Variablen der jeweiligen Transferobjekt-Variable zugewiesen. Daraufhin wird die Transferobjekt-Variable mittels eines Methodenaufrufes an ein DAO-Objekt weitergegeben. Dieses DAO-Objekt sende nun den Inhalte der Transferobjekt-Variable an eine entsprechende Schnittstelle des GoogleWave-Dummys.

Die dritte Art der Transferobjekte wird verwendet, um die Ergebnisse einer Datenbankabfrage vom Server temporär zu speichern. Hierfür wird in einem zuständigen DAO-Objekt eine entsprechende Variable mit dem Transferobjekt als Datentyp erzeugt. Ein DAO-Objekt erhält per JSON-Array eine Datenbank-Tabelle als Ergebnis einer Datenbankabfrage. Innerhalb des DAO-Objektes werden die Daten aus dem JSON-Array heraus in die Transferobjekt-Variable übertragen. Diese Variable wird nun dem LocalService zur Verfügung gestellt. Dieser reicht

die Variable weiter an seine Activity, welche die Daten für die Ausgabe auf dem Bildschrim benötigt.

Schnittstellen und Datenbank des GoogleWave-Dummys Die Schicht *Data Store* wird nach wie vor aus rein konzeptionellen Gründen im Komponentendiagramm aufgeführt und wird nicht innerhalb des Programmcodes von *AndroidWave* umgesetzt. Sie repräsentiert den GoogleWave-Dummy mit seinen Schnittstellen und der Datenbank für *AndroidWave*.

Die Schnittstellen bestehen aus PHP-Scripten. Zu ihrem Funktionsumfang gehört die Anmeldung auf einer Datenbank, das stellen von Datenbankanfragen und Senden einen HTTP-Response für die Übertragung von JSON-Arrays. Ausführlicher werden die Schnittstellen in Kapitel 3.2.3.4 erläutert.

Die Datenbank ist, üblich für XAMPP-Server, eine MySQL-Datenbank. Mit dieser Datenbank wird die Datenhaltung von GoogleWave auf dem Dummy simuliert. Dabei ist sie dem möglichen Datenbankschema des „echten“ GoogleWaves nach empfunden. Hierzu wird im Kapitel 3.2.3.3 näher auf das entwickelte Datenbankschema eingegangen.

Zusammenfassung Im Komponentendiagramm (s.a. Abbildung 3.13) werden alle benötigten Komponenten von *AndroidWave* dargestellt. Zusätzlich werden die Komponenten in Schichten und gegebenenfalls nach dem MVC unterteilt. Diese Unterteilung in Schichten beziehungsweise in das MVC ist Teil der Anforderungen (s.a. Kapitel 3.1).

Für *AndroidWave* entstehen so drei Schichten *Presentation Layer*, *Business Layer* und der *Data Access Layer*. Um das Gesamtkonzept von *AndroidWave* und dem GoogleWaveDummy zu verdeutlichen ist eine vierte Schicht dem Diagramm zugefügt worden, der *Data Store*.

Der *Presentation Layer* ist die erste Schicht von *AndroidWave*. Diese ist nochmals nach MVC in View-Klassen und Controller-Klassen unterteilt. Die View-Klassen werden in *AndroidWave* durch XML-Dateien realisiert, welche das Bildschirmlayout von *AndroidWave* definieren. Geladen und verwaltet werden die View-Klassen durch die Controller-Klassen. Controller-Klassen werden mit Activities umgesetzt. Die Controller-Klassen dienen zur Kommunikation zwischen View-Klassen und den Modell-Klassen aus dem *Business Layer*. Das heißt die Controller-Klassen erhalten von den Modell-Klassen die Anwendungsdaten, also Daten vom GoogleWave-Dummy. Die Controller-Klassen passen die Anwendungsdaten für die Bildschirmausgabe an und laden sie in die View-Klassen.

Im *Business Layer* sind die Modell-Klassen, umgesetzt durch *local Services*, untergebracht. Ihre Aufgabe ist es zum einen die Activities von rechenintensiven Anweisungen zu befreien

und zum anderen die Anwendungsdaten für die Controller-Klassen zu beschaffen.
Für die Beschaffung der Anwendungsdaten werden zwei Entwurfsmuster angewandt, die *Data Access Objects* oder auch DAO und die *Data Transfer Objects* auch kurz Transferobjekte genannt.

Die DAOs und Transferobjekte sind in dem *Data Access Layer* untergebracht. Die DAOs bestehen aus den DAO-Interfaces, DAO-Klassen und der DAO-Factory. Auch hier kommt wieder ein Entwurfsmuster zum Einsatz, die *Factory*. Die Funktionsweise der *Factory* ist ausführlich in dem Paragraph *DAO-Klassen und DAO-Interfaces mit dazugehöriger DAO-Factory* dieses Kapitels beschrieben.
Die DAOs erhalten die benötigten Informationen für einen Zugriff auf die Schnittstellen des GoogleWave-Dummys durch die Controller-Klassen. Diese erzeugen Transferobjekte mit den besagten Informationen. Diese Informationen erhalten sie wiederum von den Controller-Klassen, welche durch die View-Klassen die Benutzereingaben erfassen.
Es werden generell zwei Zugriffsarten unterschieden. Zum einen der Abruf von verfügbaren Wave, Wavelets oder Blibs, zum anderen das Erstellen von Waves, Wavelets und Blibs.
Bei der ersten Variante werden nur die Login-Daten und, je nach Abfrage, die IDs der Waves, Wavelets und/oder Blibs benötigt. Mit diesen Informationen startet die DAO den Zugriff auf eine entsprechende Schnittstelle des GoogleWave-Dummys. Die DAO erhält ein JSON-Array mit dem Ergebnis der Datenbankanfrage, in Form einer Datenbanktabelle, zurück. Diese werden den eigens entwickelte Transferobjekten übergeben. Diese mit Daten gefüllten Transferobjekte werden vom Controller benötigt, um den Nutzer auf dem Bildschirm eine Antwort auf seine Anfrage zu liefern. Doch ist die Controller-Klassen im *Presentation Layer* und die erzeugten Transferobjekte mit den Anwendungsdaten im *Data Access Layer*. Laut der Regel für die Nutzung der Schichtenarchitektur ist es untersagt, den zwischen Controller und Transferobjekt liegenden *Business Layer* zu „überspringen“.
Aus diesem Grund werden die Transferobjekte mit den Anwendungsdaten zunächst den Modell-Klassen des *Business Layers* übergeben und anschließend den Controller-Klassen in dem *Presentation Layer*, welcher die Ausgabe der Anwendungsdaten über die View-Klassen steuert.
Die zweite Variante des Zugriffes auf den GoogleWave-Dummy ermöglicht das Anlegen einer neuen Wave, Wavelet oder Blib. Hierzu wird der Nutzer zu entsprechenden Eingaben aufgefordert. Diese werden anschließend durch die Controller-Klassen an die Modell-Klassen weiter gegeben. Diese passen die Daten an die Zeichencodierung des HTTP-Request an, erfassen die Systemzeit und schreiben die gesammelten Daten in ein Transferobjekt. Dieses Transferobjekt wird einer zuständigen DAO übergeben, welche den HTTP-Request ausführt.

Der Request erhält eine entsprechende Schnittstelle des Servers. Die Schnittstellen sorgen mit einer geeigneten Datenbankanfrage für die korrekte Speicherung der übermittelten Daten auf der MySQL-Datenbank.

In dem *Data Store* befindet sich die MySQL-Datenbank des GoogleWave-Dummys und die Schnittstellen für *AndroidWave*. Die Schnittstellen werden benötigt, da *AndroidWave* aus Gründen fehlenden Funktionsumfanges nicht auf MySQL-Datenbanken zugreifen kann. Die Schnittstellen bestehen aus PHP-Scripten welche auf die GoogleWave-Dummy-Datenbank lesend und schreibend zugreifen können.

3.2.3.2 Klassendiagramm - Neue Wave starten

Das Klassendiagramm ist eine wesentliche konzeptuelle Erweiterung für den objektorientierten Entwurf. In dem folgenden Klassendiagramm sind Designentscheidungen bezüglich der Attribute, Operatoren, Sichtbarkeiten und Assoziationen aller Komponenten modelliert, welche für die Umsetztung des betrachteten Anwendungsfalles *Neue Wave starten* benötigt werden.

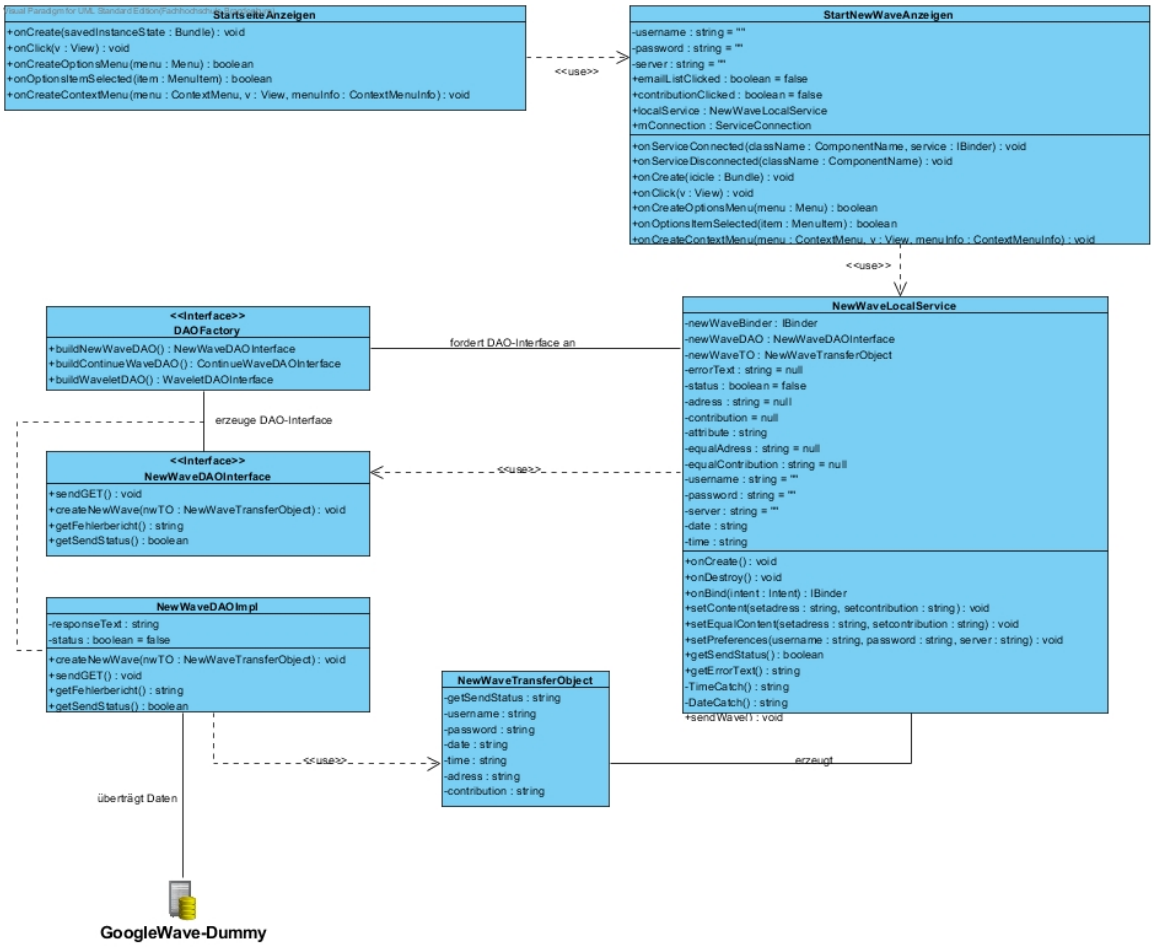

Abbildung 3.14: Klassediagramm: „Neue Wave starten“

StartseiteAnzeigen Die Klasse *StartseiteAnzeigen* besteht aus fünf Methoden. Diese Methoden sind ursprünglich Bestandteil der Klasse Activity und stehen *StartseiteAnzeigen* durch das Erben der Activity-Klasse zur Verfügung.

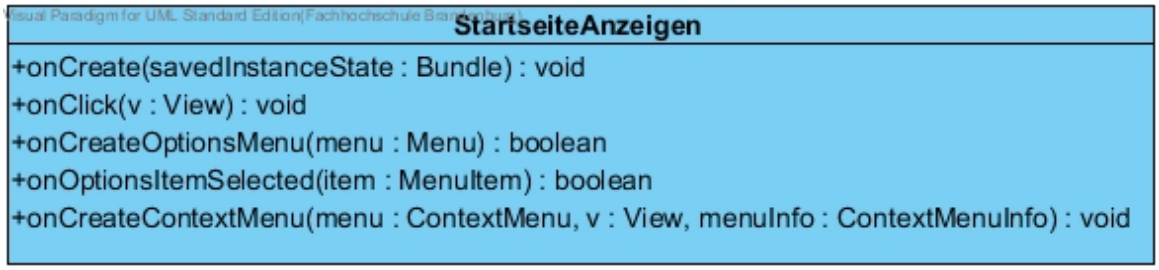

Abbildung 3.15: Ausschnitt des Klassendiagramms: *StartseiteAnzeigen*-Klassen der Applikation AndroidWave

Mit den Methoden *onCreateOptionsMenu* und *onOptionsItemSelected* wird das Hauptmenü definiert. In dem Hauptmenü wird in *AndroidWave* auf der Startseite dem Nutzer die Möglichkeit geboten, Angaben bezüglich des Nutzernamens, Passwortes und der Serveradresse vorzunehmen oder die Applikation zu beenden.
Die Methode *onCreateOptionsMenu* lädt dafür aus einer XML-Datei die Positionen, das heißt die Auswahloptionen des Menüs. Die XML-Datei wird während der Realisierung manuell erzeugt. Für das Hauptmenü sind die Optionen „Einstellungen“ und „Beenden“ vorgesehen. Durch die Methode *onOptionsItemSelected* wird den Auswahloptionen ihr Verhalten definiert. Das bedeutet, für die Option „Einstellungen“ wird die PrefenrencesActivity AndroidWave_Preferences ausgeführt und die Option „Beenden“ schließt die Anwendung *AndroidWave*.

Die Methode *onCreateContextMenu* sollte ursprünglich ein Hilfe-Menü für die einzelnen Bildschirmelemente von *AndroidWave* bereit stellen, wird jedoch aus Zeitgründen nicht in dieser Version von *AndroidWave*, ausimplementiert.

Die Methode *onCreate* ist die wohl wichtigste Methode einer Activity. Zu jeder Activity kann auf dem Bildschirm eine Bezeichnung, ähnlich wie bei einem Fenster auf dem Windows-Desktop, ausgegeben werden. Weiterhin wird hier das Bildschirm-Layout durch Laden der entsprechenden XML-Datei erzeugt.
Für die Startseite von *AndroidWave* werden in der *onCreate*-Methode zwei Schaltflächen definiert. Die erste soll das Starten einer neuen Wave und die zweite das Abrufen von bestehenden Waves ermöglichen. Um den Schaltflächen eine Funktion zuzuweisen ist ein „Click-Listener“ nötig.

Durch die Methode *onClick* wird der „Click-Listener“ realisiert. Wird ein „Click“ registriert, ist in der Methode *onClick* die auszuführende Aktion definiert. Bei einem „Click“ auf die

Schaltfläche „Neue Wave starten“ wird eine Instanz der Klasse *StartNewWaveAnzeigen* erzeugt und als Activity aufgerufen, beziehungsweise auf dem Bildschirm angezeigt.

StartNewWaveAnzeigen Auch diese Klasse besteht aus Methoden, welche durch das Erben von der Klasse Activity verfügbar sind. Auf dem Bildschirm werden zwei Textfelder(Empfängerliste und Nachricht) zu sehen sein und eine Schaltfläche zum Senden der neuen Wave.

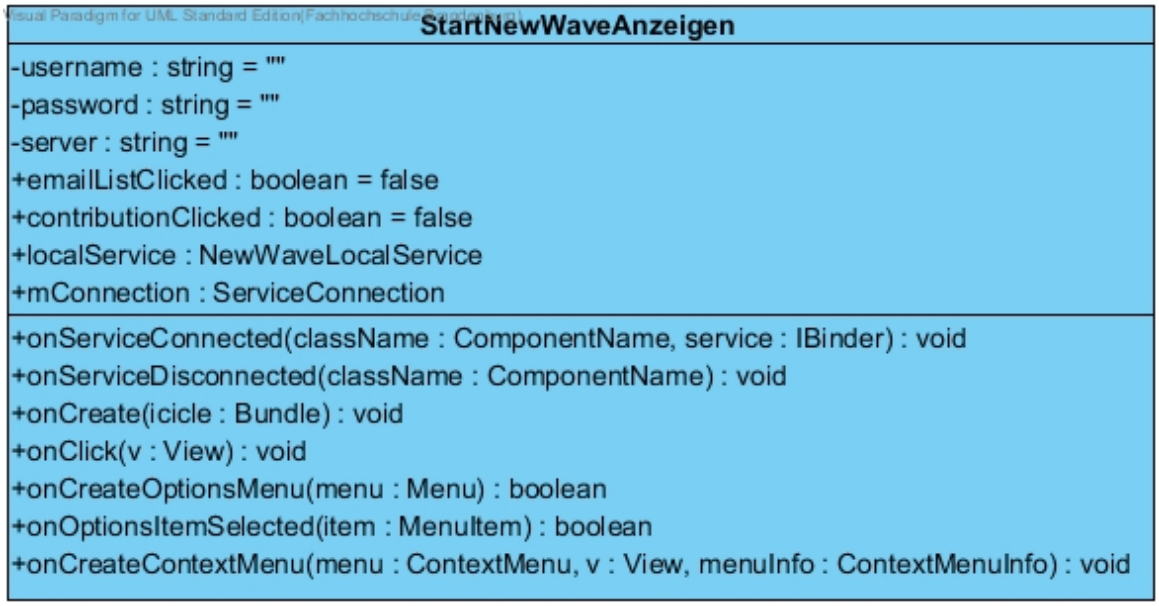

Abbildung 3.16: Ausschnitt des Klassendiagramms: *StartNewWaveAnzeigen*-Klassen der Applikation AndroidWave

Analog zu der Klasse *StartseiteAnzeigen* sind in dieser Klasse fünf Methoden zu finden. *onCreateContextMenu* bliebt nach wie vor unverändert. *onCreateOptionsMenu* und *onOptionsItemSelected* unterscheiden sich von denen aus *StartseiteAnzeigen* insofern, dass *onCreateOptionsMenu* eine andere XML-Datei übergeben wird. In diesem Menü ist die Einstellung der Login-Daten und das Beenden der Applikation nicht vorgesehen. Für die Activity *StartNewWaveAnzeigen* sind die Optionen „Kontakte“ und „Zurück“ verfügbar. *onOptionsItemSelected* definiert wieder das Verhalten der entsprechenden Optionen. Mit der Option „Zurück“ gelangt der Nutzer wieder zu Startseite von *AndroidWave*. Die Option „Kontakte“ sieht vor, die Kontaktdatenbank des Betriebssystems nach Email-Adressen zu durchsuchen. Diese sollen dann auf dem Bildschirm ausgegeben werden. Der Nutzer könnte nun aus den aufgelisteten E-Mail-Adressen wählen, um sie in die Empfängerliste aufzunehmen. Diese Funktion wird in der Version von *AndroidWave* aus zeitlichen Gründen nicht ausimplementiert.

Die Methoden *onCreate* und *onClick* unterscheiden sich im wesentlichen von denen aus der Klassen *StartseiteAnzeigen*. Zunächst wird wie in *StartseiteAnzeigen* auch der Titel der Activity für den Bildschirm festgelegt und das Bildschirm-Layout aus einer entsprechenden XML-Datei geladen. Dann werden in *onCreate* die Login-Daten aus der Datenbank für Anwendungseinstellungen gelesen und die entsprechenden Variablen übergeben. Anschließend wird eine Schaltfläche für das Absenden der neuen Wave deklariert. Mit dem Betätigen dieser Schaltfläche werden mehrere Anweisungen innerhalb der Methode *onClick* des „Click-Listener“ ausgelöst.

Zuerst wird in der Methode *onClick* die Verbindung zum *Local Service NewWaveLocalService* hergestellt. Hierfür werden die Variablen *mConnection* und *localService* benötigt. Die Variable *localService* ist ein Objekt der Klasse beziehungsweise des *Local Service NewWaveLocalService*. Die Variable *mConnection* ist ein Objekt der Klasse *ServiceConnection*. Mit der Instanzierung werden die Methoden *onServiceConnected* und *onServiceDisconnected* der Klasse *ServiceConnection* durch ihren Konstruktor erzeugt. Innerhalb der Methoden *onServiceConnected* und *onServiceDisconnected* wird die Programmlogik implementiert, welche ausgeführt werden soll, sobald mit *mConnection* die Verbindung zu einem Service aufgebaut beziehungsweise beenden wird. In der *onServiceConnected* wird die Variable *localService* letztendlich mit dem lokalen Services *NewWaveLocalService* verbunden.
Wird nun in der Methode *onClick* die Verbindung zu einem *Local Service* mit Hilfe der Variablen *mConnection* aufgebaut, repräsentiert die Variable *localService* den kontakteten *Local Service*. Über diese Variabel sind nun alle Methoden der Klasse *NewWaveLocalService* aufrufbar.
Im nächste Schritt erfasst die Methode *onClick* die Eingaben des Nutzers in den Textfeldern. Diese Eingaben und die Login-Daten werden nun Setter-Methoden der *NewWaveLocalService*-Variablen *localService* übergeben. Schließlich startet die Methode *onClick* mit dem Aufruf der Methode *sendWave* der Variablen *localService* den Mechanismus, welcher zur Speicherung der eben übergebenen Daten auf dem GoogleWave-Dummy führt. Bei Erfolg beziehungsweise Misserfolg erhält *onClick* durch Getter-Methoden von *localService* entsprechende Informationen. Sie erhält hierzu eine Boolean-Variable mit den Wert *true*, wenn das Senden erfolgreich war oder *false* wenn es fehl schlug. Im Fall, dass die Übermittlung fehlgeschlug, erhält eine Variabel über einen Getter den Fehler-Code.
In beiden Fällen, Erfolg beziehungsweise Misserfolg, wird der Nutzer mit einer entsprechenden Benachrichtigung informiert.

NewWaveLocalService Die Klasse *NewWaveLocalService* wird von der abstrakten Android-Klasse *Service* abgeleitet. Die Methoden *onCreate* und *onDestroy* sind nicht weiter zu beachten. Diese gehören zwar zum Funktionsumfang der *Service*-Klasse, sind aber für die weitere Implementierung von *AndroidWave* uninteressant.

Die Methode *onBind* ist jedoch für die Verbindung zur aufrufenden Activity von entscheidender Bedeutung. Sie übergibt der Activity, in der Methode *ServiceConnection* während der Definition der Verbindung, die Variable *NewWaveBinder*. *NewWaveBinder* ist ein Objekt eines Binders, welcher genau diese Instanz des *Local Service NewWaveLocalService* bei einem Verbindungsaufbau übergibt.

Zusammenfassend wird *NewWaveBinder* zu einem Attribut der *Local Service*, also *NewWaveLocalService* vereinbart, um sie in der Methode *onBind* an die Komponenten zurückzugeben, welche den Service startet.

Abbildung 3.17: Ausschnitt des Klassendiagramms: *NewWaveLocalService*-Klassen der Applikation AndroidWave

Die Klasse enthält entsprechende Getter- und Setter-Methoden als Schnittstelle für die Activity. In den Setter-Methoden werden die übertragenen Variablen für Nutzername, Passwort, Serveradresse, Nachricht und Empfängerliste, den jeweiligen Instanzvariablen zugewiesen. Durch die Getter-Methoden wird die Boolean-Variable für Erfolge beziehungsweise Misserfolg an die Activity weitergegeben, ebenso wie die Fehlermeldung. Der *Local Service* erhält seinerseits diese Informationen von der verwendeten DAO.

Der benötigte „Zeitstempel“ für die spätere Speicherung der neuen Wave wird mit den Methoden *TimeCatch* und *DateCatch* erfasst und ebenfalls entsprechenden Instanzvariablen übergeben.

Die Methode *sendWave* beinhaltet den Mechanismus, dessen Resultat die Speicherung der neue Wave auf dem GoogleWave-Dummy ist. Ausgelöst wird die Methode durch ihren Methodenruf in der Activity.
Zu Beginn deklariert die Methode die Variable *newWaveTO* vom Typ *NewWaveTransferObject*. Der Variable werden alle nötigen Instanzvariablen, die die Klasse aus den Setter-Methoden erhalten hat, übergeben. Zuvor erfolgen die Methodenaufrufe für *TimeCatch* und *DateCatch*, um auch den Zeitstempel übergeben zu können.
Nun wird die Variable *newWaveDAO* erzeugt, welch vom Typ *NewWaveDAOInterface* ist. Die Variable erhält den ausimplementierten Funktionsumfang der DAO-Klasse *NewWaveDAOImpl* durch den Aufruf der Methode *buildNewWaveDAO* der *DAOFactory*.
Mit der Variablen *newWaveDAO* kann nun auf die Methode von *NewWaveDAOImpl* zugegriffen werden. Die Methode *createNewWave* der Klasse *NewWaveDAOImpl* führt den Mechanismus zum Erstellen einer neuen Wave auf der *Data Access Ebene* fort. Der Methode *newWaveDAO* wird beim Methodenaufruf in der Parameterliste das Transferobjekt in Form der Variablen *newWaveTO* für die weitere Bearbeitung übergeben.
Zum Ende der Methode *sendWave* erhält sie durch eine Getter-Methode des *newWaveDAO*-Objectes den Status über Erfolg oder Misserfolg der Transaktion. Ist letzteres der Fall wird eine weitere Getter-Methode des *newWaveDAO*-Objectes aufgerufen, welche den Fehlercode übergibt. Status und, je nach Bedarf, der Fehlercode werden in Instanzvariablen für die Getter-Methoden *getSendStatus* und *getErrorText* gespeichert. Diese werden von der Activity für die weitere Verarbeitung aufgerufen.

NewWaveTransferObject Das Transferobjekt *NewWaveTransferObject* enthält alle, für das Anlegen einer neuen Wave, benötigten Daten. Ein Objekt der Klassen wird in dem *Local Service NewWaveLocalService* für die Weitergabe an die DAO *newWaveDAO* erzeugt.

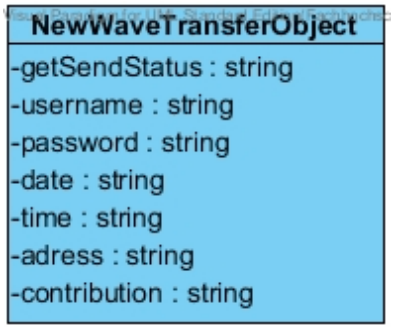

Abbildung 3.18: Ausschnitt des Klassendiagramms: *NewWaveTransferObject*-Klassen der Applikation AndroidWave

DAOFactory In der Klasse *DAOFactory* stehen Methoden für *AndroidWave* bereit, um alle DAO-Interfaces mit ihren konkreten Klassen zu verbinden.

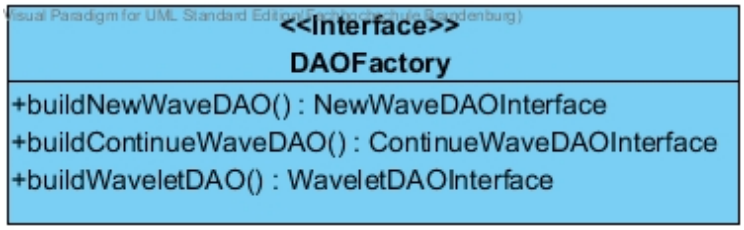

Abbildung 3.19: Ausschnitt des Klassendiagramms: *DAOFactory*-Klassen der Applikation AndroidWave

NewWaveDAOInterface Das Interface gibt der konkreten Klasse *NewWaveDAOImpl* alle zu implementierenden Methoden als Methodenrumpf vor. Für die konkrete Klasse *NewWaveDAOImpl* sind folgende Methoden vorgesehen.

- createNewWave
- getFehlerbericht
- getSendStatus

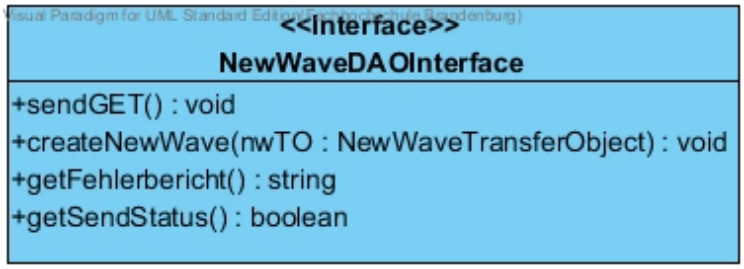

Abbildung 3.20: Ausschnitt des Klassendiagramms: *NewWaveDAOInterface*-Klassen der Applikation AndroidWave

NewWaveDAOImpl Die Klasse *NewWaveDAOImpl* realisiert alle drei ihr vorgegebenen Methoden des Interfaces *NewWaveDAOInterface*. Die Klasse besitzt des weiteren zwei Instanzvariablen für den Fehlerbericht und den Status der Übermittlung.

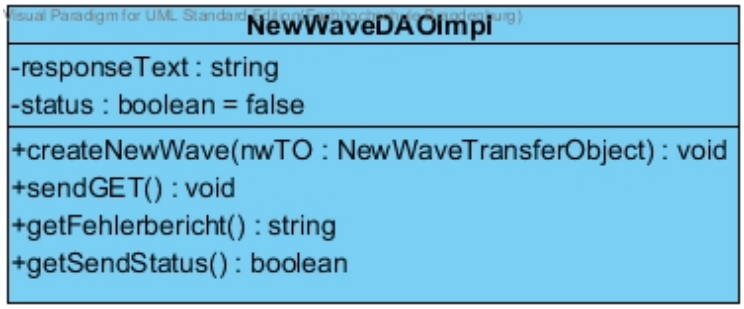

Abbildung 3.21: Ausschnitt des Klassendiagramms: *NewWaveDAOImpl*-Klassen der Applikation AndroidWave

Durch den Methodenaufruf von *createNewWave* wird ihr Programmcode ausgeführt. Die Laufzeit der Methode sieht das Erstellen einer HTTP-URL-Verbindung mit dem GoogleWave-Dummy vor.
Der Server-Schnittstelle, in Form des PHP-Scriptes *setNewWave.php*, werden in einem HTTP-Requeste alle Variablen des in der Parameterliste mit übergebenen Transferobjektes übermittelt.

Die Durchführung des Requests wird durch einen Exception-Handler überwacht. Sollte bei der Übertragung ein Fehler auftreten, wird die Status-Instanzvariable auf den Wert *false* gesetzt und der durch den Exception-Handler abgefangene Fehlerbericht der entsprechenden Instanzvariable übergeben. Die zwei Instanzvariablen werden über Getter-Methoden dem aufrufenden *Local Service* zugänglich gemacht.

GoogleWave-Dummy Das PHP-Script *setNewWave.php* nimmt den HTTP-Request der DAO entgegen und verarbeitet diesen. Das Resultet ist eine Folge von Datenbankanfragen, welche eine neue Wave auf der Datenbank des GoogleWave-Dummys anlegen.

3.2.3.3 Datenhaltung auf dem GoogleWave-Dummy

Die Datenbank auf dem GoogleWave-Dummy ist der des „echten“ GoogleWave nachempfunden. Das eigentliche Datenbankschema von GoogleWave stand während der Bearbeitungszeit dieses Dokumentes nicht zur Verfügung. Ermittelt werden die Tabellen der Datenbank des GoogleWave-Dummys mit der Abbildung 2.4 und den Beschreibungen zu den Bestandteilen einer Wave (s.a. Kapitel 2.2.2).

Für das Speichern einer Wave müssen ihre drei Komponenten, Daten-Dokumente ausgenommen, in einer Datenbank realisiert werden. Das heißt, es müssen die Tabellen Wave, Wavelet und Blib angelegt werden.
Zu jeder Wavelet existiert genau eine Empfängerliste. Daher liegt es nahe, auch hierfür eine entsprechende Tabelle anzulegen.
Weitere Überlegungen lassen den Schluss zu, dass zu jeder Blib ein Zeitstempel existieren muss. Die Angabe von Zeit und Datum einer Blib kann in einer Diskussion ausschlaggebend für den weiteren Verlauf sein. Außerdem muss eine Aussage über die letzte Änderung einer Wavelet getroffen werden können. Das Speichern eines Zeitstempels lässt sich am wirkungsvollsten durch eine eigene Tabelle realisieren.
Zu jeder Blib ist es notwendig, Angaben über den Autor machen zu können. Das heißt, Nutzername, Vorname und Nachname müssen persistent speicherbar sein. Zu dem Zweck wird das Anlegen einer weiteren Tabelle notwendig.

Diesen Überlegungen zufolge ergibt sich das folgende Datenbankschema.

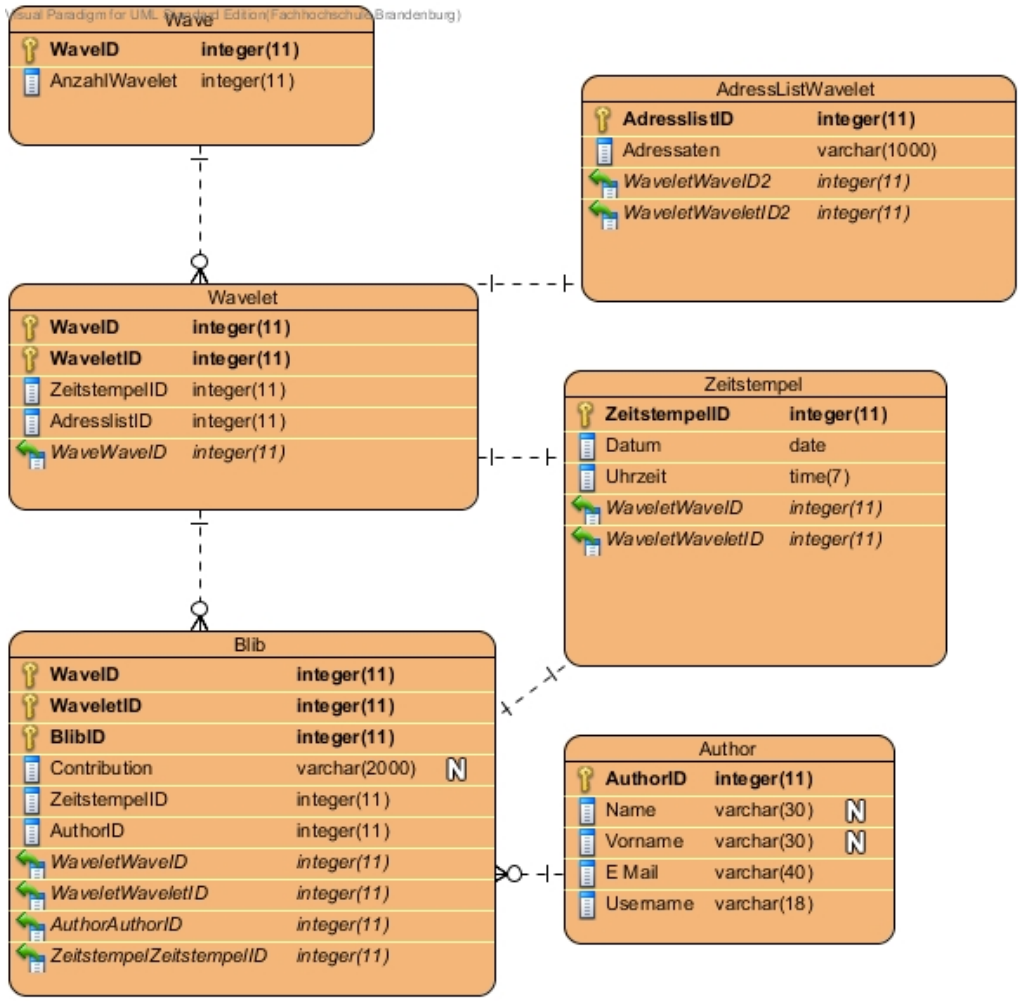

Abbildung 3.22: Das Entity-Relationship-Modell zeigt das Datenbankschema des GoogleWaveDummys

Auf dem GoogleWave-Dummy können Wave, gegliedert in Wavelets und Blibs persistent gespeichert werden. Hierfür sind die Tabellen Wave, Wavelet und Blib angelegt.

Wavelet Der Primärschlüssel der Tabelle Wavelet setzt sich aus einem Fremdschlüssel aus der Tabelle Wave und der WaveletID zusammen. Jedes Wavelet besitzt Werte für einen Zeitstempel und eine Adressliste. Der Zeitstempel ist über die ID der Tabelle Zeitstempel erreichbar. Das selbe gilt für die ID der Adressliste und der Tabelle AdressListWavelet. Zu jeder Wavelet gibt es demnach exakt eine ZeitstempelID und eine AdressID

Zeitstempel Die Tabelle Zeitstempel ist über den Primärschlüssel ZeitstempelID erreichbar. In dieser Tabelle wird zu jeweils einer ZeitstempelID ein Datum und eine Uhrzeit gespeichert.

AdressListWavelet AdressListWavelet ist eine Tabelle für das Speichern von Empfängerlisten. Der Primärschlüssel der Tabel ist die AdresslistID. Zu jeder ID gibt es genau eine Adressliste.

Blib Die Tabelle Blib wird zunächst aus ihrem zusammengesetzten Primärschlüssel gebildet. Der Primärschlüssel besteht aus dem Fremdschlüssel der Tabelle Wavelet und der BlibID. In der Variablen Contribution wird der Beitrag des Blibs gespeichert. Jeder Blib verfügt des weiteren über genau einen Zeitstempel, welcher durch die ZeitstempelID identifiziert wird. Zuletzt wird für jede Blib der Autor gespeichert.

Alle Tabellen sind mindestens in der dritten Normalform. So wird eine redundante Speicherung der Daten vermieden.

3.2.3.4 Zugriff auf die Datenbank

Die Datenbank auf dem GoogleWave-Dummy ist eine MySQL-Datenbank. Mit dem Android Software Development Kit kann kein direkter Zugriff auf eine MySQL-Datenbank ausgeführt werden. Dem SDK fehlen dafür die entsprechenden MySQL-Treiber.
Der Zugriff auf die Datenbank muss indirekt erfolgen. Serverseitig müssen PHP-Scripte angelegt werden. Diese beinhalten die jeweiligen Datenbankanfragen. Von *AndroidWave* wird ein HTTP-Request auf das entsprechende PHP-Script ausgeführt. Mit dem HTTP-Post-Request werden dem Script Variablen übergeben, beispielsweise Nutzername und Passwort, für den Login. Das PHP-Script führt die Datenbankanfrage aus und sendet das Ergebnis als JSON-Array per HTTP-Response zurück an *AndroidWave*. Auf dem Client können dann die empfangen Daten weiterverarbeitet werden.

Für jeden möglichen Zugriff werden auf dem Server entsprechende PHP-Scripte bereitgestellt. Mögliche Zugriffe sind:

1. Anlegen einer neuen Wave
2. Abrufen aller verfügbaren Waves
3. Abrufen aller verfügbaren Wavelets einer Wave
4. Erzeugen einer neuen Wavelet
5. Abrufen aller verfügbaren Blibs einer Wavelet
6. Erzeugen einer neuen Blib
7. Ändern einer bestehenden Blib

4 Realisierung

In diesem Kapitel wird die Implementierung von *AndroidWave* erläutert. Bei der Implementierung werden die Anforderungen aus Kapitel 3.1 umgesetzt und die erarbeiteten Aspekte der OOA und OOD aus Kapitel 3 angewendet. Dieses Kapitel wird in zwei Teil gegliedert. Im ersten Teil wird die Programmierung der Komponenten von *AndroidWave* erläutert. Der zweite Teil geht auf die Umsetzung der Benutzeroberfläche ein.

4.1 Programmierung

4.1.1 Android-Manifest

Die wichtigste Komponente einer Android-Applikation stellt die XML-Datei AndroidManifest.xml dar. In dieser Datei wird festgelegt, welche Activities und Services während der Programmlaufzeit ausgeführt werden dürfen. Das heißt sie werden dem Android-System bekannt gemacht. Sollte eine Applikation eine Activity starten, welche dem System unbekannt ist, wird die Anwendung mit einer Fehlermeldung geschlossen. Die folgenden Listings zeigen beispielhaft wie eine Activity beziehungsweise ein Service in der XML-Datei eingetragen werden. Der gezeigte Quellcode ist der AndroidManifest.xml von *AndroidWave* entnommen.

Listing 4.1: Ausschnitt AndroidManifest.xml: Anmeldung einer Activity

```
1<activity android:name=".GUI.Controller.StartseiteAnzeigen">
2</activity>
```

Listing 4.2: Ausschnitt AndroidManifest.xml: Anmeldung eines localServices

```
1<service android:name=".business.service.WaveletLocalService">
2</service>
```

Eine der eingetragenen Activities kann in der XML-Datei so modifiziert werden, dass sie nach dem Start des Programms angezeigt wird. Es handelt sich hierbei also um die Definition der Startseite einer Applikation.

Die entsprechende Acitvity wird durch einen sogenannten Intent-Filter erweitert. Ein Intent-Filter ist ein Mechanismus des Android-Systems, zur Vergabe von Berechtigungen für Activities oder Services.
Das folgende Listing zeigt wie der Eintrag der Activity *StartseiteAnzeigen* aus Listing 5.1 für die Applikation *AndroidWave* mit einem solchen Intent-Filter erweitert wurde.

Listing 4.3: Ausschnitt AndroidManifest.xml: StartseiteAnzeige

```
1 <activity android:name=".GUI.Controller.StartseiteAnzeigen"
2  android:label="@string/app_name">
3    <intent-filter>
4       <action android:name="android.intent.action.MAIN" />
5       <category android:name="android.intent.category.LAUNCHER" />
6    </intent-filter>
7</activity>
```

Somit wird die Activity *StartseiteAnzeigen* nach dem Starten der Applikation *AndroidWave* als Startseite angezeigt.

Für die Funktionalität von *AndroidWave* ist der Zugriff auf das Internet von elementarer Bedeutung. Jedoch kann eine Applikation nicht nach belieben auf Ressourcen wie das Internet zugreifen. Eine entsprechende Berechtigung muss erst in der AndroidManifest.xml festgelegt werden. Das Listing 5.4 zeigt den ergänzten Quellcode.

Listing 4.4: Ausschnitt AndroidManifest.xml: Zugriffsberechtigung Internet

```
1<uses-permission android:name="android.permission.INTERNET" />
```

Nun besitzt *AndroidWave* die benötigten Recht und kann auf den *AndroidWave-Dummy* über eine bestehende Internetverbindung zugreifen.

4.1.2 Einstellungen

AndroidWave soll Nutzernamen, Passwort und Serveradresse persistent auf dem System speichern (siehe Kapitel 3.1.4). Demnach soll der Nutzer in der Lage sein, diese Daten nach dem Start einzusehen und sie bei Bedarf abzuändern. Auch die Applikation soll auf die Daten zugreifen können, um sich bei dem entsprechenden Server mit Nutzername und Passwort anzumelden.

Das Software-Development-Kit von Android bietet hierfür ein äußerst komfortable Lösung. In den folgenden Zeilen wird diese im Zusammenhang mit der Implementierung von *AndroidWave* näher beschrieben.
Den Kern der Lösung bildet die XML-Datei *androidwave_preference.xml* und die spezielle Form der Activity namens *AndroidWave_Preference.java*.

Listing 4.5: Beispiel: Definition einer Einstellungsoption in XML

```
1<PreferenceScreen
2  xmlns:android="http://schemas.android.com/apk/res/android">
3
4 <PreferenceCategory
5        android:title="@string/cfg_verbindungsdatenTitel"
6       >
7       <EditTextPreference
8          android:key="username"
9          android:title="@string/cfg_verbidnungsdatenUsername"
10        />
11        <EditTextPreference
12      android:key="password"
13      android:title="@string/cfg_verbindungsdatenLoginPasswort"
14        />
15        <EditTextPreference
16      android:key="serveradresse"
17      android:title="@string/cfg_verbindungsdatenServerAdresse"
18        />
19 </PreferenceCategory>
20</PreferenceScreen>
```

Zunächst wird die XML-Datei wie in Listing 5.5 als PreferenceScreen vereinbart. So wird ein Standard-Layout geladen. Weitere Layout-Definitionen entfallen hierdurch. Es können

nun die zu erfassende Daten Nutzername, Passwort und Serveradresse als editierbare Texte definiert werden. Jedes Element erhält einen Schlüssel. Mit Hilfe des Schlüssels wird später der Zugriff auf die Elemente ermöglicht.

Für die Ausgabe des definierten PreferenceScreen, der *androidwave_preference.xml*, auf dem Bildschirm, wird eine Activity benötigt. Für die Darstellung und Erfassung von Einstellungen stellt Android die Activity-Klasse *PreferenceActivity* zur Verfügung. *AndroidWave_Preference.java* wird von diese Activity-Klasse abgeleitet. Der nächste Schritt ist die Verknüpfung der Activity mit der XML-Datei. Im Listing 5.6 ist die Klasse *AndroidWave_Preference.java* dargestellt.

Listing 4.6: Ausschnitt der PreferenceActivity

```
1 public class AndroidWave_Preference extends PreferenceActivity{
2         public void onCreate(Bundle icicle){
3                 super.onCreate(icicle);
4                 this.setTitle(R.string.cfg_verbindungsdatenTitel);
5                 this.addPreferencesFromResource(
6                 R.xml.androidwave_preference);
7         }
```

Die besagte Verknüpfung ist in Zeile fünf bis sechs zu finden. Hinter dieser Anweisung und der gesamten Klasse steckt weit mehr als nur das simple Anzeigen und Editieren der Einstellungen. Die Eingaben werden in einer eigens für Einstellungen der Applikation erzeugten Datenbank geschrieben. Jeder Applikation steht im Android-System ein eigener Ordner zur Verfügung. In eben diesem Ordner ist die Applikation selbst und diese Datenbank zu finden. Auf die Datenbank hat ausschließlich die Applikation Zugriff.

Nun kann die *PreferenceActivity* über einen Intent aufgerufen werden. Im Fall von *AndroidWave* ist *AndroidWave_Preference.java* in dem Hauptmenü der „Startseite“ integriert. Nur in der *StartseiteAnzeigen*-Activity kann der Nutzer die Einstellungen einsehen und ändern.

4.1.3 Neue Wave starten

Die Verwirklichung der Programmfunktion „neue Wave anlegen“ erfolgt strikt nach dem Schema wie es das Kapitel 3.2.3.2 und 3.2.3.1 vorgibt.

Da eine genaue Schilderung des Programmablaufes bereits in Kapitel 3.2.3.2 vorgenommen wurde, geht dieses Kapitel ausschließlich auf spezielle Programmcodeabschnitte ein.

- Verknüpfung von View- und Controller-Komponenten
- Implementierung der Verbindung mit *NewWaveLocalService*
- Implementierung des DAO-Interfaces *NewWaveDAOInterface* und der *DAO-Factory*

4.1.3.1 Verknüpfung von View- und Controller-Komponenten

In Android werden die View-Komponenten des MVC mit XML-Dateien umgesetzt. Das folgende Listing zeigt einen Ausschnitt aus der XML-Datei, welche die Startseite von Android definiert.

Listing 4.7: Beispiel: Definition eines Bildschirm-Layouts

```
1<LinearLayout xmlns:android="
2http://schemas.android.com/apk/res/android"
3    android:orientation="vertical"
4    android:layout_width="fill_parent"
5    android:layout_height="fill_parent"
6    >
7    <TextView
8        android:layout_width="fill_parent"
9        android:layout_height="wrap_content"
10       android:text="@string/title_startseite"
11       />
12
13...
14
15   <Button
16       android:id="@+id/button_startwave"
17       android:layout_marginTop="20sp"
18       android:layout_width="fill_parent"
```

```
19          android:layout_height="wrap_content"
20          android:text="@string/app_wavestart_button"
21          />
22
23 ...
24
25</LinearLayout>
```

In diesem Programmcodeauszug der Datei wird ein *LinearLayout* als Basislayout für die Seite gewählt. Das Android-SDK stellt neben dem *LinearLayout* weitere vier Basislayouts zur Wahl (s.a. [uMP09e]). Ein Basislayout legt die Anordnung der Bildschirmelemente, wie Textfelder oder Buttons fest. Grundlegend besteht die Möglichkeit verschiedene Basislayouta ineinander zu Schachteln und so den Bildschirm in unterschiedliche Bereiche mit eigenen Anordnungen der Bildschirmelemente einzurichten. Jedoch ist dies, im Hinblick auf die begrenzte Rechenleistung eines Smartphones, nicht zu empfehlen.
Zu einem Basislayout können zusätzliche Angaben bezüglich der Ausrichtung und der zu verwendenden Bildschirmoberfläche vorgenommen werden.
Da ein Smartphone entweder vertikal oder horizontal gehalten werden kann, muss im Basislayout definiert werden, für welche der beiden Varianten dieses Layout gedacht ist. Dies hat Auswirkung auf die Ausrichtung aller Elemenete des Layouts. Üblicherweise wird jedes Bildschirmlayout einer Anwendung einmal für die horizontale oder vertikale Ausrichtung des Endgerätes konzipiert. Je nachdem wie das Smartphone gehalten wird, passt sich die Bildschirmausgabe an.
Für *AndroidWave* wird in dieser Version aus Zeitgründen nur die vertikale Bildschirmausgabe definiert(siehe Zeile 3, Listing 4.7). Das heißt, wird das Smartphone horizontal ausgerichtet ändert sich die Ausrichtung des Bildschirmes nicht. Es „liegt alles auf der Seite“, um es umgangssprechlich auszudrücken.
Überdies wird der zu verwendende Bildschirmbereich angegeben. Für jedes Element, auch das Basislayout, bietet *AndroidWave* zwei Möglichkeiten. Entweder es wird nur so viel Platz auf dem Bildschirm verwendet, wie das darzustellende Element benötigt oder es wird der gesamte zu Verfügung stehende Bereich genutzt . Überlicherweise wird bei dem Seitenlayout die gesamte verfügbare Bildschirmfläche genutzt (siehe Zeile 4-5, Listing 4.7).
Im Programmcodeausschnitt wird der Begrüßungstext für den Nutzer und der Button zum Anlegen einer neuen Wave angelegt. Für die Beschriftung der Elemente beziehungsweise dem Angezeigten Begrüßungstext werden Strings aus einer XML-Resourcen-Datei von *AndroidWave* verwendet (*android:text=„@string/...“*). Unter den angegebenen Bezeichnungen

ist in der Datei *string.xml* ein entsprechender String zu finden.
Neben Layout-Definitionen wird für den Button eine ID festgelegt. Über diese ID wird es in der Activity, welche dieses Layout lädt, möglich sein, direkten Zugriff zu nehmen. Jede Angabe bezüglich des Buttons können verändert werden. Darüber hinaus kann so ein *Click-Listener* auf Betätigungen des Buttons reagieren.

Das Layout der Startseite von *AndroidWave* ist in der XML-Datei *main.xml* definiert. Um nun das Layout mit der Activity *StartseiteAnzeigen.java* zu verknüpfen, ist in die *on-Create*-Methode folgende Anweisung vonnöten (siehe Zeile 9, Listing 4.8).

Listing 4.8: Verknüpfung eines Layouts mit einer Activity

```
1 public class StartseiteAnzeigen extends Activity {
2
3     /** Called when the activity is first created. */
4     @Override
5     public void onCreate(Bundle savedInstanceState) {
6
7         ...
8
9         this.setContentView(R.layout.main);
10
11        ...
12    }
13 }
```

Durch die Betätigung des Buttons *Neue Wave starten* soll die Activity StartNewWaveAnzeigen aufgerufen werden (s.a. 3.2.3.2). Dank der im Layout vergebenen ID für den Button, kann in der Activity *StartseiteAnzeigen* auf dieses Bildschirmelement Zugriff genommen werden. Im Listing 4.9 ist ein Ausschnitt des Quellcodes der Activity *StartseiteAnzeigen* von *AndroidWave* dargestellt. In dem Ausschnitt ist die Deklaration einer Button-Variable und ihre Verknüpfung mit dem entsprechenden Button des Layouts durch die definierte ID gezeigt. Desweiteren wird das Verhalten der Activity definiert, wenn eine Betätigung des Buttons registriert wird.
Wird der Button gedrückt, erzeugt ein Intent eine Activity aus der Klasse *StartNewWaveAnzeigen.java*. Die erzeugte Activity wird anschließend gestartet.

Listing 4.9: Deklaration eines Buttons und Implementierung seines Verhaltens

```
1 public class StartseiteAnzeigen extends Activity {
```

```
2
3   /** Called when the activity is first created. */
4   @Override
5   public void onCreate(Bundle savedInstanceState) {
6
7     ...
8
9     Button B_StartWave=(Button)findViewById(R.id.button_startwave);
10
11    B_StartWave.setOnClickListener(new View.OnClickListener() {
12
13            @Override
14        public void onClick(View v) {
15        // TODO Auto-generated method stub
16
17          Intent click = new Intent(StartseiteAnzeigen.this,
18                StartNewWaveAnzeigen.class);
19          StartseiteAnzeigen.this.startActivity(click);
20        }
21    });
22
23    ...
24
25  }
26}
```

4.1.3.2 Implementierung der Verbindung von *StartNewWaveAnzeigen* mit *NewWaveLocalService*

Jeder Activity steht einer der vier *Local Services* zur Verfügung, um Methoden oder Anweisungen der Geschäftslogik in einen Hintergrundprozess auszulagern.
Die Activity *StartNewWaveAnzeigen* verwendet den *Local Services NewWaveLocalService* für diese Zwecke und muss dementsprechend mit ihm verbunden werden. Hierfür sind entsprechende Anweisungen in der Activity *StartNewWaveAnzeigen* und im *Local Services* zu implementieren.

Listing 4.10 zeigt einen Programmausschnitt von *NewWaveLocalService* mit den entsprechenden Zeilen zur Erstellung eines *IBinders*.

Listing 4.10: Implementierung eines Binders in einem *Local Services*

```
1 public class NewWaveLocalService extends Service{
2
3 ...
4
5 private final IBinder newWaveBinder = new newWaveLocalBinder();
6
7 public class newWaveLocalBinder extends Binder {
8
9                 public NewWaveLocalService getService(){
10                        return NewWaveLocalService.this;
11                }
12        }
13
14 ...
15
16        public IBinder onBind(Intent intent){
17                return this.newWaveBinder;
18        }
19
20 ...
21
22 }
```

Zunächst wird eine Variable namens *newWaveBinder* vom Typ *IBinder* deklariert. Instanziert wird sie mit dem Aufruf des Konstruktors der Klasse *newWaveLocalBinder*. Die Klasse *newWaveLocalBinder* wird eigens für diesen Service angelegt und erbt den Funktionsumfang der Android-Klasse *Binder*. Zusätzlich, zu den geerbten Methoden von *Binder*, wird in *newWaveLocalBinder* eine Methode *getService* angelegt, welche ein Objekt des Service *NewWaveLocalService* als Rückgabewert besitzt.
Das mit *newWaveLocalBinder* instanzierte Objekt *newWaveBinder* wird innerhalb des Services mit der Methode *onBind* an die aufrufenden Activity übergeben. Dies geschieht während des Verbindungsaufbaus in der Activity.

Von Seiten der Activity muss zuerst eine Variabele vom Typ *NewWaveLocalService* instanziert werden. Anschießend wird die Variable *mConnection* vom Typ *ServiceConnection* deklariert und instanziert. Der *Local Service NewWaveLocalService* wird wie eine Activity deklariert und gestartet, anschließend wird die Verbindung zwischen Activity und dem Service hergestellt. Listing 4.11 zeigt den entsprechenden Programmcodeausschnitt aus der Activity *StartNewWaveAnzeigen*.

Listing 4.11: Verbindung von Activity und Local Service

```
1 public class StartNewWaveAnzeigen extends Activity{
2
3   public NewWaveLocalService localService =
4                                   new NewWaveLocalService();
5   public ServiceConnection mConnection = new ServiceConnection(){
6     @Override
7     public void onServiceConnected
8                   (ComponentName className, IBinder service) {
9       // TODO Auto-generated method stub
10
11      localService =
12       ((NewWaveLocalService.newWaveLocalBinder)service).
13       getService();
14    }
15
16    @Override
17    public void onServiceDisconnected(ComponentName className) {
18      // TODO Auto-generated method stub
19
20        localService = null;
21    }
22  };
23
24  public void onCreate(Bundle icicle) {
25
26    ...
27
28    Button B_Wave = (Button) findViewById
```

```
29                                  (R.id.button_startnewwave);
30      B_Wave.setOnClickListener(new View.OnClickListener() {
31
32          @Override
33          public void onClick(View v) {
34            // TODO Auto-generated method stub
35
36          Intent newWaveService = new
37            Intent(getApplicationContext(),NewWaveLocalService.class);
38
39          StartNewWaveAnzeigen.this.startService(newWaveService);
40          bindService
41                  (newWaveService,mConnection,Context.BIND_AUTO_CREATE);
42
43          ...
44
45          localService.sendWave();
46
47          ...
48
49    }
```

Sobald der Button zum Senden der neuen Wave betätigt wird, wird wie bei einer Activiy zunächst ein Intent mit dem *Local Service* angelegt, welcher gleich darauf gestartet wird. Nach dem Starten des Intents verbindet sich die Activity durch die Methode *bindService* (siehe Zeile 40-41,Listing 4.11) mit dem Service. Der erste Parameter enthält den Intent, auf den sich die Anweisung beziehen soll. Der zweite Parameter bekommt die Variable *mConnection* übergeben, welche nach dem Verbindungsaufbau die Methode *onServiceConnected* ausführt. Der Variablen *localService*, vom Typ des *Local Services NewWaveLocalService*, wird nun durch die Methode *getService* der Binder-Klasse *newWaveLocalBinder* ein konkretes Objekt der Klasse *NewWaveLocalService* übergeben. Der dritte Parameter sorgt für das Starten des Services, falls dieser noch nicht läuft. Nun kann über die Variable *localService* auf die Methoden des Services zugegriffen werden(siehe Zeile 45, Listing 4.11).

4.1.3.3 Implementierung des DAO-Interfaces *NewWaveDAOInterface* und der *DAO-Factory*

Im Kapitel 3.2.3.2 wird die Funktionsweise des Entwurfsmusters *Factory* erläutert. Die folgenden Zeilen zeigen, wie die Umsetztung für *AndroidWave* konkret erfolgt ist.
Listing 4.12 zeigt zunächst das DAO-Interface *NewWaveDAOInterface*.

Listing 4.12: Das DAO-Interface *NewWaveDAOInterface*

```
1 public interface NewWaveDAOInterface {
2
3         public void createNewWave(NewWaveTransferObject nwTO);
4         public String getFehlerbericht();
5         public boolean getSendStatus();
6 }
```

Wie dem Kapitel 3.2.3.2 zu entnehmen ist gibt das Interface nur den Rumpf einer Methode vor, also die Sichtbarkeit der Methode, den Typ des Rückgabewertes, den Name und die Parameterlist. Erst in der konkreten DAO-Klasse wird die eigendliche Logik implementiert, wie Listing 4.13 zeigt.

Listing 4.13: Das DAO-Klasse *NewWaveDAOImpl*

```
1 public class NewWaveDAOImpl implements NewWaveDAOInterface {
2
3         private String responseText;
4         private boolean status = false;
5
6         @Override
7         public void createNewWave(NewWaveTransferObject nwTO) {
8
9                 HttpURLConnection meineVerbindung = null;
10                int httpCode = 0;
11
12                ...
13        }
14
15        @Override
16        public String getFehlerbericht() {
```

```
17                  // TODO Auto-generated method stub
18                  return this.responseText;
19          }
20
21          @Override
22          public boolean getSendStatus() {
23                  // TODO Auto-generated method stub
24                  return this.status;
25          }
26}
```

In dem *Local Service NewWaveLocalService* wird eine Variable *newWaveDAO* vom Typ *NewWaveDAOInterface* deklariert. Instanziert wird sie jedoch mit der konkreten DAO-Klasse *NewWaveDAOImpl*. Dies geschieht durch den Aufruf der Methode *buildNewWaveDAO* aus der *DAOFactory*. Dadurch steht der Variablen beziehungsweise dem Objekt *newWaveDAO* der konkrete Funktionsumfang der DAO-Klasse *NewWaveDAOImpl* zur Verfügung. Angenommen, es muss eine neue DAO-Klasse *NewWaveDAOImpl* geschrieben werden, so ändert sich höchstens der Konstruktoraufruf in der Methode der *DAOFactory*. Sofern sich an das Entwurfsmuster gehalten und nur Methoden aus der DAO-Klasse für den Informationsaustausch über das Objekt *newWaveDAO* an den *Local Service* verwendet wird, kann eine konkrete DAO-Klasse zu jeder Zeit durch eine neue ersetzt werden.
Listing 4.14 zeigt die Methode *buildNewWaveDAO* in der *DAOFactory*. Die Deklaration und Instanzierung der Variablen *newWaveDAO* im *Local Service NewWaveLocalService* ist dem Listing 4.15 zu entnehmen.

Listing 4.14: Ausschnitt der DAO-Factory

```
1public class DAOFactory {
2          public static NewWaveDAOInterface buildNewWaveDAO(){
3                  return new NewWaveDAOImpl();
4          }
5
6          public static ContinueWaveDAOInterface
7                                  buildContinueWaveDAO(){
8                  return new ContinueWaveDAOImpl();
9          }
10
```

```
11          ...
12 }
```

Listing 4.15: Ausschnitt des *Local Service NewWaveLocalService*

```
1 public class NewWaveLocalService extends Service{
2
3          ...
4
5          private NewWaveDAOInterface newWaveDAO =
6                                  DAOFactory.buildNewWaveDAO();
7
8          ...
```

4.1.4 Wave fortsetzten

Der Nutzer ist mit *AndroidWave* in der Lage, bestehende Waves abzurufen. In der Activity *StartseiteAnzeigen* ist neben dem Button *neue Wave starten* ein zweiter Button *Wave fortsetzen* vorhanden. Nach der Betätigung des Buttons wird eine Aktivity *ContinueWaveAnzeigen* aufgerufen. Es wird nun grob der Ablauf dieser Programmfunktion erläutert.

Nach dem Start einer *ContinueWaveAnzeigen*-Activity lädt diese zunächst die Login-Daten aus der Einstellungsdatenbank. Daraufhin verbindet sie sich mit dem *Local Service ContinueWaveLocalService*. Dieser deklariert eine Variable vom Typ *ContinueWaveDAOInterface* und wird anschließend mit der konkreten DAO-Klasse *ContinueWaveDAOImpl* instanziert. Desweiteren wird eine Variable vom Typ *ContinueWaveAccessDataTransferObject* deklariert und instanziert. Diese Transferobjekt-Variable erhält durch Setter-Methode des *ContinueWaveLocalService* die Einstellungsdaten von der Activity *ContinueWaveAnzeigen*.
Durch den Aufruf der Methode *getAllAvaiableWaves* aus dem Objekt der DAO-Klasse *ContinueWaveDAOImpl* wird das Transferobjekt mit den Einstellungsdaten an die DAO übergeben. Diese führt nun einen HTTP-Post-Request aus. Ein PHP-Script der Serverseite erhält die Login-Daten und führt eine Datenbankabfrage zur Erfassung aller verfügbaren Waves aus. Dabei wird eine Tabelle mit der WaveID, der Nachricht des Root-Blibs, der ersten Wavelet einer Wave, dazugehörigem Zeitstempel und Autor erstellt. Diese Tabelle wird als JSON-Array in einem HTTP-Response zurück an die DAO von *AndroidWave* gesendet. Wie die Extrahierung des JSON-Arrays aus dem HTTP-Response erfolgt, ist dem anschließenden Kapitel zu entnehmen.
Das extrahierte JSON-Array wird kurzfristig in einer Variablen *jArray_AllBlibs* zwischengespeichert.
Die DAO senden einen zweiten HTTP-Request an den GoogleWave-Dummy, um die Adresslisten aller Wavelets zu jeder Wave als weiteres JSON-Array zu erhalten. Der Grund für das Senden von zwei seperaten HTTP-Requests ist, dass die Datenbankanfragen in voneinander getrennten PHP-Scripten ausgeführt werden. Auch das im zweiten HTTP-Request befindliche JSON-Array wird nun extrahiert und in der Variablen *jArray_AllAdress* zwischengespeichert. Beide Arrays werden nun der Methode *fillTO* übergeben. Ziel der Methode ist es alle Root-Blibs aus dem JSON-Array *jArray_AllBlibs* in ein ein-dimensionale Array-Objekt vom Typ *ContinueWaveGetAllAvaiableWaveTransferObject* namens *allWaveTO* zu laden. Jeder Eintrag repräsentiert nun eine Wave und ihre erste Nachricht, welche nun als Betreff verwendet wird. Zu jedem Wave-Eintrag werden nun die Adresslisten ihrer Wavelets ergänzt. Zuvor werden die Empfängerlisten der Wavlets einer Wave auf doppelte Einträge untersucht. Doppelte Einträge können entstehen, wenn mehrere Wavelets einer Wave für die selbe Person

sichtbar sein soll. Dopplungen sind in dem Fall unerwünscht und werden gelöscht. So erhält der Nutzer später eine Zusammenfassung aller Empfängerlisten der Wavelets einer Wave.
Das Array-Objekt *allWaveTO* wird nun wieder durch eine Getter-Methode in der DAO an den *Local Service ContinueWaveLocalService* zurückgegeben, welcher seinerseits das empfangene Array-Objekt durch eine Getter-Methode Activity *ContinueWaveAnzeigen* zur Verfügung stellt. Die Activity *ContinueWaveAnzeigen* gibt nun den Inhalt des Array-Objektes *allWaveTO* auf dem Bildschirm aus.

In den Kapiteln 4.1.4.1 und 4.1.4.2 wird nun die Extrahierung des JSON-Arrays aus dem HTTP-Response in ein geeignetes Java-Datenformat und die Bildschirmausgabe des Array-Objektes *allWaveTO* erläutert.

4.1.4.1 JSON-Array

Kurzeinführung - Das JSON-Datenformat Das JSON-Datenformat (JavaScript Object Notation) ist ein schlankes Format für den Datenaustausch zwischen Server und Client. Es handelt sich dabei um eine Art Zeichenkette in der Name/Werte-Paare aneinandergereiht werden können, ähnlich einer Hash-Tabelle aus der Programmiersprache Java. JSON ist dabei von Programmiersprachen komplett unabhängig. In den meisten Programmiersprachen existieren entsprechende Parser, um den Inhalt einer JSON-Zeichenkette in ein eigenes Datenformat zu extrahieren.
Eine Gruppe Name/Wert-Paaren wird als Objekt bezeichnet. Beginn und Ende eines Objektes wird mit einer „{“ beziehungsweise „}“ gekennzeichnet. Innerhalb eines Objektes können mehrere Name/Wert-Paare getrennt mit einem „,“ aneinandergereiht werden. Bei einem Name/Wert-Paar wird zunächst der Schlüssel angegeben, daraufhin folgt, getrennt durch ein „:“ der Wert.
Mit dem JSON-Datenformat können Arrays dargestellt werden. Der Beginn beziehungsweise das Ende eines Arrays wird mit einer eckigen Klammer auf beziehungsweise eckige Klammer zu gekennzeichnet. Innerhalb eines Arrays können beliebig viel Objekte angeordnet werden, sie werden durch ein „,“ getrennt.

Das Ergebnis einer Datenbankanfrage ist immer eine mit Werten gefüllte Tabelle. Um die Tabelle vom GoogleWave-Dummy zum *AndroidWave*-Client senden zu können, wird diese in ein JSON-Array übertragen und abgesendet. Das empfangene JSON-Datenformat muss nun in ein für Java geeignetes Datenformat umgeformt werden. Der nächste Abschnitt erläutert das Vorgehen anhand der DAO-Klasse *ContinueWaveDAOImpl*.

Listing 4.16: Ausschnitt der DAO-Factory

```
1 private JSONArray getRootBlib
2 (ContinueWaveAccessDataTransferObject coWaveTO){
3
4         String result="";
5         InputStream inputStr = null;
6         JSONArray jArray = null;
7
8         ...
9
10        try{
11                // Herstellen der Internetverbindung
12
13                ...
14
15                HttpResponse response =
16                httpClient.execute(httppost);
17                HttpEntity entity = response.getEntity();
18                inputStr = entity.getContent();
19        }catch(Exception e){
20                Log.i("ContinueWaveDAOImpl: ",
21        "Fehler bei der Internetverbindung: "+e.toString());
22        }
23        try{
24                BufferedReader reader = new BufferedReader
25        (new InputStreamReader(inputStr,"iso-8859-1" ),10);
26                StringBuilder sb = new StringBuilder();
27                String line = null;
28                while((line = reader.readLine()) != null){
29                        sb.append(line +"\n");
30                }
31                inputStr.close();
32                result=sb.toString();
33        }catch(Exception e){
34                Log.i("ContinueWaveDAOImpl: ",
```

```
35                  "Fehler bei Konvertierung: "+e.toString());
36          }
37          try{
38                   jArray = new JSONArray(result);
39                  }catch(JSONException e){
40                          Log.i("ContinueWaveDAOImpl",
41                          "Error parsing data "+e.toString());
42                  }
43                  return jArray;
44}
```

Das Listing 4.16 zeigt einen Auszug aus dem Quellcode der konkreten DAO-Klasse *ContinueWaveDAOImpl*, in der ein JSON-Array per HTTP-Response empfangen und für die weitere Verarbeitung in ein Java eigenes JSON-Datenformat konvertiert wird.
Die DAO-Klasse erhält einen HTTP-Response. Inhalt der Übertragung ist ein JSON-Array mit einer Datenbanktabelle. Die Tabelle enthält Informationen alle verfügbaren Waves, welche dem Nutzer zugänglich sind.
Der gesamte HTTP-Response wird der Variabelen *inputStr* vom Typ *InputStream* übergeben. In der zweiten „Try-Catch"-Anweisung übergibt *inputStr* den Response einem *BufferedReader*. Anschließend wird der *BufferedReader* Zeile für Zeile ausgelesen. Jede Zeile wird einem *StringBuilder* übergeben. Sobald die letzte Zeile des *BufferedReader* ausgelesen und dem *StringBuilder* übergeben ist, erhält die gewöhnliche String-Variable *result* die erzeugte Zeichenkette. Bisher dienten alle Anweisungen lediglich dazu, den erhaltenen HTTP-Response vom GoogleWave-Dummy in einen String zu konvertieren. In der dritten „Try-Catch"-Anweisung wird in dem Einzeiler die String-Variabel *result* mit der Instanzierung der JSONArray-Variabel *jArray* übergeben. Jedes JSON-Objekt entspricht nun einem Element des Arrays, auf welches durch Angabe des Indizes Zugriff genommen werden kann. Beispiel: *jArray.getJSONObject(i)*
Diese Anweisung liefert als Ergebnis ein JSON-Objekt. Dieses Ergebnis kann einer Variablen zugewiesen werden: *JSONObject json_data = jArray.getJSONObject(i)*. Mit Hilfe dieser Variabel kann auf die Name/Wert-Paare des JSON-Objektes durch Angabe des Schlüssels zugegriffen werden: *int a = json_data.getInt("WaveID")*. Diese Anweisung liefert den Wert zum angegebenen Schlüssel. Als Ergebnis wird in dem Beispiel die WaveID als Integer-Wert an die Variable *a* übergeben. Innerhalb der DAO-Klasse werden so alle Name/Wert-Paare der JSONArrays ausgelesen, um sie in dem Transferobjekt-Array *allWaveTO* zu vereinen. Das Transferobjekt wird anschließend dem *Local Service* übergeben.

4.1.4.2 Die Activity *ListActivity*

Für die Ausgabe des Array-Objektes *allWaveTO* wird in *AndroidWave* eine spezielle Art der Activity verwendet. Die Activity *ContinueWaveAnzeigen* wird nicht wie *StartNewWaveAnzeigen* von der Klasse *Activity* abgeleitet sonder von der Android-Klasse *ListActivity*. Diese Activity gibt das Bildschirmlayout vor und nutzt als Basislayout die *ListView*. Der *ListActivity* wird hierfür ein ein-dimensoinales String-Array übergeben. Jedes Element des Arrays wird als einzelnes Bildschrimelement erkannt und dargestellt. Die Größe des String-Arrays und somit die maximale Anzahl der anzeigbaren Element ist prinzipiell unbegrenzt. Aus der so entstehenden List können einzelne Elemente ausgewählt werden. Ein *Click-Listener* erfasst hierfür den Selektionsversuch des Nutzers.
Für die Ausgabe aller verfügbaren Waves stellt die *ListActivity* somit eine komfortable und flexible Lösung dar.
Ausschließlich der Inhalt des Array-Objektes *allWaveTO* muss angemessen in einen String für das String-Array formatiert werden.
Listing 4.16 zeigt den hierfür notwendigen Programmcode.

Listing 4.17: Ausgabe des Array-Objektes *allWaveTO* auf dem Bildschirm

```
1 public class ContinueWaveAnzeigen extends ListActivity{
2
3         ...
4
5     private
6           ContinueWaveGetAllAvaiableWaveTransferObject[]
7           allWaveTO;
8     private String[] WaveArray;
9     private int waveCount = 0;
10
11    ...
12
13    public void onCreate(Bundle icicle) {
14
15        ...
16
```

```
17          localService.getAllAvaiableWaves();
18          this.waveCount=localService.getAvaiableWaveCount();
19          this.allWaveTO=localService.getAvaiableWave();
20
21          this.WaveArray= new String[this.waveCount];
22          this.makeTOtoString();
23          this.Liste();
24
25          ...
26
27      }
28
29      private void makeTOtoString() {
30          // TODO Auto-generated method stub
31          for(int i = 0;i<this.waveCount;i++){
32              this.WaveArray[i]="\n"+"Wave: "
33              +this.allWaveTO[i].WaveID+"\n"+"Thema: "
34              +this.allWaveTO[i].contribution+"\n"+"Adressaten: "
35              +this.allWaveTO[i].adress+"\n"+"Erstellt am: \n"
36              +this.allWaveTO[i].date+" \n"
37              +this.allWaveTO[i].time+"\n"+"Autor: "
38              +this.allWaveTO[i].author_Vorname+" "
39              +this.allWaveTO[i].author_Name+"\n"; } }
40
41      private void Liste(){
42          ListAdapter avaiableWaves = new
43          ArrayAdapter<String>
44          (this,android.R.layout.simple_list_item_1,this.WaveArray);
45          this.setListAdapter(avaiableWaves);
46
47          }
48
49          ...
50  }
```

Zunächst werden drei Instanzvariablen deklariert. Die Variable *allWaveTO* nimmt das Array-Objekt durch die Getter-Methode *getAvaiableWave* aus dem *Local Service ContinueWaveLocalService* auf (siehe Zeile 19 Listing 4.17). Zuvor bekommt die Instanzvariable *waveCount* die des Array-Objektes vom *Local Service* übermittelt (siehe Zeile 18 Listing 4.17). Die Größe des als Instanzvariable deklarierte String-Array *WaveArray* wird in Zeile 21 mit der Instanzvariable *waveCount* bestimmt. Daraufhin startet die Methode *makeTOtoString*, welche die Elemente des Arrays *allWaveTO* als formatierten String in das Array *WaveArray* überträgt.

Nun wird die Methode *Liste* ausgeführt, um das String-Array *WaveArray* dem vordefinierten Layout der *ListActivity ContinueWaveAnzeigen* als Adapter zu übergeben. Alle verfügbaren Waves werden nun mit allen Informationen übersichtlich strukturiert auf dem Bildschirm ausgegeben.

Wählt der Nutzer nun aus einer der aufgelisteten Waves, wird die entsprechende WaveID ermittelt und eine weitere *ListActivity* namens *WaveletAnzeigen* gestartet. Diese Activity verfügt abermals über einen *Local Service*, welcher über eine eigene DAO alle zu der WaveID passenden Wavelets ermittelt. Die ermittelten Wavelets werden im selben Verfahren wie die Waves auf dem Bildschirm ausgegeben. Der einzige Unterschied ist, dass das letzte Element keine Verfügbare Wavelet repräsentiert, sondern das Anlegen einer neuen Wavelet innerhalb der Wave ermöglicht. Für das Anlegen eines neuen Wavelets wird eine entsprechende Activity namens *NewWaveletAnzeigen* gestartet. Die Funktionsweise dieser Activity ist mit der zum Anlegen einer neuen Wave identisch. Es wird überdies die selbe Layout-XML-Datei *startnewwave.xml* verwendet. Einzig die, für das Anlegen der neuen Wavelet, an den entsprechenden *Local Service* übergebenen Daten werden um die WaveID erweitert.

Wird eine bestehende Wavelet auf dem Bildschirm ausgewählt, anstatt eine neue Wavelet zu erzeugen, wiederholt sich der Vorgang wie schon zuvor bei der Abfrage der verfügbaren Waves beziehungsweise Wavelets. Ein entsprechender *Local Service* ermittelt mit Hilfe zuständiger DAO-Komponenten alle Blibs einer Wavelet und sendet diese wieder als JSON-Array in einem HTTP-Response an *AndroidWave*, wo diese auf dem Bildschirm anzeigt werden. Auch hier wird die List der Blibs um ein letztes Element erweitert, welches das Anlegen einer neuen Blib ermöglicht. Analog zum Anlegen einer Wave oder eines Wavelets wird auch hier eine Activity mit der selben Funktionsweise erzeugt. Der einzige Unterschied hier ist, dass die Empfängerlist bereits ausgefüllt ist. Es wird die der Wavelet übernommen, in der die neue Blib angelegt wird. Die Liste der Empfänger ist jedoch weiter editierbar und kann so um neue Empfänger erweitert werden. Dementsprechend wird bei dem Erstellen der neuen Blib, die Adressliste der dazugehörigen Wavelet aktualisiert.

Wählt der Nutzer eine bereits bestehende Blib aus, wird auch hier eine Activity, gleich der zum Anlegen von Wave, Wavelet oder Blib gestartet. Hier ist nicht nur die Empfängerlist bereits ausgefüllt sondern auch das Textfeld für die Nachricht. Hier ist es dem Nutzer möglich, die bereits bestehende Nachricht der Blib zu editieren.

4.1.5 Schnittstellen zum GoogleWave-Dummy

Da es *AndroidWave* nicht möglich ist direkt auf die MySQL-Datenbank des GoogleWave-Dummys zuzugreifen sind Schnittstellen in From von PHP-Scripte implementiert worden. Die PHP-Scripte lassen sich hinsichtlich ihrer Aufgaben in zwei Gruppen unterteilen. Zum einen existieren PHP-Scripte, um auf der Datenbank Einträge, das heißt Waves, Wavelets oder Blibs anzulegen. Zum anderen sind auf dem Server PHP-Scripte vorhanden, um Wave, Wavelets oder Blibs von der Datenbank auszulesen. Im nächsten Kapitel wird die Gemeinsamkeit aller PHP-Scripte erläutert, das Aufbauen einer Verbindung und Anmelden bei der MySQL-Datenbank vom GoogleWave-Dummy. Anschließend werden entscheidene Programmcodestellen der jeweiligen PHP-Script-Gruppen anhand eines Beispiels erklärt.

4.1.5.1 Zugriff auf die Datenbank

Jedes PHP-Script auf dem GoogleWave-Dummy muss zunächst eine Verbindung zu der MySQL-Datenbank aufbauen und sich auf der Datenbank anmelden. Die notwendigen Anweisungen sind dem Listing 4.18 zu entnehmen.

Listing 4.18: Verbindungsaufbau und Authentifikation auf dem GoogleWave-Dummy

```
1<?php
2   $host = $_REQUEST['host'];
3   $username = $_REQUEST['username'];
4   $password = $_REQUEST['password'];
5
6   $con = mysql_connect("$host","$username","$password");
7   if(!$con){
8     die('Kann nicht auf Datenbank zugreifen:'.mysql_error());
9   }
10
11  mysql_select_db("GoogleWave_Dummy",$con);
12
13  ...
14
15  mysql_close($con);
16?>
```

Bevor die Verbindung zur Datenbank hergestellt werden kann, muss jedes Script die Login-Daten aus dem HTTP-Request der DAOs lesen (siehe Zeile 2-4, Listing 4.18). Anschließend kann die Verbindung zum Server der Datenbank aufgebaut werden (siehe Zeile 6, Listing 4.18). Sollte es bei dem Verbindungsaufbau zu einem Fehler kommen, wird dieser an die DAO von *AndroidWave* zurückgesendet. Der Exceptoion-Handler für die HTTP-Verbindung der DAOs wirft eine entsprechende Exception und leitet den Fehlerbericht über Getter-Methoden zur jeweiligen Activity, welche den Fehlerbericht auf dem Bildschirm ausgibt.
Sind bei dem Verbindungsaufbau zum Server keine Fehler aufgetreten, wird die Datenbank *GoogleWave_Dummy* ausgewählt(siehe Zeile 11, Listing 4.18). Nun erfolgen, je nach Aufgabe des PHP-Scriptes entsprechende Anweisungen beziehungsweise Datenbankanfragen.
Nach Beendigung aller Anweisungen, wird am Ende jedes Scripte die Verbindung zum Server geschlossen (siehe Zeile 15, Listing 4.18).

4.1.5.2 Anlegen von Einträgen

Das Vorgehen beim Anlegen einer neuen Wave, Wavelet oder Blib wird am Beispiel des PHP-Scriptes *setNewWave.php* erläutert.

Listing 4.19: Anlegen einer neuen Wave mit dem PHP-Script *setNewWave.php*

```
1<?php
2          ...
3
4          $date = $_GET['date'];
5          $time = $_GET['time'];
6          $adresslist = $_GET['adresslist'];
7          $contribution = $_GET['contribution'];
8
9          // ERSTELLEN EINES NEUEN WAVE EINTRAGES
10         $queryDB = mysql_query("SELECT MAX(WaveID) FROM Wave");
11         $row=mysql_fetch_array($queryDB);
12         $newWaveID = $row[0] + 1;
13         $queryDB = mysql_query
14         ("INSERT INTO Wave VALUE($newWaveID, 1)");
15
16         // ERSTELLEN EINES NEUEN ZEITSTEMPEL EINTRAGES
17         $queryDB = mysql_query
```

```
18          ("SELECT MAX(ZeitstempelID) FROM Zeitstempel");
19          $row=mysql_fetch_array($queryDB);
20          $newZeitstempelID = $row[0] + 1;
21          $queryDB = mysql_query
22("INSERT INTO Zeitstempel VALUE
23($newZeitstempelID, $date, $time)");
24
25
26          // ERSTELLE EINES NEUEN ADRESSLISTENWAVELET EINTRAGES
27          $queryDB = mysql_query
28          ("SELECT MAX(AdresslistID) FROM AdressListWavelet");
29          $row=mysql_fetch_array($queryDB);
30          $newAdresslistID = $row[0] + 1;
31          $queryDB = mysql_query
32          ("INSERT INTO AdressListWavelet VALUE
33($newAdresslistID, $adresslist)");
34
35          // ERSTELLEN EINES NEUEN WAVELET EINTRAGES
36          $queryDB = mysql_query
37          ("INSERT INTO Wavelet VALUE
38($newWaveID, 1, $newZeitstempelID, $newAdresslistID)");
39
40          // HOLE AUTHORID
41          $queryDB = mysql_query
42("SELECT AuthorID FROM Author WHERE Username = '$username'");
43          $row=mysql_fetch_array($queryDB);
44          $AuthorID = $row[0];
45
46          // ERSTELLEN EINES NEUEN BLIB EINTRAGES
47          $queryDB = mysql_query
48          ("INSERT INTO Blib VALUE
49($newWaveID, 1, 1, $contribution, $newZeitstempelID, $AuthorID)");
50
51          ...
52?>
```

Im HTTP-Request der DAO *newWaveDAO* werden neben den Login-Daten Variablen für den Zeitstempel, die Empfängerliste sowie die Nachricht des Root-Blibs der Wave mitgesendet. Die Werte werden aus dem HTTP-Request in Variabeln des PHP-Scriptes geschrieben (siehe Zeile 4-7, Listing 4.19).

Das Anlegen einer neuen Wave beeinflusst alle Tabellen der Datenbank. So wird als erstes ein neuer Eintrag in der Tabelle *Wave* angelegt. Die ID einer Wave wird in einem Integerwert ausgedrückt und fortlaufend nummeriert. Für die ID der neuen Wave, wird die Nummer der letzten WaveID erfasst und um eins erhöht. Als nächstes wird eine neue Wavelet für den Root-Blib angelegt. Zuvor muss jedoch ein neuer Zeitstempel und eine neue Empfängerlist in die Datenbank geschrieben werden. Auch hier wird für die jeweilige ID eine fortlaufende Nummerierung gewählt.

Im *INSERT*-Befehl der Wavelet fällt auf, dass alle Parameter aus Variablen bestehen, außer die zweite. Da dies eine neue Wave ist und in jedem Fall eine Blib enthält, muss folglich eine Wavelet existieren, zu welcher die Blib zugeordnet werden kann. Daher wird die erste Wavelet einer neuen Wave automatisch erzeugt. Auch die ID der Wavelets werden innerhalb einer Wave fortlaufend mit der Zahl eins beginnend nummeriert.

Bevor die Root-Blib der neuen Wave in die Datenbank geschrieben werden kann, muss der Autor bestimmt werden. Jeder Nutzer muss einen Account auf dem GoogleWave-Dummy besitzten. Da unter anderem mit Hilfe des Nutzernamen des Accounts der Login auf der Datenbank erfolgt. Anhand des beim Einloggen verwendeten Nutzernamens kann der Autor der neuen Wave identifiziert werden. Auch die AuthorID jedes Nutzers ist fortlaufend nummeriert. Die letzte Anweisung des PHP-Scriptes legt die Root-Blib an. Da die Root-Blib die erste Blib in einem Wavelet ist, erhält sie die ID eins. Alle IDs der folgenden Blibs der Wavelet werden ebenfalls fortlaufend nummeriert.

4.1.5.3 Auslesen von Einträgen

Alle PHP-Scripte zum Auslesen von Daten aus der Datenbank sind ähnlich strukturiert. Zu Beginn der Scripte werden die Werte aus dem HTTP-Request in lokale Variablen übergeben. Daraufhin erfolgt die Anmeldung bei dem Server. Nachdem die Datenbank *GoogleWave_Dummy* ausgewählt wurde erfolgt eine Datenbankanfrage. Die Ergebnisse werden immer in Tabellen ausgegeben. Die Inhalte der Tabellen variieren je nach Anfrage

- alle Waves und ihren Rootblibs
- alle Wavelets einer Wave und ihren ersten Blibs
- alle Blibs einer Wavelet

- Adresslisten der Wavelets

Die Tabellen werden nun als JSON-Array per HTTP-Response zurück an *AndroidWave* gesendet.

4.2 Benutzeroberfläche

Im Folgendem wird gezeigt wie die Benutzeroberfläche von *AndroidWave* umgesetzt wurde. Bei der Umsetzung wurde auf ein übersichtliches Layout der Bildschirmausgaben geachtet.

4.2.1 Startbildschirm

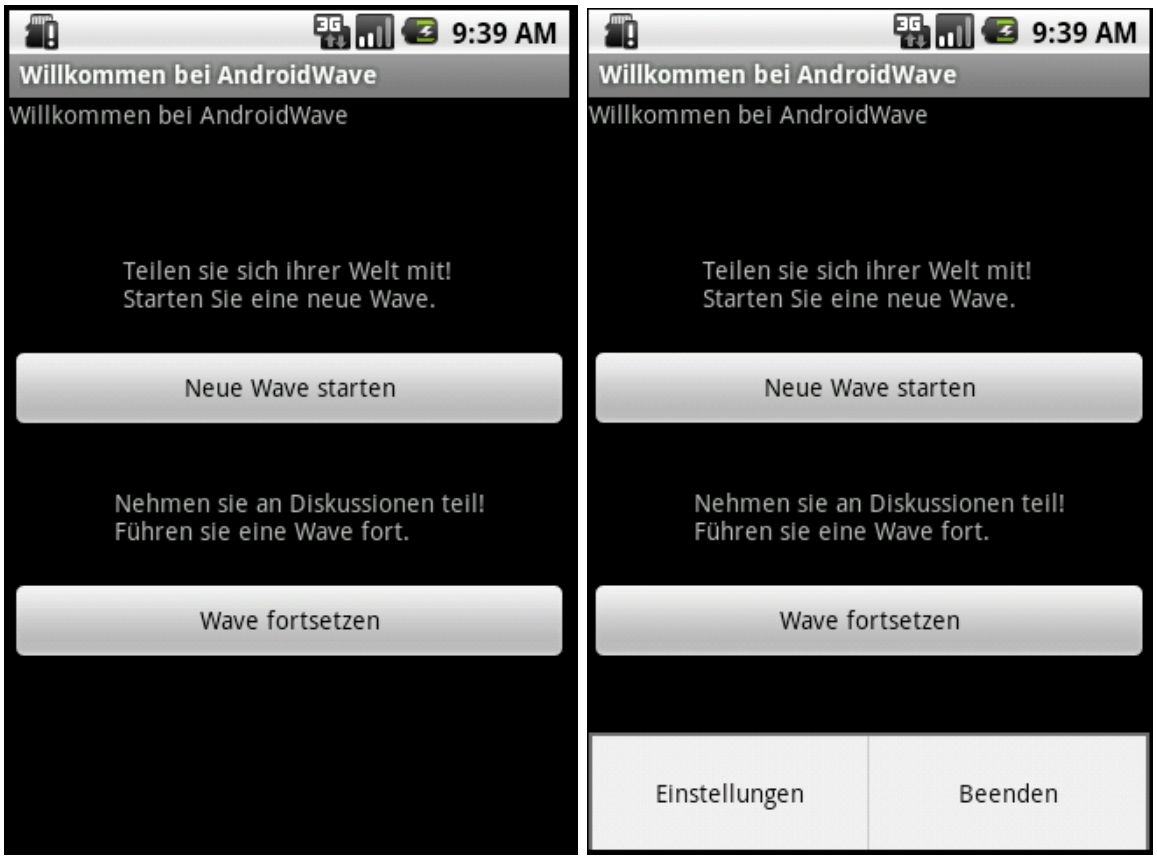

Abbildung 4.1: Startseite von AndroidWave

Die Abbildung 4.1 links, zeigt die Startseite von *AndroidWave*. Auf der Startseite sind zwei Schaltflächen zu finden. Mit der Schaltfläche *Neue Wave starten* wird eine entsprechende Bildschirmausgabe aufgerufen (s.a. 4.2.3). Der Nutzer kann nun eine neue Wave anlegen.
Mit der Schaltfläche *Wave fortsetzen* gelangt der Nutzer zu einer Bildschirmausgabe, welche ihm alle verfügbaren Waves seines Accounts ausgibt. Auf jedem Smartphone mit Android als Betriebssystem existiert eine Taste mit der das Menü einer angezeigten Bildschirmausgabe aufgerufen werden kann. Dieses Menü wird standardmäßig am unteren Rand des Bildschirmes angezeigt(Abbildung 4.1rechts).
Wird das Menü dieses Bildschirmlayouts aufgerufen, kann der Nutzer die Applikation beenden oder die Einstellungsoptionen aufrufen.

4.2.2 Einstellungen

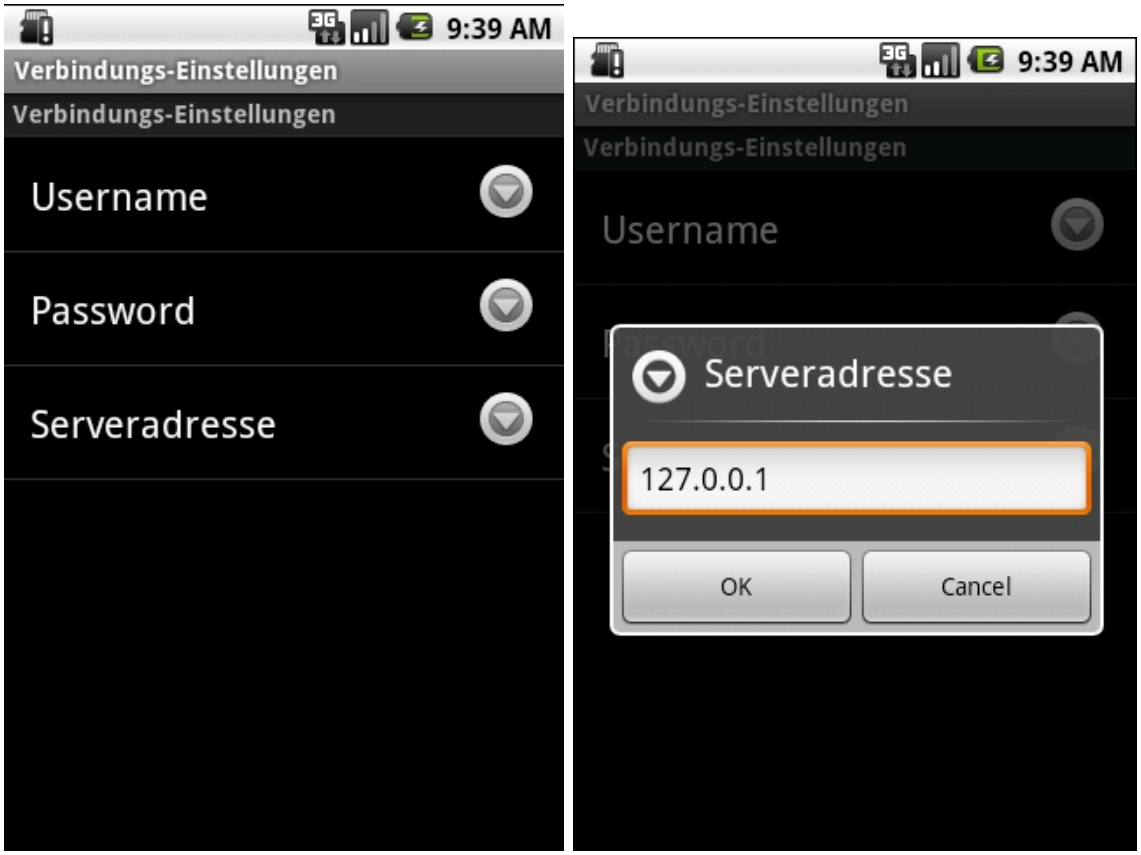

Abbildung 4.2: Einstellungsoptionen

In den Einstellungsoptionen stehen dem Nutzer die drei editierbare Einstellungen für den Zugriff auf den GoogleWave-Dummy zur Verfügung:

- Username
- Passwort
- Serveradresse

Wird eine Einstellungsoption ausgewählt, öffnet sich ein kleines Fenster, in dem die entsprechende Eingabe getätigt werden kann(Abbildung 4.2rechts). Zurück zur Startseite gelangt der Nutzer durch die Betätigung der „Zurück“-Taste, welche auf jedem Android-Smartphone zu finden ist.

4.2.3 Neue Wave starten

Abbildung 4.3: Konversationsbildschirm

Durch das Betätigen der Schaltfläche *Neue Wave starten* gelangt der Nutzer zu der in Abbildung 4.3 dargestellt Bildschirmausgabe. Im obersten Eingabefeld werden die E-Mail-Adressen eingetragen. Dies ist die Empfängerliste.
Im darunter liegendem Textfeld wird die zu sendende Nachricht geschrieben. Betätigt der Nutzer nun die Schaltfläche *Senden*, wird die neue Wave abgeschickt.
Ruft der Nutzer das Menü dieser Bildschirmausgabe auf, kann er entweder zur Startseite zurückkehren oder die Funktion Kontakte aufrufen(Abbildung 4.3rechts). Diese Funktion ist in dieser Version aus Zeitgründen nicht implementiert. In späteren Versionen wird es möglich sein, die Kontaktdatenbank des Systems auf E-Mail-Adressen zu durchsuchen. Die so ermittelten E-Mail-Adressen können dann durch eine einfach Selektion in die Empfängerliste kopiert werden.

Der hier dargestellte Bildschirm wird ebenfalls für das Erstellen einer neuen Wavelet oder Blib genutzt. Auch das Ändern einer Blib wird mit diesem Layout umgesetzt.
Wird eine neue Blib angelegt, so ist die Empfängerliste nicht leer. Sie wird mit der Emp-

fängerliste der Wavelet ergänzt in der die Blib angelegt werden soll. Fügt der Nutzer eine weitere E-Mail-Adresse hinzu, wird die Empfängerliste dementsprechend ergänzt.
Ändert ein Nutzer eine Blib wird ebenfalls dieses Layout genutzt. Die Eingabefelder werden mit den entsprechenden Informationen, das heißt der Empfängerliste und Nachricht der zu ändernden Blib gefüllt. Wird die geänderte Blib abgesendet, so werden entsprechende Aktualisierungen auf dem GoogleWave-Dummy vollzogen.

4.2.4 Wave fortsetzen

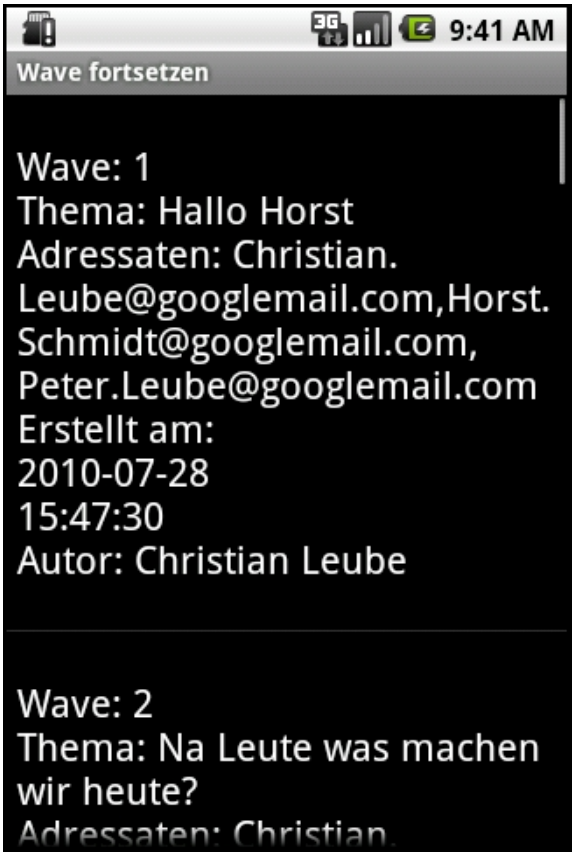

Abbildung 4.4: Bildschirmausgabe aller verfügbaren Waves

Möchte der Nutzer alle ihm verfügbaren Waves einsehen, betätigt er die Schaltfläche *Wave fortsetzen* auf der Startseite von *AndroidWave*. Die benötigten Daten werden vom Server geladen und auf dem Bildschirm angezeigt. Da nicht alle Elemente auf einmal dargestellt werden können, ist das hoch- beziehungsweise runter-„scrollen“ der Elemente möglich. Die Elemente können überdies „angeklickt“ werden. So gelangt der Nutzer zu den Wavelet der Selektierten Wave.

4.2.4.1 Wavelet

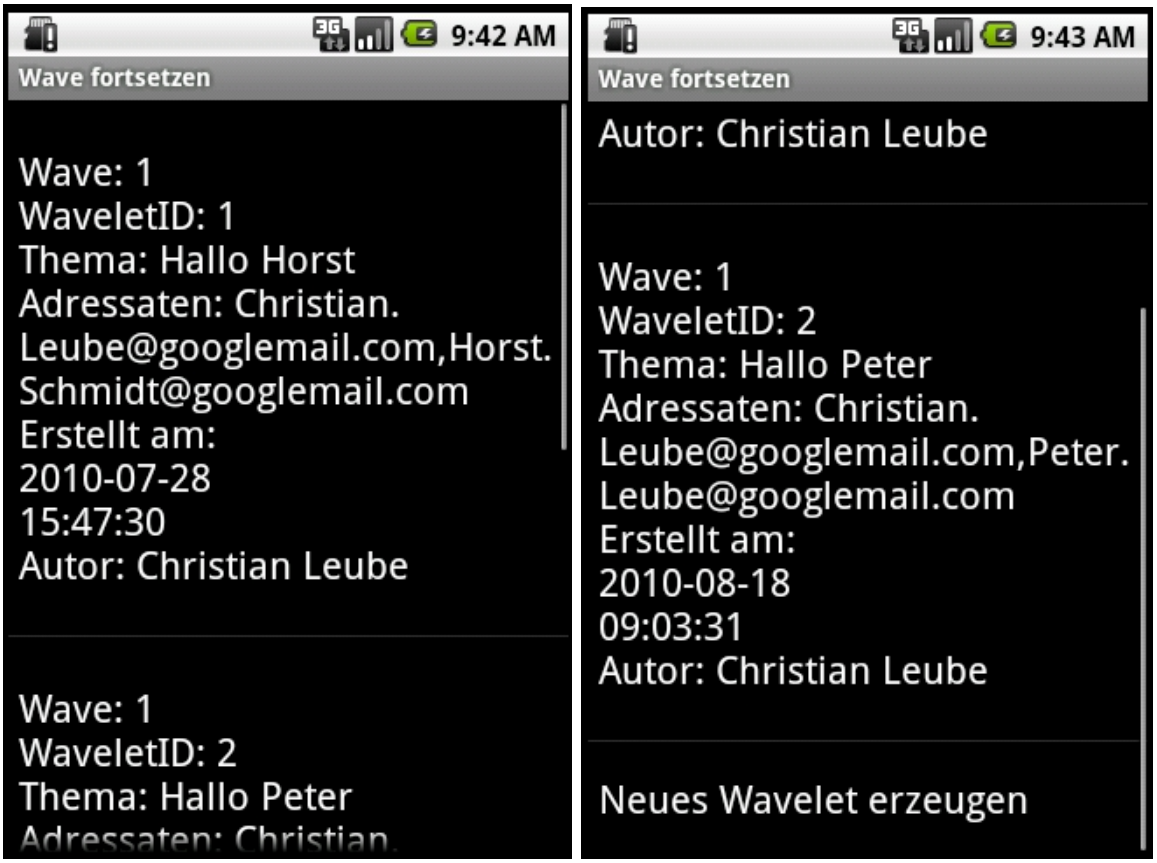

Abbildung 4.5: Bildschirmausgabe aller Wavelets einer Wave

Selektiert der Nutzer eine Wave gelangt er zu dieser Bildschirmausgabe. Auch hier ist das hoch- beziehungsweise runter-„scrollen“ aller Elemente möglich, da eventuell nicht alle gleichzeitig angezeigt werden können. Das letzte Element ist keine Wavelet der Wave(Abbildung 4.5rechts unten). Es repräsentiert eine Schaltfläche mit dem eine neue Wavelet in dieser Wave angelegt werden kann. Betätigt er dieses Element, so gelangt er zu einem Bildschirmlayout, wie in Abbildung 4.3 dargestellt ist.
Selektiert er eine Wavelet, so gelangt er zu einer Bildschirmausgabe, welche ihm alle Blibs der Wavelet anzeigt.

4.2.4.2 Blib

Abbildung 4.6: Bildschirmausgabe aller Blibs eines Wavelets

Durch die Selektion einer Wavelet wird dem Nutzer das in der Abbildung 4.6 dargestellte Bildschirmlayout angezeigt. Hier kann durch die Blibs „gescrollt“ werden. Das letzte Element ist analog zu den Wavelets eine Schaltfläche zum Anlegen einer neuen Blib(Abbildung 4.6rechts). Auch hier wird ein Bildschirmlayout gleich dem aus Abbildung 4.3 geladen.
Mit der Selektierung einer Blib kann deren Inhalt geändert werden, das heißt Empfängerlist und Nachricht. Hierfür wird ebenfalls ein Bildschirmlayout gleich der Abbildung 4.3 gezeigt.

5 Zusammenfassung und Ausblick

5.1 Zusammenfassung

Das zu Beginn der Arbeit gesetzte Ziel wurde mit Hilfe des GoogleWave-Dummys, als Ersatz für den Internetdienst GoogleWave, erreicht. Mit der Applikation *AndroidWave* wird der Nutzer in die Lage versetzt, an einer beliebigen, verfügbaren Diskussion auf der Ebene des GoogleWave-Dummys, in der elementarsten Form teilzunehmen. Der Funktionsumfang der Applikation erlaubt es ihm, neue Diskussusionen in Form von Waves zu erstellen und bestehende abzurufen. Überdies kann er alle Themen einer Diskussion in Form von Wavelets einer Wave und die dazugehörigen Beiträge in Form von Blibs einsehen. Das Erstellen einer Wavelet oder einer Blib wird ebenfalls durch die Applikation ermöglicht. Jede Blib kann im Hinblick auf ihren Inhalt verändert werden. Das heißt, Einträge in der Empfängerliste können entfernt oder durch neue E-Mail-Adressen ergänzt werden. Die geschriebene Nachricht ist ebenfalls editierbar. Alle Waves, Wavelets und Blibs werden auf der Datenbank des GoogleWave-Dummys persistent gespeichert. Der Zugriff von *AndroidWave* auf den GoogleWave-Dummy wird durch entsprechende Schnittstellen in Form von PHP-Scripten ermöglicht. Für die Implementierung von *AndroidWave* ist ein ausführliches Software Engineering in Form der objektorientierten Analyse und des objektorientierten Designs durchgeführt worden.

5.2 Ausblick

Zum Ende der Bearbeitungszeitraumes dieser Arbeit aktualisierten die Entwickler von GoogleWave die *Wave Data API*. Es werden nun Funktionen angeboten, mit deren Hilfe von externen Clients aus, auf GoogleWave zugegriffen werden kann. Der tatsächliche Datenaustausch zwischen einem externen Client und dem GoogleWave-Server erfolgt laut API über das Datenformat JSON. Die darin übermittelten Informationen sind weitesgehend mit denen des im Rahmen der vorliegenden Arbeit entwickelten GoogleWave-Dummys identisch. Analog zum JSON-Array des GoogleWave-Dummys werden auch vom „echten“ Internetdienst

WaveIDs, WaveletIDs, BlibIDs, Empfängerlisten, der Autor einer Blib und der Inhalt einer Blib übergeben.
Da im Hinblick auf die Softwarearchitektur von *AndroidWave* die Drei-Schichtenarchitektur angewendet wurde, ist das Anpassen der Applikation an die neue Teil-API von GoogleWave der *Wave Data API*, ohne weiteres möglich. Hierfür muss einzig der *Data Access Layer* ausgetauscht werden. Da die gesendeten Daten von GoogleWave weitesgehend mit denen des GoogleWave-Dummys übereinstimmen, reduziert sich die Anpassung der Schnittstellen auf ein Minimum. In späteren Versionen von *AndroidWave* kann so der GoogleWave-Dummy durch den tatsächlichen Internetdienst GoogleWave ersetzt werden.

Kurz nach der Aktualisierung der API wurde die Einstellung der Entwicklung an GoogleWave bekannt gegeben [Onl10]. Als Grund wurde mangelndes Interesse an GoogleWave, verursacht durch einen zu komplexen Funktionsumfang, angegeben.
Trotzdem werden Softwareprodukte zur Unterstützung kooperativer Arbeitsorganisation auch weiterhin eine komfortable Möglichkeit für Firmen darstellen, um die Zusammenarbeit ihrer Teams besser organisieren zu können. Neben GoogleWave existieren schließlich weitere Softwarelösungen welche stetig weiterentwickelt werden und einen ähnlichen Funktionsumfang besitzen.
Darüber hinaus ist es vorstellbar, dass es in Zukunft einen GoogleWave-Ableger geben könnte, welcher im Hinblick auf die Kritiken des Vorgängers besser an die Bedürfnisse der Anwender angepasst ist.
Für *AndroidWave* ergeben sich momentan zwei Möglichkeiten. Entweder die Applikation wird wie vorgesehen an GoolgeWave angepasst, was jedoch früher oder später die Einstellung des Softwareprojektes mit sich ziehen würde. Oder *AndroidWave* wird für eine der GoogleWave-Alternativen umgeschrieben. Des weiteren ist es vorstellbar, *AndroidWave* zu einem eigenständigen Softwareprojekt umzustrukturieren. Dazu müsste die Abhängigkeit zu einer bestehenden Software gelöst und der GoogleWave-Dummy und Teile von *AndroidWave* für den kommerziellen Gebrauch angepasst werden. *AndroidWave* könnte zu einer selbstständig laufenden Applikation für ein Smartphone entwickelt werden, um kooperative Arbeitsorganisation zu unterstützen.

A Anhang

A.1 Glossar

API

Als API (Application Programming Interface) werden Programmkomponenten bezeichnet welche von einem Softwaresystem für anderen Programm zur Verfügung gestellt wird. Mit Hilfe der API können systemfremde Programme an das System angebunden werden.

dex-Format

Android wird in der Programmiersprache Java geschrieben. Beim compelieren des Java-Codes entsteht Java-Bytecode, welches die DVM nicht lesen kann. Mit dem dex-Tool wird der Java-Bytecode in das dex-Format umgeschrieben, welches die DVM verwerten kann.

DVM

Die DVM (Dalvik Virtual Machine) ist, neben den Android-Laufzeitbibliotheken, Hauptbestandteil der Laufzeitumgebung des Betriebssystems Android.

EDGE

EDGE (Enhanced Data Rates for GSM Evolution) ist eine Erweiterung des Datendienstes GPRS. Durch zusätzliches des Modulationsverfahren von EDGE wird die Datenübertragungsrate von GPRS erhöht.

GPRS

GPRS (General Packet Radio Service) ist ein Datendienst und wurde nachträglich als Erweiterung des Mobilfunkstandards GSM entwickelt.

GSM

Als GSM (Global System for Mobile Communications) wird der Mobilfunkstandard der zweiten Generation bezeichnet.

JSON

JSON (JavaScript Object Notation) ist ein schlankes Datenformat für den Austausch von Daten zwischen Server und Client.

OOA

OOA (Objektorientierte Analyse) bezeichnet eine Phase der Softwareentwicklung. Während der objektorientierte Analyse werden Anforderungen bezüglich des zu entwickelnden Systems erfasst und beschrieben, welche das System erfüllen soll.

OOD

OOD (Objektorientierte Analyse) bezeichnet eine Phase der Softwareentwicklung. Während der Phase des objektorientierten Designs werden alle Ergebnisse aus der OOA weiterentwickelt. Es entsteht daraus ein Systementwurf der zu entwickelnden Software

SDK

Das SDK (Software Development Kit) umfasst eine Sammlung von Werkzeugen und Anwendungen, um Softwareentwicklern das implementieren von auf der SDK basierenden Produkten zu ermöglichen. **UMTS**

Als UMTS (Universal Mobile Telecommunications System) wird der Mobilfunkstandard der dritten Generation bezeichnet.

WLAN

Als WLAN (Wireless Local Area Network) wird ein lokales Funknetzwerk bezeichnet, welches den IEEE-802-11-Standard verwendet.

Literaturverzeichnis

[And10] AndroMDA. values online. http://galaxy.andromda.org/index.php?option=com_content&task=view&id=108&Itemid=89, 2010.

[Goo10a] GoogleWave. values online. http://wave.google.com/, 2010.

[Goo10b] GoogleWave. values online. http://code.google.com/intl/de-DE/apis/wave/guide.html, 2010.

[Goo10c] GoogleWave. values online. http://code.google.com/apis/wave/wavedata/, 2010.

[Goo10d] GoogleWave. values online. http://code.google.com/intl/de-DE/apis/wave/extensions/robots/, 2010.

[Goo10e] GoogleWave. values online. http://code.google.com/intl/de-DE/apis/wave/extensions/gadgets/guide.html, 2010.

[Goo10f] GoogleWave. values online. http://code.google.com/intl/de-DE/apis/wave/embed/, 2010.

[Onl10] Heise Online. values online. http://www.heise.de/newsticker/meldung/Google-Wave-wird-eingestellt-1050992.html, 2010.

[Ope10] OpenGroupware. values online. http://www.opengroupware.org, 2010.

[Tea10a] Teamr. values online. http://www.teamr.com, 2010.

[Tea10b] TeamSnap. values online. http://www.teamsnap.com, 2010.

[uMP09a] Arno Becker und Marcus Pant. *Android - Grundlagen und Programmierung*, chapter 2. DPunkt Verlag, 2009.

[uMP09b] Arno Becker und Marcus Pant. *Android - Grundlagen und Programmierung*, chapter 2.1. DPunkt Verlag, 2009.

[uMP09c] Arno Becker und Marcus Pant. *Android - Grundlagen und Programmierung*, chapter 2.2. DPunkt Verlag, 2009.

[uMP09d] Arno Becker und Marcus Pant. *Android - Grundlagen und Programmierung*, chapter 6.5.3. DPunkt Verlag, 2009.

[uMP09e] Arno Becker und Marcus Pant. *Android - Grundlagen und Programmierung*, chapter 5.6.4. DPunkt Verlag, 2009.

Abbildungsverzeichnis

Tabellenverzeichnis

Listings