Bachelor-Thesis

Fachhochschule Düsseldorf

vorgelegt am: 12. August 2011

Entwicklung einer OSGi-Service-Komponente zum dynamischen Laden von Benutzeroberflächen für Android im Umfeld von Ambient Assisted Living

Development of an OSGi Service Component for dynamic loading of user interfaces for Android in the field of Ambient Assisted Living

Name:	Thomas Schmitz
Matrikelnummer:	537957
1. Prüfer:	Prof. Dr. Lux
2. Prüfer:	Prof. Dr. Schaarschmidt

Zusammenfassung

In diesem Projekt wird ein Softwaresystem entwickelt, mit dem es möglich ist, verschiedene Geräte über ein Android Smartphone zu überwachen und zu steuern. Um auf die Dynamik der Umgebung zu reagieren, wird als Basis das OSGi Framework verwendet. Im ersten Teil wird auf die Architektur und den Aufbau von Android und des OSGi Framework eingegangen. Anschließend wird die für dieses Projekt entwickelte Architektur betrachtet. Zum Schluss wird die Implementierung der einzelnen Module und deren Zusammenwirken erläutert.

Inhaltsverzeichnis

Kapitel 1

Einleitung

1.1 Ambient Assisted Living

Menschen werden heutzutage immer älter. Das heißt, dass auch die Zahl der Personen, die alleine wohnen, stetig ansteigt. Daraus ergibt sich, dass der Betreuungsaufwand speziell für ältere Personen immer größer wird. Das relativ junge Gebiet von Ambient Assisted Living setzt genau an diesem Punkt an. Ziel dieses Forschungsgebiet ist es, älteren Personen ein möglichst langes, selbständiges Leben in ihrer gewohnten Wohnumgebung zu ermöglichen.
Dies soll durch IT basierende Systeme gelöst werden. Diese Systeme können verschiedenste Ausrichtungen haben, angefangen von einem Erinnerungssystem, bis hin zu einem System zur Notfallerkennung oder Ferndiagnose via Telemedizin. Typische Gebiete in AAL Projekten sind:

- HealthCare, z.B. Blutdruckmessung
- HomeCare, z.B. Fenstersteuerung
- Sicherheit, z.B. Feuermelder
- Privatsphäre, z.B. Datenschutz
- Versorgung und Haushalt, z.B. Erinnerungsdienste
- Soziales Umfeld, z.B. Kommunikationslösungen

Dadurch sind AAL Projekte zumeist interdisziplinär angelegt, um alle Aspekte dieses Themas berücksichtigen zu können. Ein weiterer Punkt bei so einem Projekt ist der psychologische Gesichtspunkt. Das Image, welches ein

Benutzer durch ein solches System bekommt, ist ein wichtiges Kriterium. Aus diesem Grund müssen die Komponenten eines solchen Systems mit bedacht ausgewählt und vermarktet werden. Ziel muss es sein, dass die Geräte sich unauffällig in das Leben der zu betreuenden Person einpassen, sodass weder sie selbst in ihrer Lebensqualität beeinträchtigt wird, noch ein dritter sieht, dass die Person hilfsbedürftig ist.
Dies ist das Arbeitsfeld von Embedded Systems. Hier wird versucht, die eingesetzten Geräte in Form und Größe so anzupassen, dass diese sich möglichst unauffällig in den Alltag der Menschen integrieren lassen. Dies ist oftmals problematisch, da keine Zielgruppe so verschieden ist, wie die Zielgruppe der Menschen im hohem Alter. Je nach dem, wie vital die Person ist, und welchen Aktivitäten sie nachgeht, sind die Anforderungen an die Geräte sehr verschieden.
Das gleiche Problem stellt sich für die Software Entwicklung. So besteht hier das Problem, dass ältere Menschen häufig wenig technische Kenntnisse haben. Dies stellt die Entwickler vor das Problem, dass die Benutzeroberflächen intuitiv bedienbar sein müssen. Durch die größeren Eingabekomponenten passen häufig nur wenige von diesen auf den Bildschirm. Dies kann dadurch verbessert werden, das nur Komponenten, die wirklich benötigt werden, geladen werden. Dies vereinfacht zum einen die Bedienung, da keine unnötigen Buttons oder ähnliches dargestellt werden, zum anderen können Informationen groß genug dargestellt werden. Da viele Anwender mit der Bedienung, so einfach sie auch gestaltet ist, genug Probleme haben, kann man nicht von ihnen verlangen, dass sie neue Komponenten ins System einbinden können. Aus diesem Grund sind auch hier wieder Softwareentwickler gefragt, diesen Vorgang zu automatisieren und so das System so einfach wie möglich zu gestalten.

1.2 WieDAS Projekt

Das WieDAS[1] Projekt beschäftigt sich mit der Entwicklung eines AAL Systems. Dazu wurden im Labor für Informatik an der Fachhochschule Düsseldorf Geräte entwickelt, welche im AAL Kontext benutzt werden sollen. Dazu zählen zum Beispiel:

- Wassermelder
- Türüberwachung
- Thermometer
- Notfallknopf

Zusätzlich sind weitere Geräte in der Entwicklung, um ein möglichst weites Spektrum an Einsatzgebieten abzudecken. All diese Geräte senden ihre Daten drahtlos an ein Auswertungssystem, welches durch eine PC oder Android Smartphone bereitgestellt werden kann. Um mit den, dynamischen Umfeld, in den ein solches System betrieben wird, umzugehen, wird als Basis ein OSGi Framework verwendet, welches in Kapitel 2.2 näher vorgestellt wird. Bis zur Verbindung der Ansätze der beiden Partner, Hochschule RheinMain und Fachhochschule Düsseldorf, besteht die Düsseldorfer Architektur aus sechs Schichten.

Hardware Die Hardwareschicht repräsentiert die mit Mikrocontrollern ausgestatteten Geräte, die als Sensor bzw. Aktor fungieren. Die Geräte kommunizieren mittels 6LoWPAN[2] mit dem System. Als Protokoll wird hier das weit verbreitete IPv6 verwendet. Durch dieses aktuelle Protokoll kann davon ausgegangen werden, dass auch zukünftige Geräte einfach implementiert werden können.

Hardware Connector Der Hardware Connector setzt die Daten der Hardware in für das Software Framework verwendbare Daten um. Des weiteren können über ihn Daten an Geräte gesendet werden. Dazu macht der Connector eine Umsetzung zwischen logischer Adresse im Framework und IP Adresse des Gerätes.

[1] Wiesbaden-Düsseldorfer Ambient Assisted Living Service Plattform

[2] 6LoWPAN ist ein Kommunikationsprotokoll zur Funkdatenübertragung. Der Name ist ein Akronym für „IPv6 over Low power WPAN“

Logische Geräte/Basisdienst Zu jedem realen Gerät existieren im Software Framework ein logisches Gerät, welches die spezifischen Charakteristiken des realen Gerätes darstellt. Dazu bietet es über ein Interface Methoden für das jeweilige Gerät an, die von den überliegenden Schichten genutzt werden können. Zusätzlich bereitet es empfangene Informationen auf und wertet diese entsprechend seiner implementierten Logik aus.

Basisdienste stellen Funktionen bereit, die von verschiedenen Komponenten genutzt werden können. Dadurch muss nicht jedes Modul diese einzeln implementieren. Basisdienste könnten zum Beispiel ein Erinnerungsdienst oder ein Persistenzdienst sein. Diese Dienste sind nicht auf ein spezielles Gerät angewiesen und können unabhängig von der angemeldeten Hardware betrieben werden.

Anwendung In der Anwendungsschicht werden verschieden logische Geräte bzw. Basisdienste zu einer Anwendung zusammengefasst. Ein Beispiel könnte eine Raumluftüberwachung sein, welche auf die logischen Geräte Temperatur, Fenstersteuerung und CO/2 Sensor zugreift und die empfangen Daten auswertet um, so zu prüfen, ob die Luft im Raum ok ist. Für einen Verlauf könnte die Anwendung auf einen Persistenzdienst zurückgreifen und die ermittelten Zustände in einer Datenbank speichern.

View Die View Schicht stellt die Benutzeroberflächen der Anwendung bereit. Dabei können für verschiedene Endgeräte unterschiedliche Oberflächen für eine Anwendung existieren. Dies ist nötig, da nicht alle Anzeigegeräte die gleichen Voraussetzungen mitbringen. So muss bei einem mobilen Endgerät die Oberfläche auf kleine Displays angepasst werden, während auf einem Desktop PC oder Fernseher mehr Informationen auf einmal dargestellt werden können.

Darstellung Die Darstellung ist die oberste Schicht. Sie dient lediglich zum Darstellen der Views der unteren Schicht. Für die Darstellung können verschiedene Plattformen genutzt werden. Für Desktop-Anwendungen ist dies Eclipse Rich Client, für mobile Anwendungen soll Android eingesetzt werden. Innerhalb der Darstellungsschicht befindet sich lediglich die Logik für das Aktualisieren der Oberfläche.

Für die Kommunikation zwischen den Schichten wird ein Cache benutzt, welche Veränderungen an Daten über ein Event meldet. Die Methodenaufrufe erfolgen über Interfaces, die von den einzelnen Schichten bereitgestellt werden. Abb. 1.1 zeigt den Aufbau der Architektur.

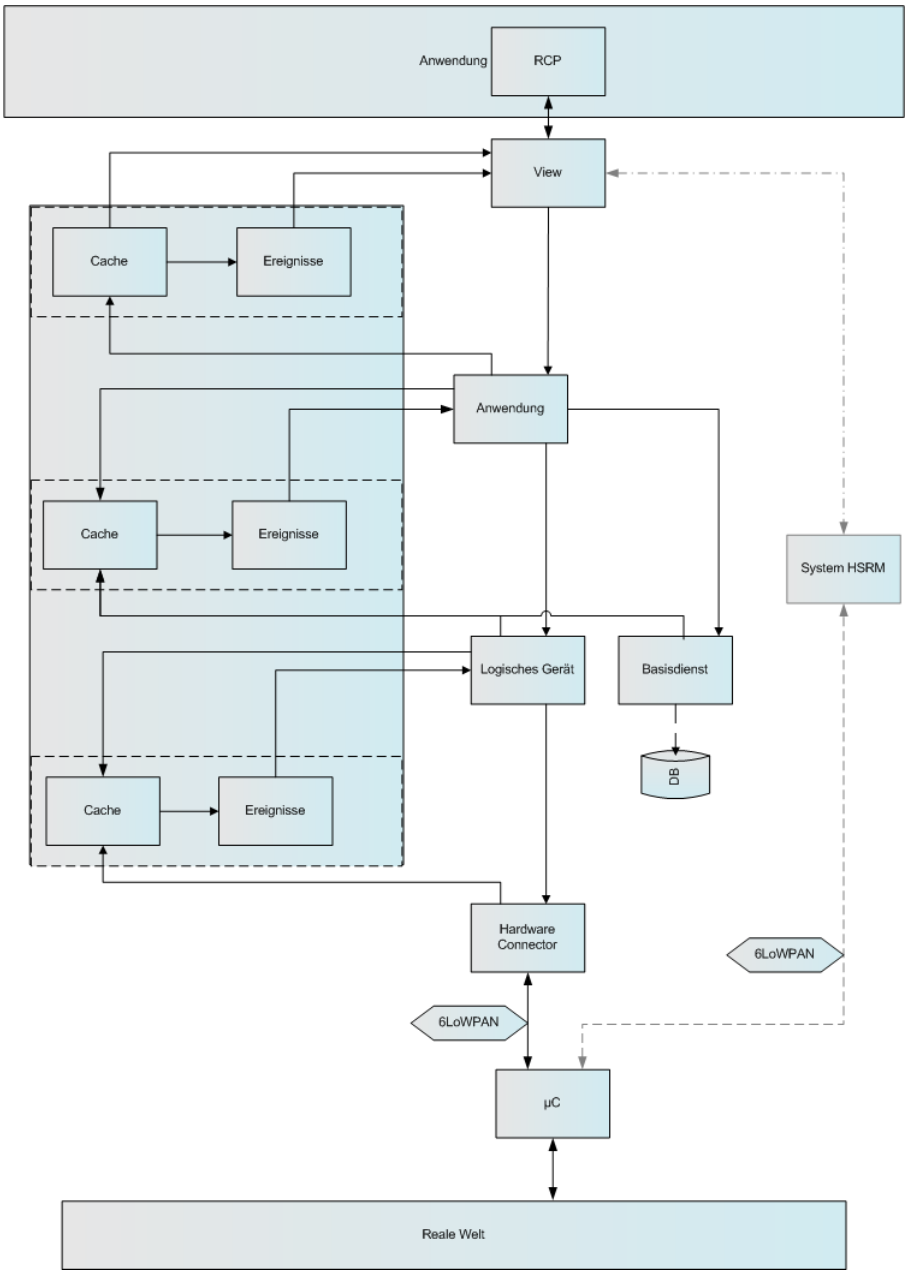

Abbildung 1.1: WieDAS-Architekturkonzept Düsseldorf [1]

1.3 Aufgabenstellung

Innerhalb des vorhergegangenen Praxisprojekts [2] wurde eine Anwendung entwickelt, welche es, erlaubt OSGi Komponenten auf einem Android Endgerät zu verwenden. Dazu wurden das OSGi Framwork Felix in eine Android Anwendung eingebettet. Während des Praxisprojektes wurde dann ein Türüberwachungs-Modul implementiert.
Jedoch ist es bisher nur beschränkt möglich, zur Laufzeit eine andere Oberfläche zu laden. Das OSGi Framework macht dies zwar grundsätzlich möglich, nur fehlt ein entsprechender Service, der dies auch dem Endanwender ermöglicht. Des weiteren müssen neue Geräte zur Zeit manuell mittels Konsole installiert werden, was sehr aufwendig ist und einem Endanwender nicht zugemutet werden kann.
Dieses beiden Probleme sollen in dieser Bachelor Thesis behandelt werden. Dazu wurden folgende Überlegungen angestellt.

Grundlegende Struktur Ausgehend von vorhergegangenen Praxisprojekt kann die daraus resultierende Anwendung grundlegend weiter verwenden. Zusätzlich zu dieser Anwendung muss ein Repository entstehen aus dem neue Oberflächen geladen werden können. Die Verbindung zu diesem Repository soll über eine HTTP Verbindung erfolgen, dies vereinfacht die Installation neuer Bundles und ist bereits in Android implementiert.

Oberflächenwechsel Für den Oberflächenwechsel muss ein Menü entwickelt werden, welches dynamisch zur Laufzeit wachsen kann. Das Menü muss stets alle im Framework installierten Geräte anzeigen und eine Möglichkeit besitzen nicht mehr benutzte Geräte zu entfernen. Um das Menü möglichst übersichtlich zu gestalten, werden verschiedene Layout Ansätze evaluiert.

Automatische Installation Um dem Endanwender zu ermöglichen neue Geräte ins System einzubringen, ist es erforderlich, dass sie sich möglichst automatisch ins Framework einbinden. Dazu wird eine Art Gerätemanager benötigt, der bei einer eingehenden Nachricht prüft, ob es sich um ein neues Gerät handelt und bei Bedarf die dazugehörigen Bundles installiert. Die Information, welche Bundles zu einem Gerät gehören, sollen in einer Datenbank im Reposetory festgehalten werden.

Ausgehend von diesen Überlegungen wird ein Konzept erstellt, welches die grundlegenden Module für ein solches System enthält. Dieses soll sich in das Düsseldorfer Gesamtmodell integrieren lassen und so für nachfolgende Projekte eine Basis bieten, in denen weitere Geräte integriert werden können. Dazu soll auch dieses Projekt auf der Grundlage der in Abb. 1.1 gezeigten Architektur aufbauen. Als Referenz kann dazu das in Eclipse RCP entwickelte Framework von Andreas Karp dienen, welches eine ähnliche Funktionalität für Desktop Systeme anbietet [3].

In den folgenden Kapitel werden zuerst die theoretischen Grundlagen von Android und OSGi vermittelt. Innerhalb der Implementierung wird zuerst das Konzept des entwickelten System vorgestellt und anschließend die praktische Umsetzung erläutert. Abschließend werden in einem Fazit die gewonnen Erfahrungen und etwaige Probleme beschrieben.

Kapitel 2

Grundlagen

2.1 Android OS

In diesem Kapitel soll auf die Grundlagen des Android Betriebssystem eingegangen werden. Als erstes wird dazu die Open Handset Alliance beschrieben. Darauf folgt ein Überblick über die Architektur des Betriebssystems. Abschließend wird der Aufbau einer Android Anwendung erläutert.

2.1.1 Open Handset Alliance

Google gründete, im Zusammenschluss mit 34 weiteren Firmen, im November 2007 Die Open Handset Aliance (OHA). Im Dezember 2008 wurde 14 weitere Firmen in das Konsortium aufgenommen. Das neuste Mitglied der OHA ist die Firma NXP Software, welche im Mai 2008 dem Konsortium beigetreten ist [4]. Damit besteht das Konsortium jetzt aus mehr als 65 Firmen.
Gemeinsam entwickeln diese Firmen das Betriebssystem Android, sowie Hardwarekonzepte und Richtlinien für Implementierer. Die 34 Gründungsmitglieder stellen sich folgendermaßen zusammen [5]:

- 7 Netzbetreiber
- 9 Halbleiter Firmen
- 4 Endgeräte Hersteller
- 10 Software Firmen
- 4 Vermarktungs-Firmen

Netzbetreiber Nahezu alle deutschen Netzbetreiber sind in der OHA vertreten. Lediglich E-Plus gehört nicht zum OHA-Konsortium. Dadurch wird in Deutschland ein großer Kundenkreis für Android Endgeräte erschlossen. Netzbetreiber haben dabei die Möglichkeit die Endgeräte günstig ihren Kunden zur Verfügung zu stellen, wodurch diese speziell für ältere Menschen erschwinglich werden.
Halbleiter Firmen Eine wichtige Komponente zum Erfolg einer Plattform ist die verwendete Hardware. Die OHA hat dafür einige große Halbleiter Firmen, auf dessen Wissen zurückgegriffen werden kann. Dies sind unter anderem [6]:

- ARM - Technologieunternehmen mit Spezalisierung auf RISC-CPUs
- Intel - führender Prozessorhersteller
- Texas Instruments - Halbleiterhersteller u.A. Mobilfunkbereich
- Nvidia - Entwicklung von Grafik-Hardware

Endgeräte Hersteller Mit Samsung, LG, Motorola und Sony-Ericsson sind vier der größten Endgeräte Hersteller Mitglied in der OHA. Lediglich Nokia ist als relevanter Entgerätehersteller nicht Mitglied, da diese, auch nach dem ausscheiden von Sony-Ericsson und Samsung aus der Symbian-Foundation, bei ihren Endgeräten weiter auf Symbian setzen. Neben den Herstellern von Mobilfunk Endgeräten sind auch Partner aus anderen Gebieten in der OHA vertreten. Als große Vertreter aus der Computertechnik sind dort zum Beispiel Toshiba und ASUSTek zu nennen [6].
Software Firmen Im Bereich der Softwarefirmen stechen vor allem zwei Unternehmen heraus. Dies ist zum einen Google, welches auch als treibende Kraft im Konsortium tätig ist. Zum anderen ist es Ebay, welches auch auf ein breites Wissen in Internettechnologien zurückgreifen kann. Des weiteren sind auch mehrere kleinere Software Unternehmen im Konsortium vertreten, welche innerhalb ihrer Spezialgebiete Wissen und Technologie zum Projekt beitragen.
Vermarktungs-Firmen Die Vermarktungs-Firmen arbeiten als Bindeglied zwischen den einzelnen Partnern. Sie arbeiten mit den Soft- und Hardware-Herstellern zusammen, um die entwickelte Software in eine Hardwareumgebung zu integrieren, aus welcher dann ein Produkt entwickelt werden kann.

2.1.2 Architektur

Im folgenden Kapitel soll auf die Android Architektur eingegangen werden, auf welche jede Android Version aufbaut. Wie in Abb. 2.1 zu sehen kann diese in fünf Komponenten aufgeteilt werden. Diese werden nun genauer betrachtet.

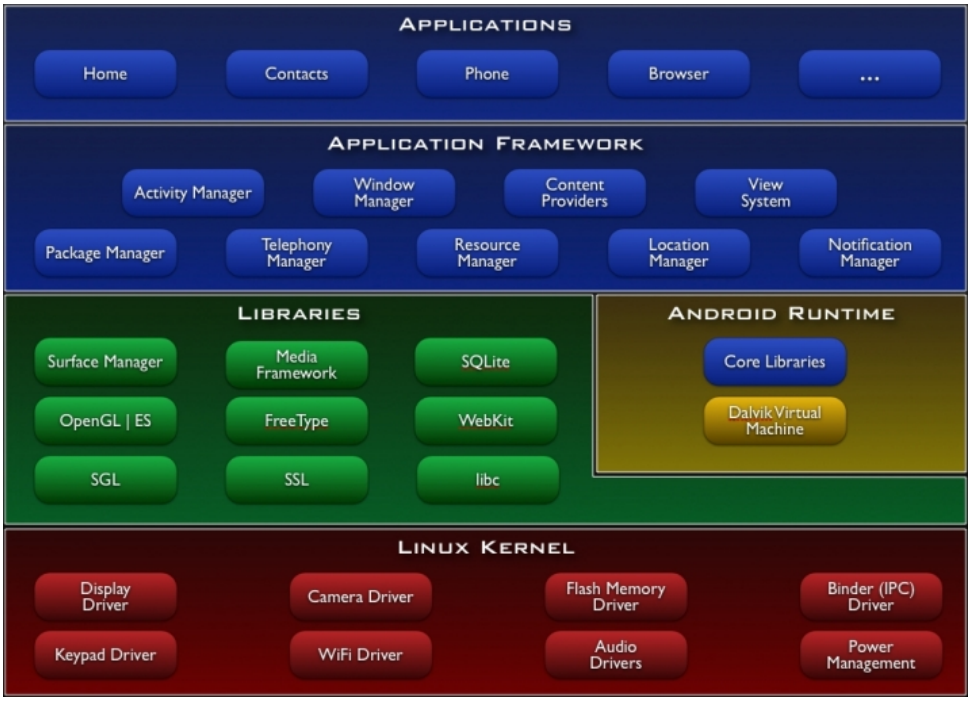

Abbildung 2.1: Android Architektur [7]

2.1.2.1 Linux Kernel

Android baut auf einen Linux Kernel der Version 2.6 auf. Dieser dient als Abstraktionsschicht über den die überliegenden Komponenten auf die Hardware zugreifen können. Der Linux Kernel übernimmt auch die Aufgaben der Energie und Speicherverwaltung, wofür dieser speziell optimiert wurde. Im Kernel ist ebenfalls das Sicherheitsmodell implementiert, welches die Rechte für die Interprozesskommunikation enthält, sowie auch für das Dateisystem.

2.1.2.2 Libraries

Die Funktionalitäten des Android Betriebssystem sind in C/C++ Bibliotheken hinterlegt. Über Java Schnitstellen können diese in Anwendungen eingebunden werden und dort verwendet werden. In den Libraries findet man zum einen bekannte Open-Source-Produkte wie:

- SQLite - SQL Datenbank für mobile Plattformen
- WebKit - Webbrowser Engine
- FreeType - Konvertierung von Vektor in Rastergrafik und umgekehrt
- SSL - häufig angewendetes Verschlüsselungsprotokoll
- OpenGL/ES - 3D-Grafik Schnittstelle für Embedded Systeme

Zusätzlich wurden auch speziell für das Android System Bibliotheken Entwickelt wie:

- Surface Manager - Zugriffssteuerung des Anzeigesystem
- SGL - Rendering Bibliotheken für Vektorgrafiken
- Media Framework - Codecs für Multimedia Inhalte

2.1.2.3 Android Runtime

Innerhalb der Android Rumtime werden die Anwendungen ausgeführt. Dabei wird für jede Anwendung ein eigener Prozess, sowie eine eigene Virtuelle Maschine angelegt. Der Ressourcenverbrauch ist dabei zwar höher als bei der nativen Ausführung der Anwendungen, es hat aber den Vorteil, dass ein fehlerhafter Prozess nur seine eigens für ihn erstellte Virtuelle Maschine zum Absturz bringt. Alle anderen Anwendungen sind durch einen fehlerhaften Prozess nicht in ihrer Ausführung beeinträchtigt.
Die Android Runtime besteht aus 2 Modulen:

- Core Libraries
- Dalvik Virtual Machine

Core Libraries
In den Core Libraries befinden sich die auf Android portierten Pakete aus der Java Standart Edition. Es wurden nicht alle Pakete auf Android portiert, da sie zum Teil dort nicht benötigt werden wie z.B. javax.print, oder Android dafür eigene Implementierungen besitzt wie z.B. APIs für die grafischen Benutzeroberflächen. Die folgende Liste zeigt, welche Pakete auf Android portiert wurden [5]:

- Sprach und Unterstützungsbibliotheken
  - java.lang - stellt Elementare Grundtypen von Java bereit
  - java.util - Collections, Event-Handling, Datum
- Basisbibliotheken
  - java.io - Eingabe und Ausgabe von Dateien/Streams
  - java.math - mathematische Funktionen und Konstanten
  - java.net - Basis Netzwerkfunktionen
  - java.nio - Erweiterung der I/O Paket
- Integrationsbibliotheken
  - java.sql - Datenbank Schnitstelle
  - javax.sql - Erweiterungen des SQL Paket
- Sicherheitsbibliotheken
  - javax.crypto - stellt Verschlüsselungstechnologien bereit
  - java.security - (veraltet) Java Sicherheits Framework
  - javax.security - (aktuelle) Java Sicherheits Framework
- weitere Bibliotheken
  - javax.net - erweiterte Netzwerkfunktionalität
  - javax.sound - Paket für die Audioverarbeitung
  - javax.xml - XML Verarbeitung
  - org.w3c.dom - Schnittstelle für Document Object Model
  - org.xml.sax - SAX-Parser

Dalvik Virtual Machine

Die Dalvik Virtual Machine(DVM) ist der Kern der Android Runtime. In ihr werden die entwickelten Anwendungen ausgeführt. Die DVM wurde von dem Google Mitarbeiter Dan Bornstein entwickelt und basiert auf der quelloffenen Java VM *Apache Harmony* [8]. Die DVM wurde auf die Bedürfnisse von mobilen Plattformen angepasst. Um Rechenzeit und dadurch Energie zu sparen wurde die DVM von vornherein an auf ARM Prozessoren angepasst. Die Standard Java VM nutzt in ihrer virtuellen Prozessorarchitektur die Vorteile moderner Mikroprozessoren nicht aus. Moderne Prozessoren arbeiten mit Registerspeichern, die direkt vom CPU genutzt werden können. Innerhalb der DVM wurde die virtuelle Prozessorarchitektur so umgebaut, dass diese nun diese Register nutzen kann. Zu Beginn wurde die DVM nur auf ARM Prozessoren eingesetzt. Durch die quelloffene Entwicklung, haben weitere Hersteller die DVM auf ihre Systeme portiert, so dass die DVM auf vielen modernen Systemen lauffähig ist.

Durch die neue virtuelle Prozessorarchitektur und aus lizenzrechtlichen Gründen ist es nicht möglich, Java Bytecode in der DVM auszuführen. Um weiterhin die Vorteile der Java Programmiersprache und die dafür zur Verfügung stehenden Hilfsmittel zu nutzen, greift man auf Cross-Compiling zurück. Dabei wird mit Hilfe der *Android Developer Tools*, welche im vorhergehenden Praxisprojekt [2] beschrieben wurden, aus dem Java Bytecode Android Bytecode generiert, der dann auf dem Endgerät ausgeführt werden kann. Abb. 2.2 zeigt den oben beschriebenen Ablauf.

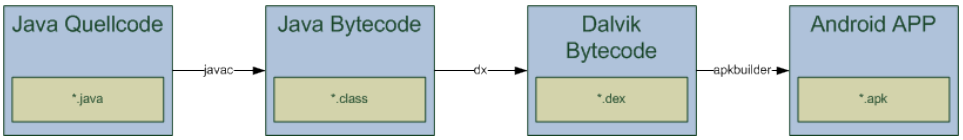

Abbildung 2.2: Android Cross-Compiling

2.1.2.4 Application Framework

Das Application Framework bietet über seine Managerklassen Zugriff auf die Schnittstellen des Android System. Das Application Framework ist vollständig in Java geschrieben und stellt der über ihr liegenden Anwendungsschicht über Java-Interfaces seine Funktionen zur Verfügung. Jede Android Anwendung nutzt diese Schnittstellen. Dadurch ist es möglich, selbst die von Android mitgelieferten Anwendungen gegen eine eigene Implementierung zu tauschen. Dies könnte hilfreich sein, um das Gerät an eine spezielle Zielgruppe anzupassen. Im WieDAS Projekt wäre eine Anpassung der Oberflächen für ältere Menschen sinnvoll, die auf Kosten von Funktionalität eine einfachere Bedienung ermöglichen würde. Dies ist jedoch nicht Teil dieses Projektes.

2.1.2.5 Applications

In der obersten Schicht sind die Android Anwendungen angesiedelt. Hier sind alle vorinstallierten oder selbst erstellten Anwendungen zu finden. Jede Android Version bringt je nach Hardware Ausstattung verschiedene Basisanwendungen mit. Diese sind zum Beispiel:

- Browser - Internet Browser
- Home - Verwaltung des Startbildschirms
- Kontakte - Adressverwaltung
- Mediaplayer - Abspielen von Multimedia Dateien
- u.v.m

Alle Anwendungen können einzelne Komponenten oder Daten anderen Anwendungen zur Verfügung stellen. Dies erhöht die Wiederverwertbarkeit von Programmen und erleichtert die Interprozesskommunikation. Ein Beispiel ist, dass in der Anwendung *Kontakte* Namen und Telefonnummern hinterlegt sind. Die Anwendung kann diese Daten nun anderen Anwendungen zur Verfügung stellen. Dadurch ist es der Anwendung *Telefon*, möglich mit den Daten aus *Kontakte* ein Telefonanruf zu starten.

2.1.3 Endgeräte

Wie im Kapitel 2.1.1 beschrieben gehören einige große Endgerätehersteller zur OHA. Dadurch sind in Laufe der Zeit viele Endgeräte entwickelt worden, die für die verschiedensten Zwecke verwendet werden. Auch wenn Android lediglich die Software der Endgeräte darstellt, werden im allgemeinen oft die Endgeräte damit verbunden.
Das erste Endgerät auf dem Android betrieben wurde war das G1 vom Hersteller HTC, welches es exclusiv nur bei T-Mobile gab.

Abbildung 2.3: HTC G1 [9]

Das G1 erschien im Oktober 2008 und hatte einen 528MHz ARM11 Prozessor für das Betriebssystem, sowie einen 384MHz ARM9 für das UMTS/GSM Modem [10]. Für die Ausführung von Programmen standen 192MB RAM zur Verfügung. Der Speicher des HTC G1 konnte mit MicroSD Karten um bis zu 32GB Erweitert werden. Als Display kam ein 3,2"Touchscreen mit 480x320 Pixel zum Einsatz.

Heute kann auf eine große Zahl unterschiedlichster Endgeräte mit Android zurückgegriffen werden. Dabei sind längst nicht mehr nur Mobiltelefone die mit Android ausgestattet sind, sondern auch Tablet PCs und SetTop Boxen. Für Softwareentwickler gibt es beim Programmieren für mobile Endgeräte zusätzliche Punkte zu beachten.

***Speicher*:** Mobile Endgeräte besitzen aus Platz- Gewichts- und Energiegründen nur verhältnismäßig wenig Speicher. Dies muss bei der Entwicklung von Software beachtet werden.

***Eingabemethoden*:** Durch ihre zum Teil geringe Größe müssen Bedienelemente bei mobilen Anwendungen geringer ausfallen als bei Desktop Anwendungen. Für das WieDAS Projekt gilt es in diesem Punkt einen guten Kompromiss zwischen Bedienbarkeit und Funktionalität zu erhalten.

***Energie*:** Rechenzeit bedeutet Energieverbrauch. Deshalb sollte darauf geachtet werden möglichst, leichtgewichtige Anwendungen zu entwickeln. Anwendungen, die große Berechnungen, durchführen belasten den Akku und verkürzen dadurch die Zeit in der das Gerät netzunabhängig betrieben werden kann.

***Netzwerk*:** Da Android Endgeräten größtenteils mobil betrieben werden, muss damit gerechnet werden, dass keine konstante Netzwerkverbindung vorhanden ist. Aus diesem Grund müssen Anwendungen so konzipiert werden, dass diese auch ohne Netzwerkverbindung arbeiten können. Jedoch muss in Kauf genommen werden, dass bei fehlender Netzwerkanbindung einige Funktionen nicht bereitgestellt werden können.

2.1.4 Anwendungsaufbau

Um einen grundlegenden Einblick in den Aufbau einer Android Anwendung zu bekommen, wird in diesem Kapitel anhand einer simplen Anwendung der Aufbau erklärt. Dazu wird eine *HelloWorld*[1] Anwendung erstellt, anhand der die einzelnen Komponenten erläutert werden. Die Abb 2.4 zeigt die Projektstruktur einer Android Anwendung.

[1] Ein HelloWorld Programm ist ein einfaches Programm, welches einen Enblick in die Syntax und den Aufbau eines Programmes gibt. Ziel eines HelloWorld Programms ist die Ausgabe eines Textes auf der Oberfläche.

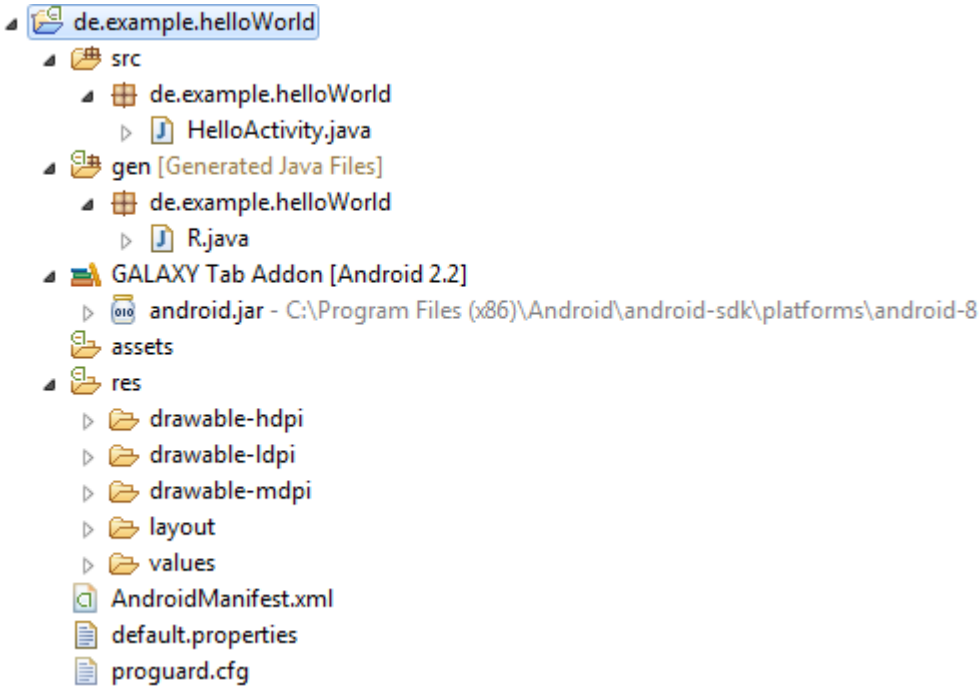

Abbildung 2.4: Projektstruktur HelloWorld Programm

Diese Struktur wird durch Eclipse und das ADT beim Anlegen eines neuen Projektes erzeugt. Nun soll auf die einzelnen Komponenten genauer eingegangen werden.

Activity Komponente

Eine Activity Komponente stellt ein Display bereit, über den ein Benutzer mit der Anwendung interagieren kann. Um eine Activity zu erzeugen, wird eine neue Klasse erstellt, welche von der Klasse Acitvity erbt. In diesem Beispiel ist dies die Klasse HelloActivity.java. Diese wurde beim Anlegen des Projektes bereits durch Eclipse erzeugt und hat den folgenden Inhalt:

```
1 import android.app.Activity;
2 import android.os.Bundle;
3
4 public class HelloActivity extends Activity {
5     /** Called when the activity is first created. */
6     @Override
```

```
7       public void onCreate(Bundle savedInstanceState) {
8           super.onCreate(savedInstanceState);
9           setContentView(R.layout.main);
10      }
11  }
```

Listing 2.1: HelloActivity

Zunächst wird in der onCreate() Methode ein leeres Display erzeugt. Erst beim Aufruf der setContetView() Methode wird eine View Komponente ins Display geladen. Die onCreate() Methode ist eine Callback Methode welche aufgerufen wird, sobald die Anwendung das erste mal gestartet wird. Eine Activity kann vier Zustände annehmen. Je nach Zustand liegt die Activity auf verschiedenen Ebenen auf einen Stack.

aktiv Die Activity, welche ganz oben im Stack liegt ist im Zustand aktiv. Dem Benutzer ist es möglich, mit der Activity zu interagieren. Eine aktive Aktivity wird nur in Notfällen vom System beendet. Im Regelfall wird die Activity beim Start einer neuen Anwendung in den Zustand gestoppt oder pausiert übergehen.

pausiert Wird eine neue Activity gestartet, welche die alte Activity nicht ganz verdeckt, so geht die alte Activity in den Zustand pausiert. Im Stack liegen diese unter den aktiven, jedoch über den gestoppten Activitys. Das System wird diese Activity nur dann beenden, wenn keine Ressourcen durch beenden von gestoppten Activitys gewonnen werden kann.

gestoppt Eine Activity wird gestoppt, sobald sie für den Benutzer nicht mehr sichtbar ist. Die Activity bleibt jedoch solange im Speicher bis dieser für andere Anwendungen benötigt wird. Es werden zuerst die Activitys beendet, welche am längsten im Zustand gestoppt sind.

zerstört Ist eine Activity am unteren Ende des Stacks angekommen und wird aus dem Stack gelöscht, um Speicher frei zu geben, geht sie in den Zustand zerstört und verbraucht keine Ressourcen mehr.

Weitere Informationen zum Lebenszyklus einer Anwendung können dem vorhergehenden Praxisprojekt entnommen werden [2]

View Komponenten

Alle Komponenten, die zur Erzeugung einer Benutzeroberfläche genutzt werden können, erben von der Klasse View. Dadurch ist es möglich, jede Komponente in einer Activity ins Display zu laden. Innerhalb einer Android Anwendung werden diese Komponenten in XML Dateien beschrieben. Es ist jedoch auch möglich, die View Komponenten über Java Quelltext zu erzeugen.
Android nutzt anders als Desktop-Systeme nicht die Java typischen GUI APIs wie AWT, SWT oder Swing, sondern bringt seine eigene GUI API mit. Die Standard Java APIs sind für eine Bedienung mit Eingabegeräten wie Maus und Tastatur optimiert. Diese sind auf mobilen Endgeräten nicht verfügbar. Durch eine eigne API für die grafischen Benutzeroberflächen, kann diese besser an die Bedürfnisse eines mobilen Endgerätes angepasst werden.
Eine Spezialform der View Klasse ist die ViewGroup. In ihr können mehre View Komponenten zusammengefasst werden. Die ViewGroup dient dabei als Container, der die Komponenten aufnimmt. Für die Ausrichtung der Komponenten kann aus verschiedenen Layouts gewählt werden. Dazu zählen zum Beispiel:

- Absolut Layout
- Linear Layout
- Frame Layout
- Relativ Layout

Der Aufbau einer Oberfläche kann mit Hilfe eines Baumes beschrieben werden. Abb. 2.5 zeigt dies exemplarisch. Dadurch ist es möglich, schnell einen Überblick über die verwendeten Komponenten zu bekommen. Für komplexere Oberflächen ist dies zudem auch eine Möglichkeit die Entwicklung auf mehrere Programmierer aufzuteilen.

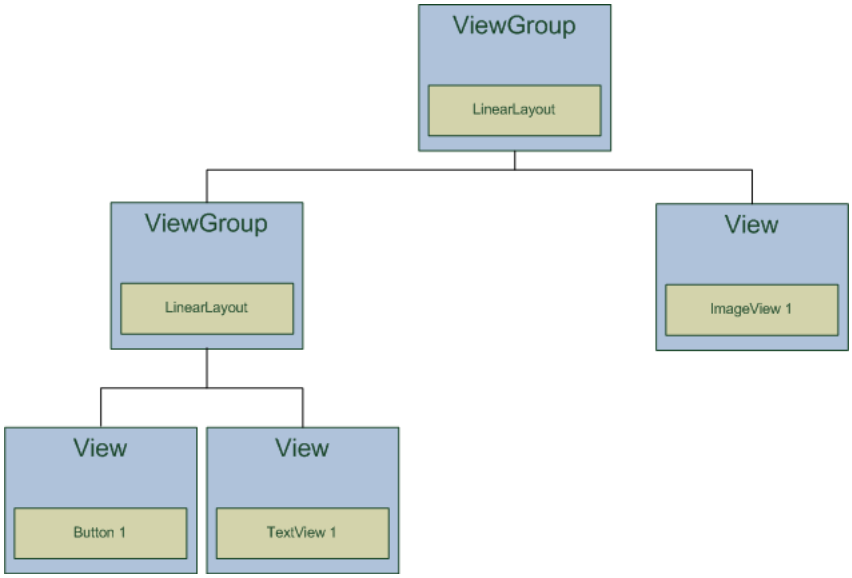

Abbildung 2.5: View Baum

AndroidManifest

Jede Android Anwendung beinhaltet eine Manifest Datei. Diese Datei wird automatisch bei der Projekt Erstellung von Eclipse generiert. Der Name der Datei ist AndroidManifest.xml. In der Manifest Datei werden Metadaten, Komponenten und Zugriffsrechte definiert. Listing 2.1.1 zeigt den Inhalt der AndroidManifest.xml der HelloWorld Anwendung.

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <manifest xmlns:android="http://schemas.android.com/apk/res/
       android"
3        package="de.example.helloWorld"
4        android:versionCode="1"
5        android:versionName="1.0">
6      <application android:icon="@drawable/icon" android:label="
           @string/app_name">
7          <activity android:name=".HelloActivity"
8                    android:label="@string/app_name">
9              <intent-filter>
10                 <action android:name="android.intent.action.
                       MAIN" />
```

```
11                  <category android:name="android.intent.
                        category.LAUNCHER" />
12              </intent-filter>
13          </activity>
14      </application>
15  </manifest>
```

Listing 2.2: AndroidManifest.xml

Im ersten Teil der Manifest Datei werden das Package der Android Anwendung sowie dessen Version beschrieben.
Im Tag Application, welches nur einmal in der Manifest Datei vorkommen darf , werden das Programm Icon sowie der Name der Anwendung festgelegt. Innerhalb des Application Tags werden alle Komponenten der Anwendung aufgeführt. Im Fall der HelloWorld Anwendung ist dies lediglich die Activity in der HelloActivity Klasse. Jede Komponente, die innerhalb der Anwendung verwendet wird, muss in der Manifestdatei eingetragen sein. Das Aufrufen einer Komponente, die nicht in der Manifest Datei deklariert ist, wird mit einer Exception abgebrochen. Im Tag Intent-Filter wird festgelegt, von welcher Komponente die Activity aufgerufen wird. In diesem Fall ist dies der Programm Einstiegspunkt MAIN und der Programm Starter LAUNCHER. Im Sicherheitssystem von Android schreibt vor, dass jede Anwendung für sicherheitskritische Ressourcen des Gerätes wie zum Beispiel GPS, Kontakte oder Telefonie Rechte anmelden muss. Der Benutzer muss dann bei der Instalation der Software zustimmen, ob die Anwendung diese Rechte bekommt. Die benötigten Rechte werden ebenfalls in der Manifest Datei festgelegt und könnten folgendermaßen aussehen:

```
1 <uses-permission android:name="android.permission.INTERNET">
2 </uses-permission>
3 <uses-permission android:name="android.permission.VIBRATE">
4 </uses-permission>
```

Listing 2.3: Android Permissions

In diesem Beispiel würde die Anwendung Zugriff auf Netzwerk und Internet erhalten und den Vibrationsalarm benutzen dürfen. Es gibt noch einige weitere Tags, die in dieser Dokumentation nicht erläutert werden. Für eine vollständige Liste ist ein Blick in das Android Dev Guide [11] zu empfehlen.

Ressourcen

Im res Ordner des HelloWorld Programm befinden sich die Ressourcen, wie Grafiken, Sounds, Farbdefinitionen und ähnliches. In Android Anwendungen sollte möglichst kein direkter Bezug zu den Ressourcen im Quelltext aufgebaut werden. Zu diesem Zweck legt Eclipse für jede Ressource eine ID an, welche über die automatisch generierte R.Java Klasse abgerufen werden kann. Ein Zugriff auf diese Datei ist jedoch nur aus der Android Anwendung sinnvoll möglich. Für eine dynamische Anwendung, in der Ressourcen zur Laufzeit hinzugefügt und entfernt werden können, ist der Ressourcenmanager nicht ausgelegt. Dennoch soll hier ein grundlegender Einblick über die Verwendung von dieses Ressourcen gegeben werden.
Wie oben beschrieben werden den Ressourcen IDs zugeteilt über welche diese abgefragt werden können. Die Verbindung zwischen Ressource und ID ist in der R.java gespeichert. Nachfolgend ist die R.java der HelloWorld dargestellt.

```
1  public final class R {
2      public static final class attr {
3      }
4      public static final class drawable {
5          public static final int icon=0x7f020000;
6      }
7      public static final class layout {
8          public static final int main=0x7f030000;
9      }
10     public static final class string {
11         public static final int app_name=0x7f040001;
12         public static final int hello=0x7f040000;
13     }
14 }
```

Listing 2.4: R.java

Wie im oberen Listing zu sehen wird zu jeder Ressource eine ID angelegt. Auch Basis Typen wie Strings können als Ressource gelistet werden, dies hat den Vorteil, dass für mehrsprachige Softwareversionen keine Veränderungen in Quelltext notwendig sind. Dies wird dadurch erzielt, dass zum Beispiel bei Strings die verschiedenen Übersetzungen alle die selbe ID besitzen und

je nach Ländereinstellung des Endgerätes die entsprechende Lokalisierung geladen wird. Dadurch können Ressourcen lose an die Software gebunden werden. Weitere Informationen zur Nutzung von Ressourcen sind im vorausgegangenen Praxisprojekt [2] und im Android Dev Guide [11] zu finden.

Weitere Komponenten

Neben den Activitys, welche als Hauptkomponente für Benutzerschnittellen dienen, gibt es in Android noch weitere Komponenten. Diese sollen hier kurz betrachtet werden.

Service Ein Service ist eine im Hintergrund laufende Komponente ohne Benutzerschnittstelle. Ein Service kann dazu dienen, im Hintergrund Daten aufzunehmen und auszuwerten, auch wenn die Anwendung nicht aktiv ist.

ContentProvider Ein ContentProvider regelt, welche Daten von einer Anwendung veröffentlicht werden. Im Regelfall ist es Anwendungen nicht gestattet, auf Daten anderer Anwendungen zuzugreifen. Mit Hilfe von ContentProvidern können Daten aus Anwendungen gezielt Exportiert werden damit sie von anderen Anwendungen genutzt werden können. Android bringt dafür einige CotentProvider bereits mit, um zum Beispiel auf Kontaktdaten oder Multimediadaten der Android Basisanwendungen zuzugreifen.

Intent Über einen Intent kann man Nachrichten an das ganze System senden. Die Nachrichten können Daten oder Anweisungen enthalten. Dies bietet eine Möglichkeit zur Interprozesskommunikation.

BroadcastReciver Über einen BroadcastReciver kann eine Anwendung Intents empfangen. Für eine genauere Einteilung ist es möglich, Filter einzusetzen.

Notification Mittels Notificationen können Anwendungen Benutzer über Meldungen informieren. Dies wird überwiegend von BroadcastRecivern und Services verwendet. Um eine Notification zu erzeugen, muss eine Anwendung nicht aktiv sein.

2.2 OSGi Framework

In diesem Kapitel soll auf die Grundlagen des OSGi Framework eingegangen werden. Zunächst wird erläutert wozu das OSGi Framework verwendet wird und welche Funktionen es bietet. Darauf folgt ein Überblick über die Architektur des Frameworks. Abschließend wird der Aufbau eines OSGi Bundles beschrieben.

2.2.1 Definition

Für die Entwicklung modularer Software ist ein Framework notwendig, welches diese Möglichkeit bereitstellt. Android und Java verfügen von Haus aus nur über eine sehr rudimentäre Modularisierung, die es nicht erlaubt, während der Laufzeit einer Anwendung Komponenten hinzuzufügen und zu entfernen. Für Java gibt es in diesen Zusammenhang mittlerweile eine Lösung. Das OSGi Framework erlaubt es Anwendungen in Module sogenannte Bundles aufzuteilen, die dynamisch zur Laufzeit geladen werden können.
OSGi stellt dabei nicht eine Implementierung dar, sondern ist lediglich eine Spezifikation die von der OSGi Alliance herausgegeben wird. Die Umsetzung dieser Spezifikationen, wird von verschiedenen Entwicklergruppen, zum Teil frei, zum Teil kommerziell, übernommen. Die in diesem Zusammenhang bekanntesten Umsetzungen sind zum einen Eclipse Equinox, auf dessen Basis die Entwicklungsumgebung Eclipse aufbaut, zum anderen das Apache Felix Framework.
Das Apache Felix Framework läuft seit der Version 1.0.3 ebenfalls auf Android [12]. Dadurch ist es nun auch auf Android möglich, Anwendungen besser zu modularisieren.
Zusätzlich zur Modularisierung bietet das OSGi Framework auch eine serviceorientierte Middleware an, mit der es möglich ist, das einzelne Bundles Dienste über eine zentrale Service Registry bereitstellen und abrufen können. Durch diese Methode kann eine lose Kopplung zwischen den einzelnen Bundles erreicht werden. Zur Laufzeit können die Bundles Services abrufen und registrieren, dadurch ist man in der Lage, dynamisch auf Veränderungen zu reagieren und neue Funktionen bereitzustellen.
Im folgenden Kapitel soll die Architektur des OSGi Frameworks betrachtet werden, um einen Einblick über dessen grundlegenden Aufbau zu bekommen.

2.2.2 Architektur

Die Architektur des OSGi Frameworks besteht aus vier Schichten. Diese Schichten bilden die Grundlage einer jeden Implementierung und werden durch die OSGi Alliance vorgegeben. Abb. 2.6 zeigt den Aufbau der Architektur.

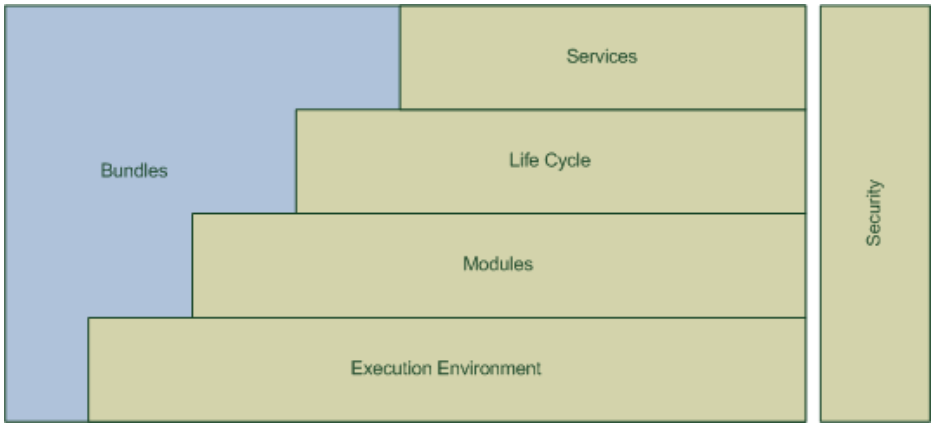

Abbildung 2.6: Architektur OSGi

2.2.2.1 Execution Environment

Die unterste Schicht bildet das Execution Environment. Das Execution Environment bildet sich aus der Spezifikation der Java Umgebung, in der das Framework ausgeführt wird. Es wird dabei zwischen den normalen Java Runtimes ab Version 1.1 und den von der OSGi Alliance veröffentlichten OSGi/Minimum-1.0 und OSGi/Minimum-1.1 Umgebungen unterschieden. In den von der OSGi Alliance spezifizierten Umgebungen werden grundlegende Anforderungen an die Runtime gestellt, so dass es möglich ist, OSGi auf verschiedenen Java Umgebungen auszuführen.

2.2.2.2 Modules

Die Modul-Schicht definiert die Classloader Funktionen und bildet das Grundkonzept der Modularisierung. Die einzelnen Module werden auch als Bundle bezeichnet. Das OSGi Framework besitzt leistungsstarke Funktionen zum Laden von Komponenten. Es basiert auf dem Classloader Modell aus Java, erweitert dies aber um eine höhere Modularisierung zu erreichen. Der normale Classloader in Java kennt nur einen Pfad in dem sich Ressourcen und

Klassen befinden. Die Modulschicht fügt diesem Konzept hinzu, dass jedes Bundle einen eigenen Classloader besitzt. Zusätzlich bietet es die Möglichkeit, zwischen privaten und öffentlichen Komponenten zu unterscheiden, dies verbessert die Kapselung der einzelnen Komponenten. Der Security Layer überwacht die Aufrufe auf ein Bundle, um unberechtigte Aufrufe abzufangen.

2.2.2.3 Life Cycle

In der Modul-Schicht wird lediglich das statische Konzept der Modularisierung definiert. Die Life Cycle Schicht erweitert dies um die dynamischen Anteile. Hier wird der Lebenszyklus eines Bundles beschrieben. Dies beinhaltet die einzelnen Zustände die ein Bundle annehmen kann und die Übergänge zwischen diesen. Abb. 2.7 zeigt den Lebenszyklus eines OSGi Bundles.

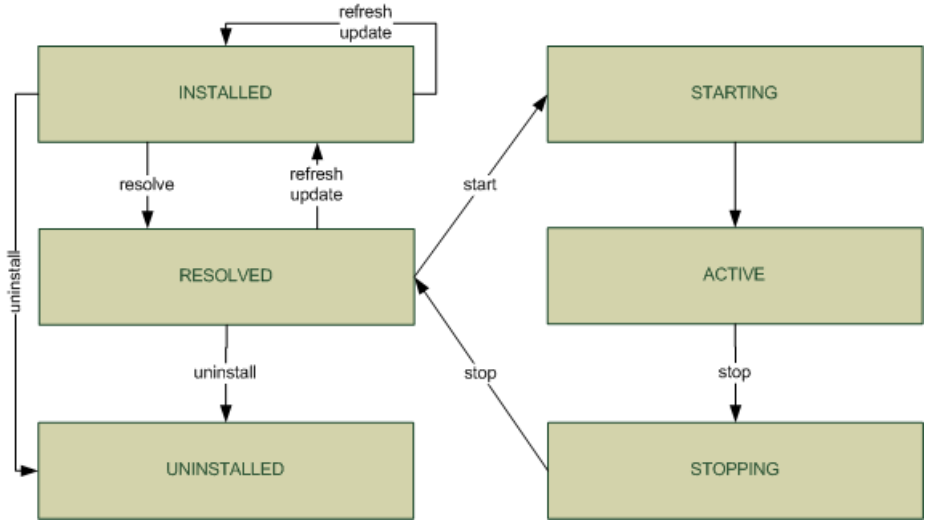

Abbildung 2.7: Bundle Lebenszyklus

Im folgenden sollen die Zustände, die ein Bundle annehmen kann, kurz erläutert werden.

INSTALLED Ein Bundle befindet sich in den Zustand INSTALLED, wenn alle Daten des eigenen Bundles installiert wurden.

RESOLVED Nach dem Zustand INSTALLED folgt der Zustand RESOLVED. Dieser wird erreicht, wenn alle Abhängigkeiten mit anderen

Bundles aufgelöst werden konnten. Je nach Framework Implementierung erfolgt die Prüfung der Abhängigkeiten direkt nach der Installation des Bundles, oder beim ersten Start.

STARTING Ein Bundle geht in den Zustand STARTING, wenn es vom Framework ein Signal zum Start bekommt. Dieser Zustand hält solange an, bis die Startmethode des Bundles abgeschlossen ist und geht anschließend in den Zustand ACTIVE. Sollte bei der Ausführung der Startmethode eine Exception auftreten, geht das Bundle zurück in den Zustand RESOLVED.

ACTIVE Nachdem die Startmethode des Bundles ohne Exception beendet wurde, ist das Bundle im Zustand ACTIVE. Das Bundle läuft nun im Framwork und auf seine bereitgestellten Dienste kann zugegriffen werden.

STOPPING Ähnlich wie beim Zustand STARTING wird der Übergang zum Zustand STOPPING durch ein Signal erzeugt. Dieses veranlasst das Bundle seine Stopmethode aufzurufen. Nach Rückkehr aus der Stopmethode geht das Bundle in den Zustand RESOLVED.

UNINSTALLED Wenn ein Bundle im Framework nicht mehr benötigt wird, kann es deinstalliert werden. Ein Bundle kann dann entfernt werden, wenn es sich im Zustand INSTALLED oder RESOLVED befindet. Während die Daten des Bundles aus dem Framework entfernt werden, ist dies im Zustand UNINSTALLED.

Der Wechsel zwischen den einzelnen Zuständen werden durch Signale des Frameworks ausgelöst. Diese Signale können vom Framework selbst kommen, oder vom Benutzer gesteuert werden. Die meisten OSGi Implementierungen bieten dafür eine Managementkonsole an. Damit ist es zum Beispiel möglich, Bundles zu installieren, zu starten und zu stoppen.
Die beiden Signale *update* und *refresh* dienen zum Aktualisieren eines bereits installierten Bundles. Bei dem Signal update werden lediglich die Daten des Bundles aktualisiert. Exportierte Komponenten des alten Bundles bleiben solange bestehen, solange sie von anderen Bundles importiert werden. Wurden Aktualisierungen an den exportierten Komponenten durchgeführt, sollte ein refresh durchgeführt werden. Dadurch werden alle Abhängigkeiten des Bundles neu aufgelöst und alte Verbindungen getrennt.

2.2.2.4 Service-Schicht

In dieser Schicht wird die Service-Registry definiert. Sie dient dazu, von Bundles angebotene Services, oder auch Dienste genannt, zentral zu verwalten. Grundsätzlich wäre es zwar möglich über geteilte Komponenten Funktionen anzubieten, dies ist aber in einer dynamischen Umgebung nicht sinnvoll. Das OSGi Framework bietet als, dynamische Alternative, Dienste an, die von Bundles zur Laufzeit registriert und deregistriert werden können. Die Dienste bestehen dabei aus normalen Java Objekten.
Das OSGi Framework bringt eine Reihe von Diensten bereits mit, die vom Entwickler verwendet werden können. Diese sind unter anderem:

Log Service Der Log Service erfasst Informationen, Warnungen, Debug Informationen und Fehler. Diese leitet er an Bundles weiter, die diese Informationen verarbeiten. Die Bundels können sich zu diesem Zweck an den Log Service anmelden und bestimmen welche Informationen sie bekommen möchten.

User Admin Servie Dieser Dienst kann für die Authentifizierung und Authentisierung von Benutzern genutzt werden. Dazu werden in einer Datenbank Benutzerinformationen und Berechtigungen hinterlegt. Dadurch kann das gesamte Framework eine Benutzerverwaltung nutzen.

Component Runtime Um die Dynamik von Diensten zu verbessern kann, die Component Runtime verwendet werden. Diese erlaubt es, Dienste deklarativ zu beschreiben. Dazu wird eine XML Notation verwendet. Dies erlaubt es Dienste erst dann zu starten, wenn alle Komponenten verfügbar sind und der Dienst auch genutzt wird.

Event Admin Über den Event Admin Service wird ein generisches Event System bereitgestellt. Damit können Bundles Nachrichten ins Framework senden und empfangen. Jede Nachricht bekommt beim Versenden ein Topic zugeteilt. Dieses kann vom Bundle, welches es versendet, bestimmt werden. Bundles, die Nachrichten empfangen wollen, können dies über den EventHandler anmelden. Zusätzlich kann der Empfänger filtern, welche Nachrichten er bekommen möchte. Dadurch stellt das Event System eine mächtige Möglichkeit der Interbundlekommunikation dar.

2.2.2.5 Security Schicht

Die Security Schicht regelt die Berechtigungen der einzelnen Bundles. Die Security Schicht basiert dabei auf der Java2 Security und erweitert diese um OSGi spezifische Komponenten. Dazu zählen unter anderem:

Package Permission Diese Berechtigung erlaubt es einem Bundle, Pakete zu importieren und zu exportieren.

Bundle Permission Die Bundle Permission ist ähnlich wie die Package Permission. Der Unterschied liegt darin, dass es nicht nur andere Pakete importieren darf, sondern dass ein Bundles auch von einem anderen abhängen darf. Damit die Abhängigkeiten aufgelöst werden können, müssen alle beteiligten Bundles diese Berechtigung haben.

Service Permission Diese Berechtigung wird benötigt, wenn das Bundle Services von der Service-Registry anfordert. Es können Einschränkungen vorgenommen werden, sodass ein Bundle nur einen bestimmten Service abrufen darf. Dadurch kann verhindert werden, dass ein Modul Zugriff auf Dienste bekommt, die für seine Aufgaben nicht benötigt werden.

Admin Permission Mit dieser Berechtigung ist es einem Bundle möglich, adminstrative Aktionen im Framework auszuführen. Dazu zählen zum Beispiel die Installation von weiteren Bundles und das Starten und Stoppen. Über Filter können diese Rechte eingeschränkt werden. Diese Berechtigung sollte nur wenigen, für die Administration nötigen, Bundles gegeben werden, um die Sicherheit der Anwendung zu erhöhen.

Für weitere Informationen über das OSGi Framework und dessen Architektur möchte ich zum einen auf die Bachelor Thesis [3] von Andreas Karp, sowie auf das Buch "Die OSGi Service Platform" [13] verweisen.

2.2.3 Bundle Struktur

In diesem Kapitel soll anhand eines HelloWorld Bundles der Aufbau eines OSGi Bundles erläutert werden. Nach Erstellung eines Plugin Projekt in Eclipse wird folgende Projektstruktur automatisch aufgebaut.

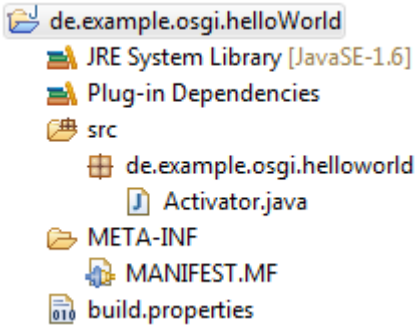

Abbildung 2.8: OSGi Bundle Aufbau

Activator

In der Klasse Activator.java sind die Start und Stop Methoden des Bundles hinterlegt. Diese werden ausgeführt, wenn das Bundle im Framework gestartet bzw. gestoppt wird. Dazu implementiert die Klasse das Interface BundleActivator. In der Start Methode können zum Beispiel Dienste registriert und abgerufen werden, oder persistente Daten aus einer Datenbank gelesen werden. In der Stop Methode können andere Bundles über das Stoppen informiert werden und flüchtige Daten geschrieben werden. Es ist darauf zu achten, dass innerhalb dieser Methoden keine lang andauernden Berechnungen o.Ä. durchgeführt werden, da das Bundle erst nach Beendigung der Start/Stop Methode vollständig gestartet bzw. gestoppt ist. Listing 2.5 zeigt den Activator des HelloWorld Bundles.

```
1 public class Activator implements BundleActivator {
2     private static BundleContext context;
3
4     public void start(BundleContext bundleContext) throws
          Exception {
5         Activator.context = bundleContext;
6         System.out.println("Hello World");
7     }
```

```
8       public void stop(BundleContext bundleContext) throws
            Exception {
9           Activator.context = null;
10          System.out.println("Goodbye World");
11      }
12
13      public static BundleContext getContext(){
14          return Activator.context;
15      }
16  }
```

Listing 2.5: Activator.java

Manifest

Die Manifest Datei beschreibt das Bundle im OSGi Framework. Sie enthält Informationen wie Bundle Name und Version sowie zu seinen Abhängigkeiten. Nachfolgend soll die Manifest Datei des HelloWorld Bundles betrachtet werden.

```
Manifest-Version: 1.0
Bundle-ManifestVersion: 2
Bundle-Name: HelloWorld
Bundle-SymbolicName: de.example.osgi.helloWorld
Bundle-Version: 1.0.0.qualifier
Bundle-Activator: de.example.osgi.helloworld.Activator
Import-Package: org.osgi.framework;version="1.3.0"
Export-Package: de.example.osgi.helloworld
Bundle-RequiredExecutionEnvironment: JavaSE-1.6
```

Abbildung 2.9: Bundle Manifest

Wie in Abb. 2.9 zu sehen, ist die Manifest Datei nach dem Schema TAG:WERT aufgebaut. Über den Bundle-SymbolicName wird das Bundle im Framework referenziert. Bundle-Name dient lediglich zur besseren Identifikation für den Entwickler. Über Import und Export-Package werden die Abhängigkeiten zwischen den Bundles definiert. Durch diese Einträge ist es dem Framework möglich, Abhängigkeiten aufzulösen. Zusätzlich zu den OSGi Informationen können der Manifest Datei weitere Informationen angehängt werden, um das Bundle besser verwalten zu können.

Kapitel 3

Implementierung

Mit Hilfe der in Kapitel 2 beschriebenen Grundlagen kann nun ein Konzept für eine Implementierung erstellt werden. Dazu wurden die Anforderungen an das System analysiert und daraus benötigte Teilmodule abgeleitet.
Als OSGi Framework wird Apache Felix zum Einsatz kommen, welches sich bereits im vorhergegangenen Praxisprojekt[2] als stabiles leichtgewichtiges Framework herausgestellt hat. Als Endgerät wird ein Samsung Galaxy ACE mit der Android Version 2.2 verwendet, welches ebenfalls bereits im Praxisprojekt genutzt wurde.
Da zur Zeit keine lauffähige Implementierung von Distributet OSGi verfügbar ist, wird die komplette Anwendung auf dem Smartphone ausgeführt. Dies hat den Nachteil das rechenintensive Anwendungsteile nicht ausgelagert werden können. Deshalb muss darauf geachtet werden, eine möglichst leichtgewichtige Anwendung zu entwickeln. Jedoch bietet es auch den Vorteil, dass dieses System als Einstiegsvariante für ein AAL System genutzt werden kann, da nur das Smartphone, ein Router und die entsprechenden Endgeräte benötigt werden. Somit stellt es eine günstige Alternative zu großen Implementierungen dar.
Für die Verbindung zwischen Smartphone und Geräten wird 6LoWPAN verwendet. Über eine Netzwerkbrücke werden die 6LoWPAN Pakete ins WLAN weitergeleitet, wo diese als IPv6 Pakete verfügbar sind.
Als Repository für Device und View Bundles wird ein Apache Webserver sowie eine MySQL Datenbank verwendet. Für Abfragen wird ein PHP Script verwendet, dadurch ist es möglich vom Smartphone über eine HTTP Verbindung die Datenbank anzusteuern.

3.1 Architektur

Anhand der Anforderungen wurde eine Architektur entwickelt, welche die benötigten Komponenten und deren Verbindung untereinander darstellt.

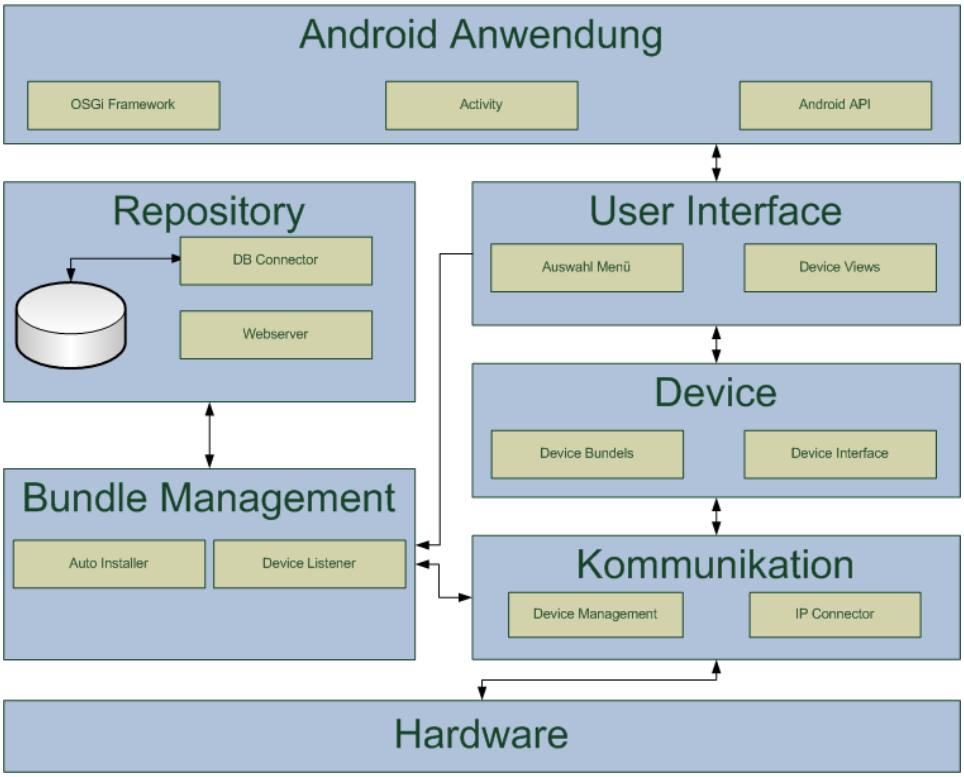

Abbildung 3.1: Basis Architektur

Die Architektur ist angelehnt an die WieDAS Architektur, damit die in diesem Projekt entwickelten Module ins WieDAS Projekt einfließen können. Zusätzlich bietet dies die Möglichkeit, bereits in WieDAS vorhandene Module durch kleine Änderungen in dieses System zu übernehmen. Auf den folgenden Seiten soll nun auf die Implementierung und Funktion der einzelnen Module eingegangen werden.

3.2 Android Basisanwendung

Die Android Basisanwendung stellt, wie in Abb. 3.1 gezeigt, drei Komponenten zur Verfügung. Der Zusammenhang der einzelnen Komponenten wird in Abbildung 3.2 dargestellt.

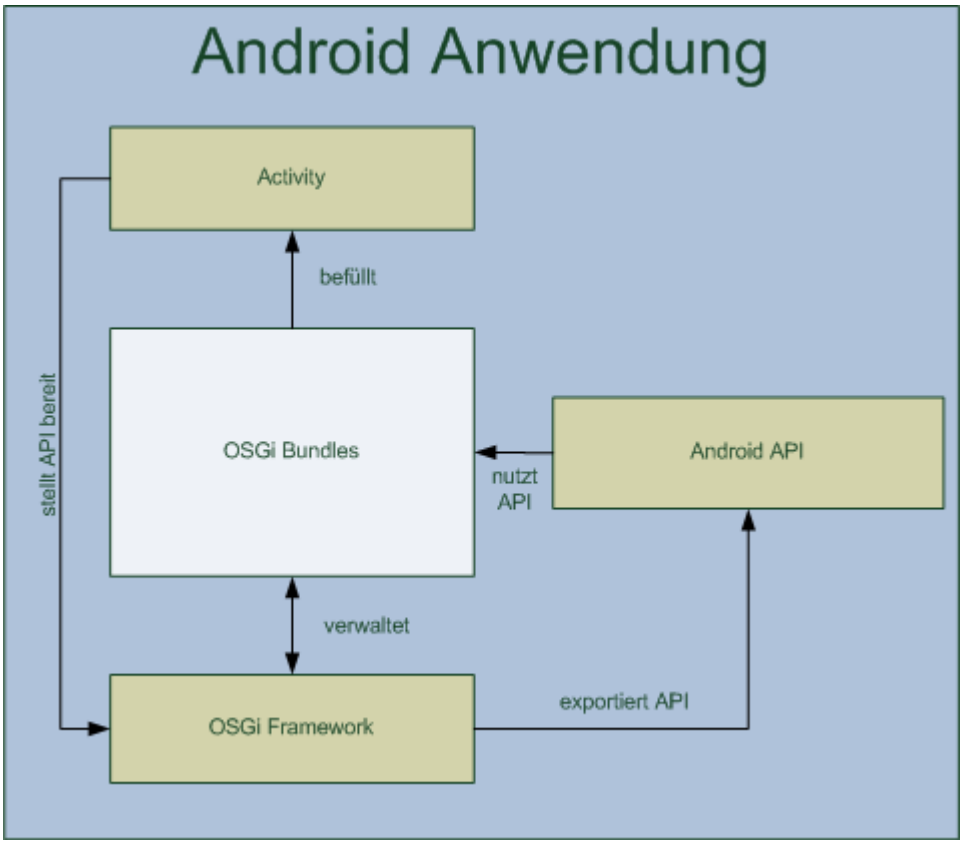

Abbildung 3.2: Komponenten der Basisanwendung

Dabei stellt die Basisanwendung die Hülle der gesamten Architektur dar. In ihr wird zum einen das OSGi Framework bereitgestellt. Dieses bietet die Möglichkeit das System durch OSGi Bundles zu erweitern. Zum anderen stellt es die Activity bereit in der die Oberflächen dargestellt werden können. Um aus den OSGi Bundles auf die Android API zurückzugreifen werden diese ebenfalls von der Basisanwendung bereitgestellt.
Die in der Abbildung 3.2 gezeigte Komponente *OSGi Bundles* beinhaltet alle Bundles die im OSGi Framework installiert sind. Diese sind nicht Teil der

Basisanwendung sondern sollen lediglich den Aufbau verdeutlichen. Durch diese Bundles kann das System erweitert werden ohne das die Basisanwendung angepasst werden muss.
Auf den folgenden Seiten werden nun die einzelnen Komponenten und deren Funktion genauer betrachtet.

OSGi Framework

In der Basisanwendung wird das Apache Felix Framework eingebettet. Dazu wurde das Bundle mit dem Framework der Anwendung als Bibliothek hinzugefügt. Um nun eine Instanz des Frameworks zu starten wurde eine Initialisierungs-Methode entwickelt, auf die nun näher eingegangenen wird. Als erstes werden die Properties für den Framework Cache, Exportierte Pakete und die Remote Konsole gesetzt. Dies ist im folgenden Listing zu sehen:

```
1  [...]
2  private void init(){
3      Properties configProps = new Properties();
4      File chacheDir = this.getFilesDir();
5      configProps.setProperty(
6          Constants.FRAMEWORK_STORAGE,
7          chacheDir.getAbsolutePath());
8      configProps.setProperty(
9          Constants.FRAMEWORK_SYSTEMPACKAGES_EXTRA,
10         ANDROID_FRAMEWORK_PACKAGES);
11     configProps.setProperty(
12         "osgi.shell.telnet.ip",
13         "172.23.25.69");
14     [...]
```

Listing 3.1: Init Methode 1/4

Anschließend kann eine Instanz des Felix Framework erstellt werden und diese gestartet werden. Da beim Starten des Frameworks Exeptions auftreten können, wird es innerhalb eines Try Catch Blocks gestartet. Nach Abschluss der Felix Start Methode kann über die Variable felix auf das Framework zugegriffen werden. Listing 3.2 zeigt die Instanziierung und das Starten des Frameworks.

```
1     [...]
2     felix = new Felix(configProbs);
3     System.out.println("Starting Felix");
4     try {
5         felix.start();
6     [...]
```

Listing 3.2: Init Methode 2/4

Um das Framework besser administrieren zu können, wird eine Remote Konsole über Telnet genutzt. Dazu ist es notwendig das zusätzlich zwei Bundles installiert werden diese sind:

- **org.apache.felix.shell** - Lokale Konsole
- **org.apache.felix.shell.remote** - Zugriff auf Konsole via Telnet

Für die innerhalb des Projektes verwendeten Standart Services werden ebenfalls zwei Bundles benötigt. Dabei handelt es sich um das EventAdmin Bundle und die Service Component Runtime. Alle oben genannten Bundles sind auf der Apache Felix Projekt Homepage zu finden [14]. Beispielhaft soll die Installation des EventAdmin Bundles gezeigt werden. Weiter Bundles können analog dazu ebenso installiert werden.

```
1     [...]
2     Bundle eventAdminBundle = felix.getBundleContext().
3         installBundle("file:///mnt/sdcard/bundles/" +
4         "org.apache.felix.eventadmin-1.2.2.jar");
5     eventAdminBundle.start();
6     [...]
```

Listing 3.3: Init Methode 3/4

Das OSGi Framework ist bestrebt beim Starten den letzten bekannten Zustand wiederherzustellen. Dabei werden alle Bundles, die beim letzten Stoppen des Frameworks geladen waren, wieder geladen. Dies hat den Vorteil dass Einstellungen erhalten bleiben und bereits installierte Geräte sofort verfügbar sind. Um ein definierten Start zu bekommen und immer mit dem Auswahlmenü zu starten, wurde eine kleine Routine entwickelt, die alle anderen geladenen Views beendet. Listing 3.4 zeigt diese Routine.

```
1     [...]
2     Bundle bundles[] = felix.getBundleContext().getBundles();
3     for (int i=0;i<bundles.length;i++){
4         if(bundles[i].getSymbolicName().endsWith("view")){
5             bundles[i].stop();
6         }
7     }
8     [...]
```

Listing 3.4: Init Methode 4/4

Nach Abschluss der Initialisierungs-Methode steht ein OSGi Framework zur Verfügung, das über eine Remote Konsole gesteuert werden kann. Zusätzlich stellt es Bundles für die Verwendung des EventAdmins und der Service Component Runtime für Declarative Services bereit. Aufbauend auf dieses Framework können nun die weiteren Komponenten implementiert werden

Activity

Die Activity dient als Display für die Views der einzelnen Geräte. Um die einzelnen Views in der Activity anzuzeigen, wurde ein Service entwickelt, der beim Starten einer View Komponente diese in die Activity lädt.
Die Activity der Basisanwendung besteht lediglich aus einem LinearLayout. Diese kann nun über einen Service befüllt werden. Die Android Anwendung ist dabei der Service Consumer. Über einen ServiceTracker wird er über Bundles informiert, die eine entsprechende Implementierung anbieten. Das Interface des Service ist in Listing 3.5 dargestellt.

```
1     public interface ViewFactory {
2         public View create(Context context);
3     }
```

Listing 3.5: ViewFactory Interface

Sobald nun ein Bundle gestartet wird, welches eine Implementierung des Service bietet, wird über den ServiceTracker die addingService Methode aufgerufen. Diese erstellt eine Referenz auf den Service und lädt die neue View ins Layout. Das folgende Listing zeigt die dafür angelegte Methode.

```
1  public Object addingService(ServiceReference ref) {
2      final ViewFactory fac = (ViewFactory) felix.
           getBundleContext().getService(ref);
3      if (fac != null) {
4          runOnUiThread(new Runnable() {
5              public void run(){
6              layout.addView(fac.create(FelixStart.this));
7              }
8          });
9      }
10 System.out.println("View added");
11 return fac;
12 }
```

Listing 3.6: addingService Methode

Da in Android ähnlich wie bei RCP Anwendungen nur der Thread, der die Oberfläche erstellt hat, auf diese zugreifen kann wird von der Activity eine Methode bereitgestellt mit der auf den UI-Thread zugegriffen werden kann. Dies ist die runOnUiThread Methode, welche eine Runnable Instanz übergeben wird, welche anschließend in dem UI-Thread ausgeführt wird.
Um eine neue Oberfläche zu laden, sollten zunächst alle Benutzeroberflächen aus dem Layout entfernt werden. Dies geschieht, wenn die alte Benutzeroberfläche entladen wird. Dazu wird ebenfalls der ServiceTracker verwendet. Sobald das Bundle mit der alten Benutzeroberfläche gestoppt wird, wird die removeService Methode aufgerufen. Danach wird geprüft, ob es sich bei der alten Benutzeroberfläche um das Auswahlmenü gehandelt hat. Ist dies nicht der Fall, wird als nächstes versucht, das Auswahlmenü zu starten. Es werden alle installierten Bundles durchsucht, wenn das Bundle mit dem Auswahlmenü gefunden wurde, wird dies gestartet und in das Layout der Anwendung geladen. Die removeService Methode ist im Listing 3.7 zu sehen.

```
1  public void removedService(final ServiceReference ref,
2                             Object service) {
3      felix.getBundleContext().ungetService(ref);
4      runOnUiThread(new Runnable() {
5  [...]
```

```
6  [...]
7  public void run() {
8      layout.removeAllViews();
9      String bundleID = ref.getBundle().getSymbolicName();
10     if(!(bundleID.equals("de.fhd.inflab.android.viewmenu"))){
11         org.osgi.framework.Bundle[] bundles = ref.getBundle().
               getBundleContext().getBundles();
12         for(int i=0 ; i<bundles.length ; i++){
13             if(bundles[i].getSymbolicName().
14                 equals("de.fhd.inflab.android.viewmenu"))
15             try {
16                 bundles[i].start();
17             } catch (BundleException e) {
18                 e.printStackTrace();
19             }
20         }
21     });
22 }
```

Listing 3.7: removeService Methode

Durch die oben aufgeführten Methoden ist es nun möglich, auf sich dynamisch ändernde Oberflächen zu reagieren. Es ist dabei immer eine Oberfläche aktiv die aktuell angezeigt wird. Abb. 3.3 und 3.4 soll den Ablauf eines Oberflächen Wechsels nochmal verdeutlichen.

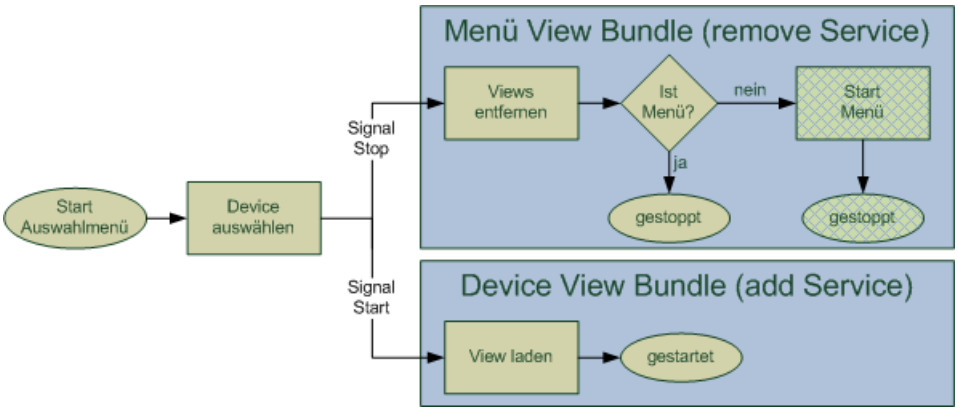

Abbildung 3.3: Starten einer Device View

Die erste Möglichkeit eines Oberflächenwechsels ist ein Umschalten vom Auswahlmenü zu einer Device View. Im Auswahlmenü wird nun ein Gerät ausgewählt. Durch die Auswahl wird an das Menü View Bundle ein Stop Signal gesendet. Das Bundle beginnt nun zu stoppen und deregistriert seinen Service mit dem es seine View zur Verfügung stellt. Dies wird vom ServiceTracker erfasst und die removeService Methode wird aufgerufen. Dadurch werden alle geladenen Views entfernt. Da es sich beim Bunde um das Bundle mit der View des Auswahlmenüs handelt wird nichts weiter unternommen und das Bundle geht in den Zustand gestoppt über. Anschliessend wird das Bundle mit der Device View gestartet. Dieses registriert an der ServiceRegistry nun seine ViewFactory wodurch die addingService Methode aufgerufen wird und die neue View geladen wird.
Die andere Möglichkeit für einen Wechsel ist das Schließen einer Device View. In diesem Fall muss das Auswahlmenü wieder geladen werden. Abb. 3.4 zeigt den dazugehörigen Ablauf.

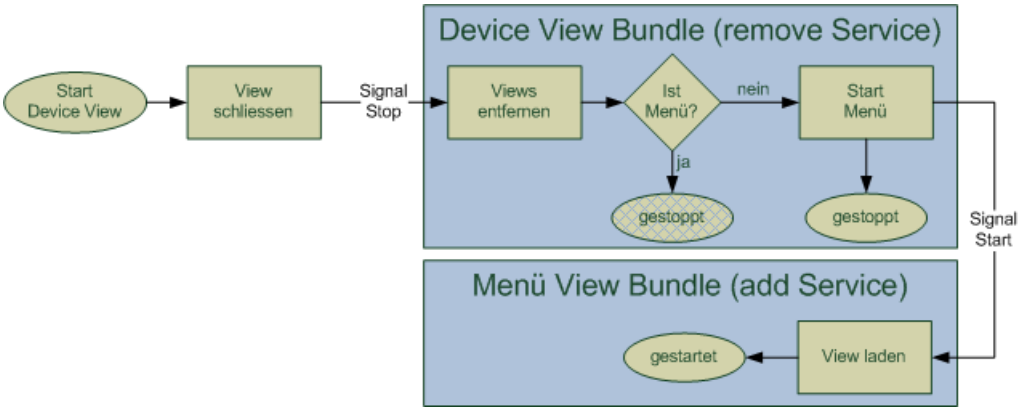

Abbildung 3.4: Starten der Menü View

Anders als der Wechsel zur Device View wird hier nur das Stop Signal an das aktuell geladene Bundle gesendet. Da es in diesem Zustand lediglich möglich ist ins Auswahlmenü zurückzukehren, wird dieses beim Deregistrieren des alten Service geladen. Dies hat den Vorteil, dass sich ein Entwickler, der neue Views entwirft, nicht darum kümmern muss, wie er ins Menü zurückkehrt. Es reicht aus, dass er eine Methode bereitstellt, die sein View Bundle stoppt.

Android API

Um innerhalb der OSGi Bundles auf die Android API zurückzugreifen, ist es notwendig, dass diese von irgendeinem Bundle exportiert werden. Innerhalb der Android Anwendung kann man auf die Android API zugreifen durch das Property FRAMEWORK_SYSTEMPACKAGES_EXTRA, welches in der Init Methode gesetzt wurde, wird dieser Zugriff über das Felix Framework Bundle weitergegeben. Dem Framework wird mit diesem Property ein String übergeben, in dem alle Bundles enthalten sind, die exportiert werden sollen. Der entsprechende String hat den folgenden Aufbau.

```
1  private static final String ANDROID_FRAMEWORK_PACKAGES = (
2              "android; " +
3              "android.app;" +
4              "android.content;" +
5              "android.database;" +
6              "android.database.sqlite;" +
7              "android.graphics; " +
8              [...]
9              "org.bluez; " +
10             "org.json; " +
11             "org.w3c.dom; " +
12             "org.xml.sax; " +
13             "org.xml.sax.ext; " +
14             "org.xml.sax.helpers; " +
15             "de.fhd.inflab.android.framework.Iview").intern();
```

Listing 3.8: Export Android API

Damit die Pakete innerhalb der OSGi Bundles verwendet werden können, müssen diese die benötigten Bundles explizit importieren. Damit während der Entwicklung von Bundles die importierten Bundles aufgelöst werden können, sollte ein Bundle erstellt werden, welches diese exportiert. Dies ist nötig, da Eclipse bei der Entwicklung für OSGi keine Referenz auf Android Projekte zulässt. Das Bundle mit der Android API muss jedoch nicht ins OSGi Framework auf dem Smartphone eingebunden werden. Dort werden die Abhängigkeiten durch die exportierten Pakete des Framework Bundles aufgelöst.

3.3 User Interface

Im Modul User Interface werden die grafischen Benutzerschnittellen für die einzelnen Geräte und das Auswahlmenü bereitgestellt. Die einzelnen Oberflächen sind dabei in OSGi Bundles verpackt. Die Bundles stellen eine Implementierung der ViewFactory bereit, die im vorhergehenden Abschnitt beschrieben wurde. Nachfolgend soll das Bundle mit dem Auswahlmenü und als Beispiel einer Device View eine Benutzeroberfläche für ein Thermometer betrachtet werden.

Auswahlmenü

Im Verlauf des Projektes wurden verschiedene Menü Varianten getestet. Da dieses Projekt im Umfeld vom Ambient Assisted Living durchgeführt wird, gilt es die Benutzeroberflächen entsprechend aufzubauen. Die einzelnen Menüvarianten wurden dabei nach folgenden Punkten bewertet.

- Bedienbarkeit
- Erweiterbarkeit
- Implementierungsaufwand

Ausgehend von verschiedenen Android Anwendungen, in denen es möglich ist, mit dem Finger durch wischen nach rechts und links zwischen verschiedenen Oberflächen zu wechseln wurde dies als erster Ansatz untersucht. Die Bedienbarkeit für wenige Benutzeroberflächen wäre sehr gut, jedoch wird diese mit steigender Anzahl von Oberflächen stetig schlechter, da nicht gezielt zu einer Anwendung gesprungen werden kann. Ältere Menschen können zudem mit der Bedienung Probleme haben, da Fingergesten verwendet werden müssten. Theoretisch ist es möglich beliebig viele Oberflächen in dieser Version anzusteuern. Der Aufwand für eine altersgerechte Implementierung läge jedoch deutlich über den in diesem Projekt verfügbaren Zeitraum.
Im zweite Ansatz wurde eine Menü Variante betrachtet, die aus großen Buttons besteht. Dabei bestand die Oberfläche aus vier großen Buttons. Durch die Größe der einzelnen Buttons war die Bedienung auch für ältere Menschen problemlos möglich. Jedoch stellte sich heraus, dass ohne aussagekräftige Buttons nicht erkannt werden kann, zu welchem Gerät die Schaltflächen gehören. Durch die Größe der Buttons war es problematisch, wenn die

Anzahl der Geräte erhöht. Dafür hätten mehrere Menüebenen erstellt werden müssen, um alle Geräte ansprechen zu können. Dadurch hätte sich der Implementierungsaufwand drastisch erhöht und ist für dieses Projekt nicht praktikabel.
Das letztendlich im Projekt eingesetzte Menü ist ein Kompromiss der ersten Versionen. Die einzelnen Oberflächen werden durch ein Icon und einen Text repräsentiert. Durch den zusätzlichen Text ist sofort ersichtlich, zu welchem Device eine Oberfläche gehört. Die tabellarische Anordnung ist einfach zu erweitern und bietet Platz für einen Vielzahl von Devices. Sobald die Liste nicht mehr komplett auf dem Monitor dargestellt werden kann, kann durch wischen in ihr gescrollt werden. Um zu einer Oberfläche zu wechseln kann, mit dem Finger, auf einen Text gedrückt werden. Zum deinstallieren eines Device kann lange auf einen Eintrag gedrückt werden. Abb. 3.5 zeigt das Auswahlmenü mit zwei Einträgen.

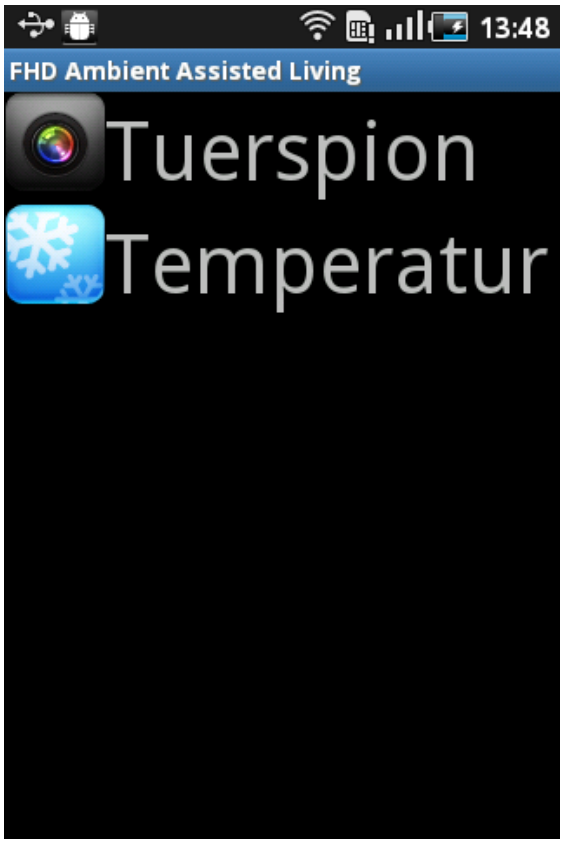

Abbildung 3.5: Auswahlmenü

Im Activator des Auswahlmenü Bundles wird zunächst eine Instanz der View erzeugt und der Service an der ServiceRegistry angemeldet. Durch das Anmelden des Service wird die Oberfläche wie in Kapitel 3.2 beschrieben in die Activity geladen. Das Auswahlmenü besteht aus einer Kombination aus ScrollLayout und TableLayout. Dies ermöglicht das einfache Befüllen des TabelLayout mit den Scroll Funktionalitäten des ScrollLayout zu kombinieren. Die Abb. 3.6 zeigt den Aufbau des Auswahlmenüs in Baumstruktur.

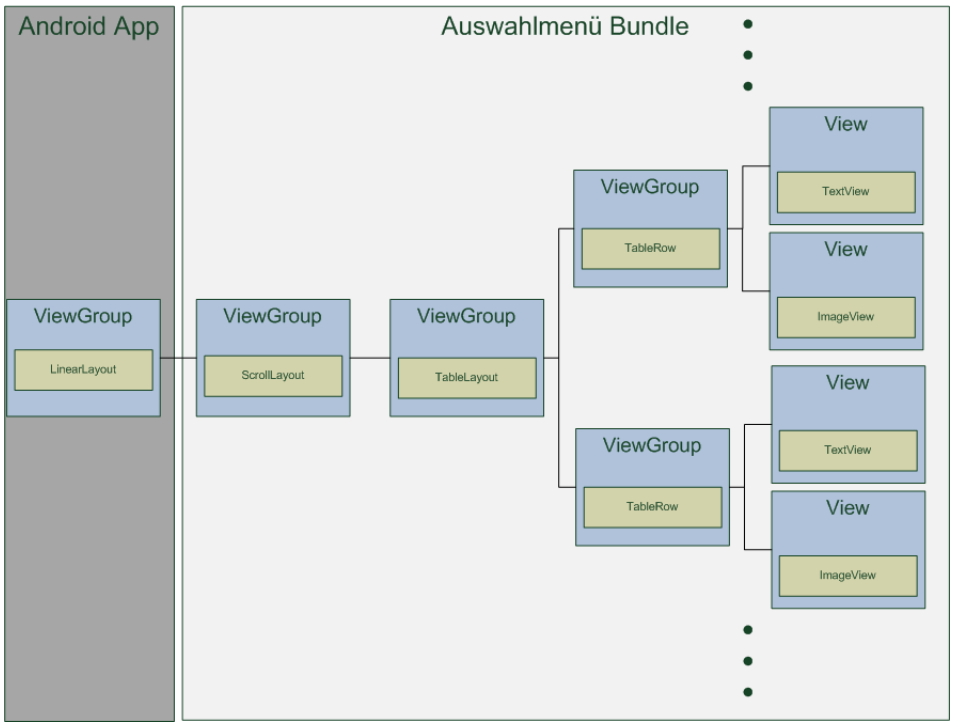

Abbildung 3.6: Aufbau Auswahlmenü

Zum Befüllen der Liste mit den Einträgen der installierten Devices werden mit Hilfe einer Schleife alle installierten Bundles überprüft und die View-Bundles in die Liste übernommen. Der symbolische Name der View-Bundles endet mit *view*, dadurch werden diese von der Anwendung erkannt. Sobald ein View-Bundle erkannt wurde, wird eine neue TableRow angelegt. Wie oben beschrieben besteht ein Eintrag aus einem Icon und einen Text. Das Icon wird durch das View-Bundle bereitgestellt. Sollte das View-Bundle kein Icon enthalten, wird automatisch ein Standard Icon benutzt. Als Text wird der Bundle Name gesetzt, der aus dem Header ausgelesen wird.
Damit der Benutzer durch einen Fingerdruck auf den Eintrag die entspre-

chende Benutzeroberfläche laden kann, wird ein Listener benötigt, der eingehende Ereignisse verarbeitet. Beim kurzen Drücken auf einen Eintrag wird das Auswahlmenü gestoppt und die entsprechende Device View gestartet. Der ausführende Thread ist der UI-Thread, dadurch wird auch nach Beendigung des Auswahlmenüs die ausgewählte View gestartet. Für die Deinstallation eines Gerätes wurde ein OnLongClickListener verwendet. Dieser reagiert, wenn der Benutzer länger als 1 Sekunde auf den Menüeintrag drückt. Wenn dieses Ereignis festgestellt wurde, wird aus dem Header des View-Bundles die Device ID ausgelesen und diese, der uninstall Methode des BundleManagers übergeben.
Um auf Veränderungen im Framework zu reagiern, z.B. die Installation eines neuen Gerätes, wird aus dem OSGi Framework der BundleChangeListener verwendet. Sobald eine Veränderung im Framework eintritt, wird ein Event an den Listener geschickt. Dadurch wird veranlasst, dass alle Einträge der Liste entfernt werden und neu eingelesen werden. Dies stellt die Aktualität des Auswahlmenüs sicher.

Device View

Die Device Views stellen die spezifischen Informationen der einzelnen Geräte graphisch dar. An Hand einer simpelen Beispiels, dass eine erfasste Temperatur anzeigt, soll der Aufbau einer Device View gezeigt werden.
Die Benutzeroberfläche wird durch ein OSGi Bundle repräsentiert. Damit das OSGi Framework zuordnen kann, zu welchem Gerät die Benutzeroberfläche gehört, wurde in der Manifest Datei des OSGi Bundles ein zusätzlicher Header **DeviceID** eingesetzt. Der im Auswahlmenü angezeigte Text für die Oberfläche wird ebenfalls in der Manifest angegeben. Dazu wird der Header **Bundle-Name** verwendet. Der Header **Bundle-SymbolicName** muss mit *view* enden damit dieses Bundle vom Framework als Benutzeroberfläche erkannt wird. Abb. 3.7 zeigt die Manifest Datei der Temperatur View.

```
Manifest-Version: 1.0
Bundle-ManifestVersion: 2
Bundle-Name: Temperatur
Bundle-SymbolicName: de.fhd.inflab.android.temp.view
Bundle-Version: 1.0.0
Bundle-Activator: de.fhd.inflab.android.temp.view.Activator
Import-Package: android.app,
 android.content,
 android.graphics.drawable,
 android.os,
 android.view,
 android.widget,
 de.fhd.inflab.android.framework.Iview,
 org.osgi.framework;version="1.3.0",
 org.osgi.service.event;version="1.2.0"
Bundle-RequiredExecutionEnvironment: JavaSE-1.6
Service-Component: OSGI-INF/component.xml
DeviceID: temp
```

Abbildung 3.7: Manifest Datei Temperatur View

Wie weiter oben beschrieben, stellt das View Bundle ein Icon für den Eintrag im Auswahlmenü ein Icon bereit. Dieses wird in PNG Format im Ordner /res/icon.png hinterlegt. Das Icon sollte möglichst einen quadratischen Formfaktor haben und eine Größe von 60x60 Pixel nicht überschreiten.
Damit die Benutzeroberfläche beim Starten des Bundles in das Layout der Android Anwendung geladen wird, muss das Bundle seinen Service an der Service Registry anmelden. Dies geschieht im Activator des Bundles. Listing 3.11 zeigt die Start Methode im Activator der Temperatur View.

```
1    public void start(BundleContext bundleContext) throws
         Exception {
2        Activator.context = bundleContext;
3        context.registerService(ViewFactory.class.getName(),
4                new TempView(bundleContext), null);
5    }
```

Listing 3.9: Activator Temperatur View

Die Klasse TempView implementiert das ViewFactory Interface, welches vom OSGi Framework in der Android Anwendung exportiert wird. Innerhalb der create Methode wird die Benutzeroberfläche erzeugt und zum Schluss als

View an die Android Anwendung zurückgegeben. Anders als bei einer standard Anwendung für Android, in der die grafische Oberfläche durch XML beschrieben wird, muss innerhalb von OSGi die Oberfläche in Java programmiert werden. Listing 3.10 zeigt dieses Beispielhaft für die Temperatur View.

```
1  public View create(Context context) {
2          LinearLayout layout = new LinearLayout(context);
3          LinearLayout control = new LinearLayout(context);
4          control.setOrientation(LinearLayout.VERTICAL);
5          ImageView thermometerPic = new ImageView(context);
6          thermometerPic.setImageDrawable(Drawable.
               createFromPath("/mnt/sdcard/iKons/thermo.png"));
7          TextView tempText = new TextView(context);
8          tempText.setText("Keine Daten vorhanden");
9          tempText.setTextSize(32);
10         Button exit = new Button(context);
11         exit.setText("Menü");
12         exit.setOnClickListener(new OnClickListener() {
13         @Override
14             public void onClick(View arg0) {
15                 try {
16                     bundleContext.getBundle().stop();
17                     } catch (BundleException e) {
18                     e.printStackTrace();
19                 }
20             }
21         });
22         control.addView(tempText);
23         control.addView(exit);
24         layout.addView(thermometerPic);
25         layout.addView(control);
26         return layout;
27 }
```

Listing 3.10: create Methode Temperatur View

Damit der Benutzer aus der Benutzeroberfläche des einzelnen Gerätes zum Menü zurückkehren kann, wird eine Methode benötigt, die das Bundle stoppt. Im Fall der Temperatur View ist dies durch einen Button gelöst. Denkbar sind auch andere Möglichkeiten wie evtl. vorhandene Knöpfe am Endgerät. Abb. 3.8 zeigt die erstellte Benutzeroberfläche.

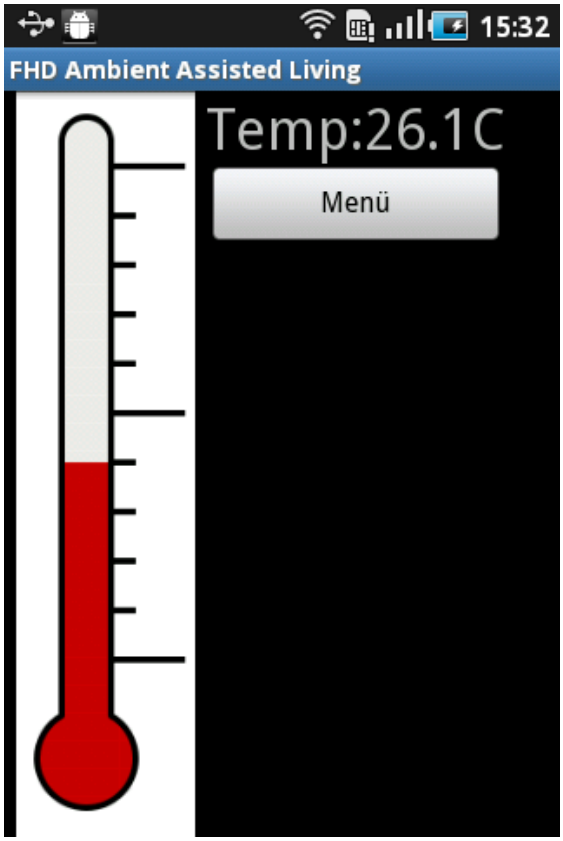

Abbildung 3.8: Temperatur View

Um auf aktuelle Werte zu reagieren, wird das OSGi Event System verwendet. Die View meldet seinen EventHandler, der für die Verarbeitung von Events benötigt wird, in der Service Registry an. Bei der Registrierung des Service wird, mittels einen Topics, angegeben, welche Events an die View weitergeleitet werden. Die im Event enthaltene Daten werden an den UI-Thread weitergereicht und dort mittels Textfeld ausgegeben.

3.4 Device

Für jedes Gerät, welches in diesem System verwendet werden soll, wird ein Device Bundle benötigt. Dies hat die Aufgabe die Geräte spezifischen Eigenschaften innerhalb der Software abzubilden. Dazu zählen zum einen das bereitstellen von Methoden zur Steuerung eines Geräts. Für einen Fensteröffner könnten dies Methoden zum Öffnen und Schließen des Fensters sein. Zum anderen müssen Daten, die vom Gerät gesendet werden, ausgewertet und an die Benutzeroberfläche weitergeleitet werden.
Das Device Bundle bekommt die Daten über das Kommunikationsmodul. Dazu wird das OSGi Event System verwendet. Damit das Kommunikationsmodul die Daten den einzelnen Devices zuordnen kann, müssen sich die Module am Kommunikationsbundle anmelden. Dafür wird die Service Component Runtime verwendet. Diese stellt sobald alle benötigten Komponenten verfügbar sind, die Verbindung zwischen dem Device Bundle und dem Kommunikationsbundle her. Sobald das Device Bundle gestartet wird, führt die Service Component Runtime die Activate Methode aus, welche in der Komponentenbeschreibung des Service definiert ist. Für das Temperatur Device sieht diese folgendermaßen aus.

```
1  public void activate(ComponentContext context) {
2      BundleContext bundleContext = context.getBundleContext();
3      Bundle bundle = bundleContext.getBundle();
4      this.bundleID = bundle.getSymbolicName();
5      if (socketEventReg.startServer(tempPort,
6          ISocketEventRegistration.UDP)) {
7          socketEventReg.addSocketEventHandler(this.bundleID,
8          "temp","de/fhd/inflab/temp");
9          System.out.println("TEMP-Plugin: Event Registrated");
10     }
11 }
```

Listing 3.11: Activate Methode Temperatur Device

Wenn der Server im Kommunikationsbundle ohne Probleme gestartet wurde, leitet er ankommende Daten, mit der DeviceID *temp*, als OSGi Event unter dem Topic *de/fhd/inflab/temp* weiter. Beim Beenden des Device Bundle wird der Server gestoppt und der Service deregistriert.

3.5 Kommunikations Modul

Das Kommunikationsbundle hat in diesem System zwei Aufgaben. Die erste ist das Device Management, die andere ist das Umsetzten von UDP Datenpaketen in OSGi Events. Das Kommunikationsbundle ist ein für dieses System angepasstes Bundle, welches auf dem Kommunikationsbundle aus der RCP Anwendung von Andreas Karp basiert [3]. Wie im Kapitel 3.4 beschrieben melden sich die Device Bundles beim Kommunikationsbundle an. Bei der Anmeldung wird ein UDP Server auf den übergebenen Port geöffnet. Dieser nimmt nun alle UDP Pakete auf dem angegebenen Port entgegen. Ein Paket ist dabei folgendermaßen aufgebaut:

(TIME) /IPAddress#DeviceID:Data

Time: Zeitstempel des Pakets

IPAddress: IPv6 bzw. IPv4 Adresse des absendenden Gerätes

DeviceID: Enthält den Gerätetyps des Absenders

Data: Enthält die gesendeten Daten

Sofern noch kein EventHandler für diesen Gerätetypen registriert ist, wird ein neuer erzeugt. Dieser verarbeitet UDP Pakete bei denen die vom Device Bundle angegebene DeviceID mit der des UDP Pakets übereinstimmt. Dafür werden zunächst die im Datenpaket enthaltenen Informationen getrennt und in einem Dictionary gespeichert. Anschließend wird ein Event erzeugt mit dem vom Device Bundle angegebenen Topic und dem Dictionary als Event Daten. Um das Event ins Framework zu schicken, wird der EventAdmin verwendet. Jede dem System bekannte DeviceID wird in einem Dictionary gespeichert. Dadurch ist eine primitive Geräteverwaltung möglich, auch wenn keine weiteren Bundles für die Verwaltung verfügbar sind. Beim Stoppen oder Deinstallieren eines Device Bundles wird der EventHandler deinstalliert und sofern kein anderes Gerät den selben Port verwendet, wird der UDP Server ebenfalls geschlossen. In diesem Projekt wurde der Device Manager des Kommunikationsbundle um die Möglichkeit zur automatischen Installation erweitert. Dies ist jedoch nur möglich wenn das Bundle Management Modul zur Verfügung steht. Dazu wird bei jeder unbekannten DeviceID über das

Bundle Management ein Request ausgelöst, um zu prüfen, ob im Repository Bundles für dieses Gerät verfügbar sind. Da es auch vorkommen kann, dass aus irgend einen Grund von einem bereits bekannten Gerät Bundles im System fehlen, wurde zusätzlich eine Abfrage implementiert. Diese Abfrage prüft beim Empfang eines Datenpakets von einem bekannten Gerät, ob alle benötigten Bundles installiert sind. Das System ist somit in der Lage sich im gewissen Maße selber zu überwachen und kleinere Probleme selber zu lösen.

3.6 Bundle Management

Wie im vorhergehenden Kapitel beschrieben, stellt das Bundle Management Modul Dienste für das automatische Installieren von Geräten bereit. Dazu wurden zwei Bundles entwickelt. Dies ist zum einen das Device Listener Bundle und ein Bundle, welches die Dienste für die Abfrage und Installation von Geräten bereitstellt.
Das Device Listener Bundle dient dazu, im Kommunikationsbundle einen UDP Server zu öffnen, über den Daten von möglichen Geräten empfangen werden können. Dazu wurde ein Pseudo Device Bundle geschrieben, welches lediglich einen Server öffnet, aber keinen EventHandler im Kommunikationsbundle registriert. Sobald nun ein Gerät Daten an das System sendet, erkennt das Kommunikationsbundle diese als unbekanntes Gerät und versucht diese über den Auto Installer zu installieren.
Der Auto Installer implementiert die Methoden zum Installieren und Deinstallieren von Geräten. Zusätzlich bietet er eine Methode, um zu prüfen ob ein Gerät bereits ordnungsgemäß installiert ist. Die Implementierung der Methoden soll nun genauer betrachtet werden.

deviceRequest Methode

Diese Methode wird von allen drei folgenden Methoden verwendet. Über diese Methode wird ein Request ans Repository gesendet und geprüft, ob Bundles für ein Gerät verfügbar sind. Dazu wird beim Methodenaufruf die DeviceID des Gerätes an die Methode übergeben. Um die Kommunikation zwischen Bundle Manager und Repository schlank und standardisiert zu halten wurden HTTP Aufrufe benutzt. Durch den auf jeden Android verfügbaren HTTP Client ist es möglich, diesen transparent für den Nutzer für

diese Anfragen zu verwenden. Dafür wird der übergebene String zusammen mit dem Key *deviceid* in eine Array List geschrieben. Diese Liste wird später für die Abfrage benötigt.

```
1 [...]
2 ArrayList<NameValuePair> nameValuePairs =
3     new ArrayList<NameValuePair>();
4 nameValuePairs.add(new BasicNameValuePair("deviceid",
5     deviceID));
6 [...]
```

Listing 3.12: HTTP request 1/4

Um eine HTTP Anfrage durchzuführen, wird ein HTTP Client benötigt. Die Android API bietet hierzu eine Klasse, die diese Aufgabe übernimmt. Nach der Instanziierung des HTTP Client kann ein neuer HTTP Post erzeugt werden, welcher unsere Anfrage beinhaltet und ans Repository sendet. Dieses wertet die Anfrage aus und sendet einen Response mit den benötigten Bundle Informationen zurück. Diese sind anschliessend als Inputstream verfügbar. Listing 3.13 zeigt Anfrage und Response.

```
1 [...]
2 HttpClient httpclient = new DefaultHttpClient();
3 HttpPost httppost = new HttpPost("http://"+HOST+"/getit.php");
4 httppost.setEntity(new UrlEncodedFormEntity(nameValuePairs));
5 HttpResponse response = httpclient.execute(httppost);
6 HttpEntity entity = response.getEntity();
7 Inputstream is = entity.getContent();
8 [...]
```

Listing 3.13: HTTP request 2/4

Für eine weitere Verarbeitung der Ergebnisse ist es sinnvoll, die Daten in einem String zu wandeln. Dazu wird ein BufferedReader sowie die StringBuilder Klasse verwendet. Dabei wird der Stream zeilenweise eingelesen und mit Hilfe des StringBuilders in einen String gewandelt. Der String liegt nun in JSON Format vor. JSON bietet ähnlich XML eine Möglichkeit Objekte serialisiert zu übertragen. Durch seinen einfachen Aufbau eignet es sich

besonders für kleinere Webanwendungen und Geräte mit wenig Leistung. Listing 3.14 zeigt die Umwandlung des InputStreams in einen String.

```
1  [...]
2  BufferedReader reader = new BufferedReader(
3      new InputStreamReader(is,"iso-8859-1"),8);
4  StringBuilder sb = new StringBuilder();
5  String line = null;
6  while ((line = reader.readLine()) != null) {
7      sb.append(line + "\n");
8  }
9  is.close();
10 result=sb.toString();
11 [...]
```

Listing 3.14: HTTP request 3/4

Durch das JSON Format ist es möglich, in Java aus dem empfangenen String wieder Objekte zu erzeugen. Dazu wird ein JSONArray erzeugt, dem der String übergeben wird. Innerhalb einer For Schleife können nun aus dem Array wieder Objekte erzeugt werden. Dafür werden die einzelnen Felder ausgelesen und in ein neues Objekt geschrieben. Damit später einfacher auf die einzelnen Elemente zugegriffen werden kann, werden die erzeugten Objekte in einem Dictionary gespeichert, welches anschließend zurück gegeben wird. Listing 3.15 zeigt das Erzeugen der neuen Objekte. Das zurückgegebene Dictionary beinhaltet alle Informationen die für eine Installation des Bundles benötigt werden. Für jedes Bundle, welches installiert werden muss, wird eine Instanz der Klasse BundleClass erzeugt. Diese Klasse beinhaltet folgende Informationen:

ID des Bundles in der Datenbank

SymbolicName des Bundles im Framework

URL für die Installation des Bundles

DeviceID für die Art des Gerätes

```
1 [...]
2 JSONArray jArray = new JSONArray(result);
3 for(int i=0;i<jArray.length();i++){
4     json_data = jArray.getJSONObject(i);
5     bundleList.put(i, new BundleClass(json_data.getInt("id"),
6                     json_data.getString("symbolic_name"),
7                     json_data.getString("url"),
8                     json_data.getString("deviceid")));
9 [...]
```

Listing 3.15: HTTP request 4/4

installDevice Methode

Sobald ein neues Gerät erkannt wurde, wird mit Hilfe der Installationsroutine versucht, die Bundles für das neue Gerät zu installieren. Dafür wird zuerst über die deviceRequest Methode am Repository angefragt, welche Bundels für das Gerät benötigt werden. Durch das von der deviceRequest Methode zurückgegebene Dictionary wird zunächst bestimmt, wie viele Bundles installiert werden müssen. Die Anzahl kann über die size Methode des Dictionary bestimmt werden. Anschliessend wird ein neues Array für die zu installierenden Bundles angelegt. Die Installation der Bundles erfolgt über die in Dictionary enthaltenen URLs. Diese enthalten eine HTTP Adresse, die auf den Pfad im Repository zeigt.
Damit das Gerät nach der Installation sofort genutzt werden kann, wird nach der Installation das Device Bundle gestartet. Dieses meldet sich am Kommunikationsbundle an und kann nun verwendet werden. Durch den BundleChange Listener im Auswahlmenü Bundle wird beim Installieren eines neuen Bundles automatisch geprüft, ob es sich um ein View Bundle handelt. Ist dies der Fall wird es automatisch in das Auswahlmenü übernommen. Listing 3.16 zeigt die Installationsroutine

```
1 public void installDevice(String deviceID){
2     System.out.println("Install bundle: " + deviceID);
3     Dictionary<Integer, BundleClass> device = deviceRequest(
          deviceID);
4     Bundle bundles[] = new Bundle[device.size()];
```

```
5       for(int i=0;i<device.size();i++){
6           System.out.println(device.get(i).getSymbolic_name());
7           try {
8               bundles[i] = bundlecontext.installBundle(device.
                    get(i).getUrl());
9           } catch (BundleException e) {
10              e.printStackTrace();
11          }
12      }
13      for(int i=0;i<bundles.length;i++){
14          if(bundles[i].getSymbolicName().endsWith("device")){
15              try {
16                  bundles[i].start();
17              } catch (BundleException e) {
18                  e.printStackTrace();
19              }
20          }
21      }
22  }
```

Listing 3.16: Installationsroutine

uninstallDevice Methode

Um zu ermitteln, welche Bundles zu einem Gerät gehören, wird bei der Deinstallation ebenfalls am Repository angefragt. Damit wird sichergestellt, dass nur Bundles deinstalliert werden, die für das Gerät zuständig sind. Für die Deinstallation wird zunächst ein Array erzeugt, welches alle zur Zeit im System befindlichen Bundles beinhaltet. Dieses Array wird nun nach den zu deinstallierenden Bundles durchsucht. Dazu werden die symbolischen Namen verglichen und bei einer Übereinstimmung das entsprechende Bundle deinstalliert.

```
1   public void uninstallDevice(String deviceID){
2       Dictionary<Integer, BundleClass> device = deviceRequest(
            deviceID);
3       Bundle installedBundles[] =
4           bundlecontext.getBundles();
```

```
5       for(int i=0;i<device.size();i++){
6           for (int b=0;b<installedBundles.length;b++){
7               if(installedBundles[b].getSymbolicName().
8                   equals(device.get(i).getSymbolic_name()))
9                   try {
10                      installedBundles[b].uninstall();
11                  } catch (BundleException e) {
12                      e.printStackTrace();
13              }
14          }
15      }
16  }
```

Listing 3.17: Deinstallationsroutine

isInstalled Methode

Um zu prüfen ob, ein Gerät ordnungsgemäß im System installiert ist, wurde diese Prüfroutine entwickelt. Zuerst wird analog zur (De)Installationsroutine am Repository angefragt, welche Bundles installiert sein müssten. Anhand des vom Repository zurückgegebenen Dictionary wird nun geprüft, ob alle benötigten Bundles installiert sind. Dafür wird für jedes Bundle das installiert sein sollte, ein Eintrag in ein boolean Array erzeugt, der bei korrekter Installation auf true gesetzt wird. Nachdem alle Bundles getestet wurden, wird geprüft, ob im boolean Array alle Elemente den Wert true haben. Ist dies der Fall, ist das Gerät ordnungsgemäß installiert und die Methode gibt true zurück. Anderenfalls fehlt ein Bundle für das Gerät und es wird false zurückgegeben. Listing 3.18 zeit die Prüfroutine.

```
1   public boolean isInstalled(String deviceID){
2       Dictionary<Integer, BundleClass> device =
3           deviceRequest(deviceID);
4       Bundle installedBundles[] = bundlecontext.getBundles();
5       boolean installed[] = new boolean[device.size()];
6       for(int i=0;i<device.size();i++){
7           for (int b=0;b<installedBundles.length;b++){
8               if(installedBundles[b].getSymbolicName().equals(
                    device.get(i).getSymbolic_name()))
```

```
9                   installed[i] = true;
10              else
11                  installed[i] = false;
12              }
13          }
14      for (int i=0;i<installed.length;i++){
15          if(!installed[i])
16              return false;
17      }
18      return true;
19  }
```

Listing 3.18: Prüfroutine

3.7 Repository

Als letzte Komponente soll auf das Repository eingegangen werden. Es ist dafür zuständig, dem System die benötigten Bundels für ein Gerät bereitzustellen. In diesem Projekt wird dazu ein Apache Webserver und eine MySQL Datenbank verwendet. Beide Komponenten werden, im Paket XAMPP [15], von der Entwicklergruppe Apache Friends zur Verfügung gestellt. Als Interface wird ein PHP Script verwendet, welches es erlaubt über eine HTTP Verbindung die Datenbank abzufragen. Für dieses Projekt wird nur eine Abfrage benötigt. Listing 3.19 zeigt diese Abfrage.

```
1  <?php
2  mysql_connect("127.0.0.1","root","");
3  mysql_select_db("bundledata");
4  $q=mysql_query("SELECT * FROM bundles WHERE deviceid='".
5      $_REQUEST['deviceid']."'");
6  while($e=mysql_fetch_assoc($q))
7      $output[]=$e;
8  print(json_encode($output));
9  mysql_close();
10 ?>
```

Listing 3.19: Datenbank Connector

Zuerst wird eine Verbindung zum lokalen Datenbankserver aufgebaut und die Tabelle mit den eingetragenen Bundles aufgerufen. In dieser wird nun die Anfrage durchgeführt. Dazu wird der SQL Befehl SELECT genutzt. Die Abfrage liefert nun alle Einträge, die als DeviceID den übergebenen Wert hinterlegt haben. Aus diesem Ergebnis wird ein Array erstellt und JSON codiert. Zum Schluss wird die Verbindung zur Datenbank geschlossen.
Durch die Abfrage der Datenbank mittels PHP Script ist es möglich, bei Bedarf weitere Abfragen zu implementieren. Dafür reicht es ein neues PHP Script mit der entsprechenden Abfrage zu erstellen.
Ebenso bietet das Benutzen eines Webservers als Repository einen weiteren Vorteil. Dadurch lässt sich das Repository sehr einfach in ein Rechenzentrum auslagern. Dies ermöglicht es, einen zentrales Repository für alle Benutzer bereit zustellen und dadurch den Verwaltungsaufwand zu minimieren.

3.8 Installation eines Gerätes

Nachdem nun alle Komponenten des Systems erläutert wurden, soll in diesen Kapitel der Ablauf der Installation und das Zusammenwirken der einzelnen Bundles erläutert werden. Ausgehend davon, dass es sich für das System um ein unbekanntes Gerät handelt, ist der Ablauf wie folgt.

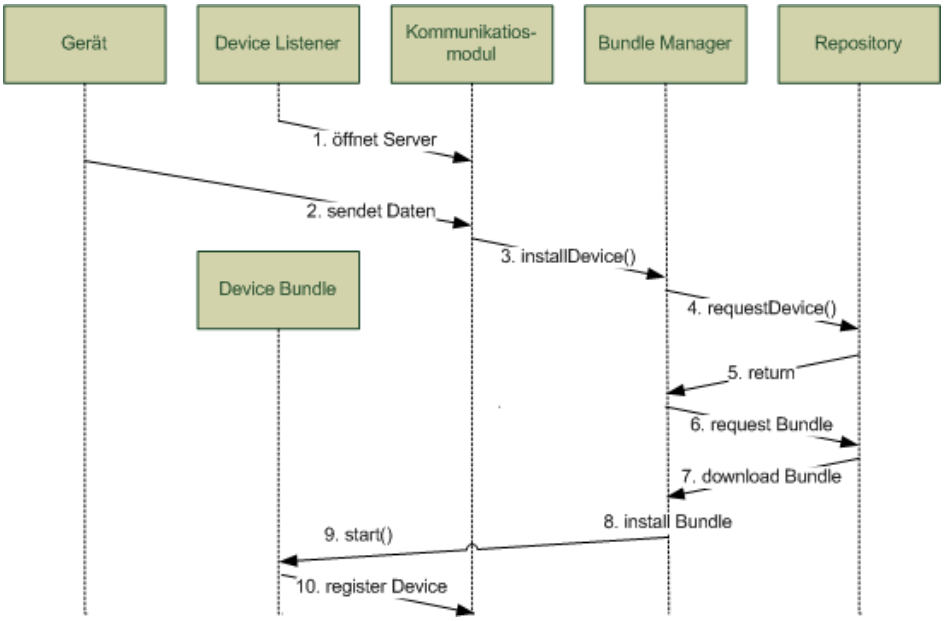

Abbildung 3.9: Installationsverlauf

Beim Start des Systems meldet sich der Device Listener beim Kommunikationsmodul an. Dadurch wird ein Server geöffnet, welcher Datenpakete von neuen Geräten entgegen nimmt. Sendet nun ein Gerät ein Datenpaket, prüft das Kommunikationsmodul, ob es sich um ein neues Gerät handelt. Ist dies der Fall, ruft es die installDevice Methode des BundleManagers auf. Dieser startet nun eine Anfrage beim Repository, welche Bundles für das Gerät benötigt werden. Als Antwort erhält der Bundle Manager ein Dictionary mit den Daten der benötigten Bundles. Über eine HTTP Verbindung werden nun die Bundles heruntergeladen und im System installiert. Nach dem alle benötigten Bundles installiert wurden, wird das Device Bundle des Geräts gestartet, welches sich am Kommunikationsmodul anmeldet. Durch den BundleListener des Auswahlmenüs wird die Installation der neuen Bundles erkannt und die neue Benutzeroberfläche zur Auswahl hinzugefügt. Nun kann das Gerät im System verwendet werden.

Kapitel 4

Inbetriebnahme

In diesem Kapitel soll die Inbetriebnahme des erstellten Systems betrachtet werden. Dazu wird zunächst erläutert wie die einzelnen OSGi Bundles aufs Smartphone exportiert werden. Danach wird die Installation der Basisanwendung erläutert und wie das Repository in Betrieb genommen wird. Abschließend werden die Punkte aufgezeigt, welche Konfiguriert werden müssen, um das System außerhalb des Testcase zu betreiben.

4.1 Exportieren der OSGi Bundles

Die OSGi Bundles die für den Grundlegenden Betrieb zur Verfügung stehen müssen, werden auf der SD-Karte des Smartphones abgelegt, damit diese auch ohne Netzwerkverbindung verfügbar sind. Dazu zählen folgende Bundles:

- **OSGi Shell** - org.apache.felix.shell
- **Remote Shell** - org.apache.felix.shell.remote
- **EventAdmin** - org.apache.felix.eventadmin
- **Service Component Runtime** - org.apache.felix.scr
- **Kommunikations Bundle** - de.fhd.inflab.android.com.socket
- **Device Listener** - de.fhd.inflab.android.bundle.listener
- **Bundle Manager** - de.fhd.inflab.android.bundle.manager
- **Auswahlmenü** - de.fhd.inflab.android.viewmenu

Anhand des Auswahlmenüs soll nun gezeigt werden, wie ein Bundle aufs Smartphone exportiert wird. Dazu wird zunächst das Bundle in Eclipse exportiert. Abbildung 4.1 zeigt diesen Vorgang.

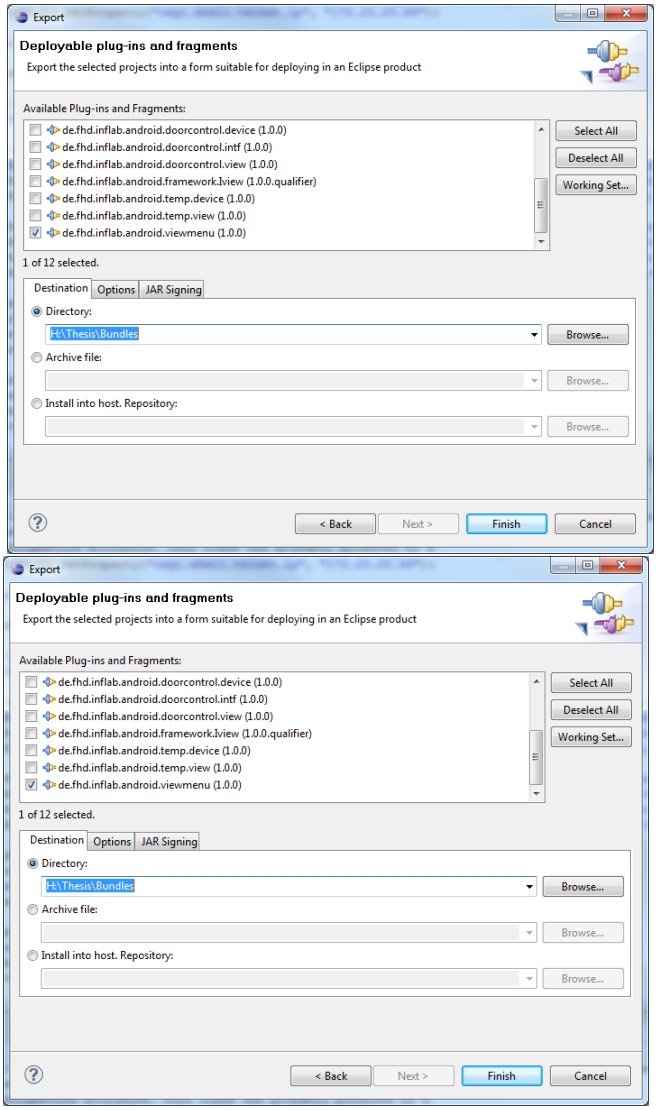

Abbildung 4.1: Bundle Export

Damit die Bundles auf Android ausgeführt werden können, müssen diese noch in Android Bytecode konvertiert werden. Dies wird mit den Hilfsmitteln der Android Developer Tools gemacht. Dazu werden folgende Befehle verwendet:

Erstellen von konvertierten Bytecode
dx `--dex` `--output`=X:\classes.dex X:\de.fhd.inflab.android.viewmenu_1.0.0.jar

Hinzufügen des konvertierten Bytecode zum Bundle
aapt add X:\de.fhd.inflab.android.viewmenu_1.0.0.jar X:\classes.dex

Nun liegt das fertige OSGi Bundle mit Android Bytecode bereit und kann auf die SD-Karte des Smartphone kopiert werden. Für den Testcase wurde dafür auf dem Smartphone ein Ordner *bundles* angelegt. Zum kopieren kann entweder ein Kartenleser verwendet werden, oder das Smartphone mit dem PC verbunden werden und der Zugriff auf die Speicherkarte gestattet werden. Die Freigabe der SD-Karte wird in Abbildung 4.2 gezeigt. Damit die SD-Karte wieder vom Smartphone verwendet werden kann muss die Freigabe wieder abgeschaltet werden.

Abbildung 4.2: Freigabe der SD-Karte

4.2 Installation der Basisanwendung

Nachdem die OSGi Bundles für die grundlegenden Funktionen auf der SD-Karte des Smartphones gespeichert sind, kann die Android Basisanwendung installiert werden. Dafür kann das Eclipse Plugin der Android Developer Tools verwendet werden. Damit die Anwendung auf dem Smartphone installiert werden kann, muss das USB Debugging im Smartphone eingeschaltet

werden. Zusätzlich muss das Installieren aus unbekannten Quellen ermöglicht werden. Abbildung 4.3 zeigt die dafür benötigten Einstellungen auf dem Smartphone.

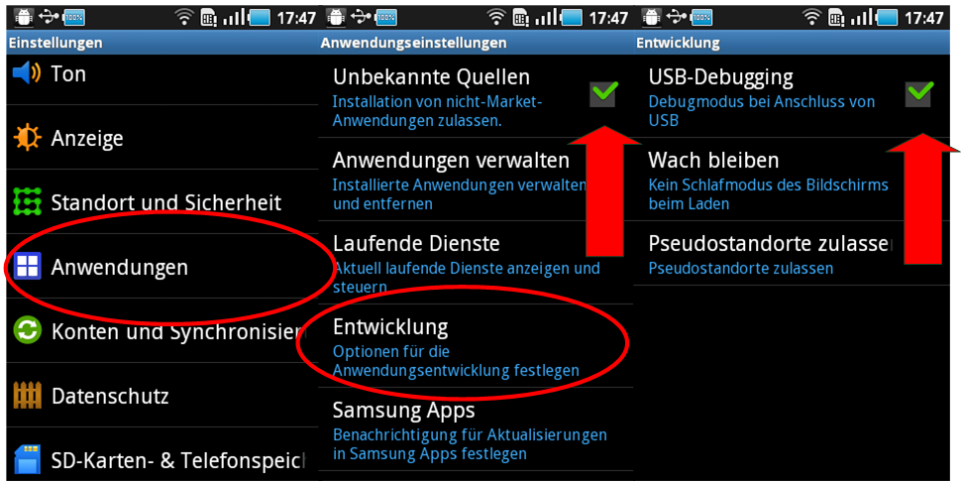

Abbildung 4.3: Einstellungen am Smartphone

Wenn die Einstellungen vorgenommen wurden kann das Smartphone mit dem PC verbunden werden. Damit die Anwendung auf das Smartphone übertragen wird reicht es wenn diese in Eclipse als Android Anwendung gestartet wird. Wenn zusätzlich zum realen Smartphone der Emulator genutzt wird, kann beim starten ausgewählt werden wo die Anwendung gestartet werden soll. Dies zeigt Abbildung 4.4.

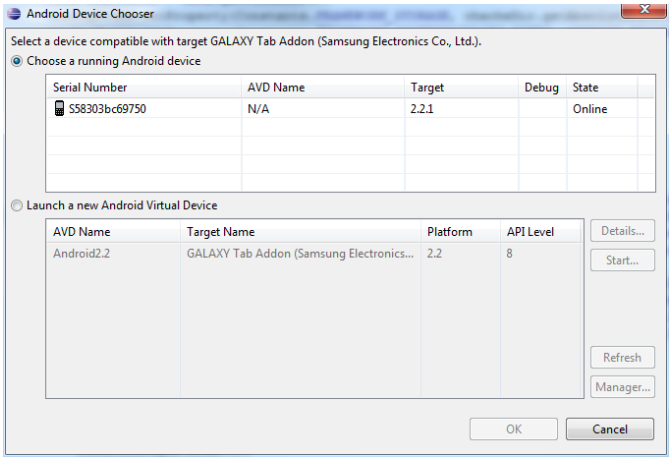

Abbildung 4.4: Starten der Anwendung

4.3 Starten des Repositories

Für das Repository wird wie im vorhergehenden Kapitel beschrieben XAMPP verwendet. Für die Installation kann der Installer des Paketes verwendet werden. Zum Starten des Webservers und der MySQL Datenbank bietet das Paket ein Control Panel an. Dieses ist im Windows Startmenü zu finden. Die zu bereitstellenden Bundels und das PHP Script für die Datenbank Abfrage werden im *htdocs* Ordner im Installationspfad des XAMPP Paketes gespeichert.
Für die Verwaltung der Datenbank bietet das XAMPP Paket das Tool PHPmyAdmin an. Damit können Datenbanken angelegt und verwaltet werden. Dieses Tool erreicht man indem man, über einen beliebigen Webbrowser, die Adresse des Servers aufruft, auf dem das Repository ausgeführt wird. Für den Einsatz innerhalb des Testcase wurden keine weiteren Konfigurationen vorgenommen. Soll das Repository außerhalb des Testcase eingesetzt werden sollten vor allem die Konfigurationen der Sicherheitseinstellungen verbessert werden.

4.4 Konfiguration

Aus Zeitgründen wurden einige Konfigurationen innerhalb des Systems fest in den Quelltext einprogrammiert. In diesem Kapitel sollen diese Einstellungen gezeigt werden, damit das System auch außerhalb der Testcase Umgebung eingesetzt werden kann.

Android Basisanwendung

Innerhalb der Init-Methode wird das Property *osgi.shell.telnet.ip* gesetzt. Dieses beinhaltet die IP des Smartphone und gibt an auf welchem Interface der Telnet Server, für den Remote Zugriff, geöffnet wird. Solange nichts weiteres angegeben ist wird der Port 6666 verwendet.
Ebenfalls in der Init-Methode wird das Installieren der Basis Bundles ausgeführt. Hier muss bei einer Installation außerhalb des Testcase geprüft werden, ob die angegebenen Pfad Angaben noch stimmen.

BundleManager

In der Klasse BundleManager.java ist in der Konstanten HOST die IP-Adresse des Repository einzutragen. Soll zusätzlich auch der Dateiname der Abfragedatei geändert werden kann dies bei der Konstanten PHP erfolgen.

PHP Script

Das PHP Script für die Abfrage der Datenbank liegt im htdocs Ordner des XAMPP Paketes. In ihr sind die Daten für die Datenbankverbindung gespeichert. Diese müssen beim Einsatz außerhalb des Testcase gegebenenfalls angepasst werden. Dazu zählen die folgenden Informationen:

- IP-Adresse des Datenbank Servers
- Benutzername und Passwort des Datenbank Servers
- Datenbank Name
- SQL Abfrage

Hinzufügen neuer Geräte

Um neue Geräte zum Repository hinzuzufügen, müssen alle für das Gerät benötigten Bundles in den htdocs Ordner des XAMPP Paketes kopiert werden. Damit die Bundles bei einer Abfrage der Datenbank berücksichtigt werden, müssen diese Bundles mit Hilfe des PHPmyAdmin Tools in die Datenbank eingetragen werden. Ab diesen Zeitpunkt können die Geräte über das Repository automatisch installiert werden. Zu beachten ist das die symbolischen Namen von Device Bundles mit *.device* enden müssen und View Bundles mit *.view*.

Kapitel 5

Fazit

5.1 Zusammenfassung des Ergebnis

Dieses Projekt zeigt wie mit einer Kombination verschiedener Techniken dynamische Software auch auf mobilen Geräten vorteilhaft eingesetzt werden kann. Durch die weite Verbreitung von Android und seiner für die mobile Nutzung ausgelegten Struktur bietet diese eine ideale Plattform für portable Anwendungen. Zusammen mit der Dynamik des OSGi Framework bildet es eine starke Grundlage für praxistaugliche Systeme für verschiedene Anwendungsgebiete.
Im Rahmen von AAL ist es denkbar, diese Plattform als Benutzerschnittstelle zum System zu benutzen. Wie in diesem Projekt gezeigt, ist aber auch ein autarkes System denkbar. Dieses würde als lokalen Verwaltungspunkt im Haus lediglich einen Router benötigen, der die Pakete der einzelnen Geräte zum Smartphone weiterleitet. Diese Art von AAL System würde die Anschaffungspreise deutlich senken und so die Akzeptanz der Systeme steigern. Durch den Einsatz aktueller Technik ist für dritte zunächst nicht ersichtlich, dass es sich um ein Assistenz System handelt. Dadurch fühlen sich die Benutzer nicht als hilfebedürftig bloßgestellt, was ein weiterer Pluspunkt für ein solches System ist.
Das in diesem Projekt entwickelte System erfüllt alle grundlegenden Anforderungen, die gestellt wurden. Durch eine übersichtliche und einfache Menüführung ist es auch älteren und körperlich benachteiligten Menschen möglich, das System zu bedienen. Für die Installation von neuen Geräten durch den Nutzer sind keine technischen Kenntnisse erforderlich. Dafür wurde ein

Repository eingerichtet, welche alle verfügbaren Device Bundles und Oberflächen beinhaltet. Wenn der Benutzer das Gerät zum ersten mal in Betrieb nimmt, sucht das System im Repository nach den benötigten Bundles und installiert diese selbständig. Damit wird ein hohes Maß an Benutzerfreundlichkeit erreicht. Für die Entwicklung neuer Geräte wurde als Beispiel eine simple Benutzeroberfläche für einen Temperatursensor entwickelt. Durch die an das WieDAS Projekt angepasste Architektur ist es möglich, die bereits für das WieDAS Projekt entwickelten Device Bundles zu übernehmen. Diese Merkmale helfen dabei, die Entwicklungszeit für neue Geräte zu reduzieren.

5.2 Aufgetretene Probleme

Während der Entwicklung kam es zwischenzeitlich zu Problemen durch die Vermischung des Android LifeCycles und dem OSGi LifeCycle. Dadurch kam es vor, dass die Anwendung in nicht definierte Zustände gerät und mit einer Exception beendet wird. Dabei stellte sich vor allem die Socket Verwaltung als problematisch herraus. Dies konnte zum Teil behoben werden, indem das OSGi Framework gestoppt wird, sobald die Android Anwendung in den Hintergrund verschwindet. Dies hat jedoch zur Folge das auftretende Events nur verarbeitet werden, wenn die Anwendung aktiv ist.
Ursprünglich wurde geplant, das Smartphone lediglich als Benutzerschnittstelle zu nutzen. Die restlichen Komponenten sollte auf einem leistungsstärkeren externen System untergebracht werden. Diese Idee musst recht früh verworfen werden, da es zur Zeit keine Implementierung eines Distributed OSGi Service gibt, die auf Android lauffähig ist. Aus diesem Grund wurden die ansonsten auf einem externen System laufenden Komponenten ebenfalls auf dem Smartphone ausgeführt.

5.3 Aussicht

Aus Zeitgründen konnten leider nicht alle Probleme gelöst werden. Um einen stabileren Betrieb zu gewährleisten, müsste vor allem die Kombination der beiden Lebenszyklen besser gelöst werden. Dafür könnten ein Android Services genutzt werden, der ankommende Events abfängt und weitere Aktionen ausführt. Für eine bessere Administration sollte zusätzlich zur Telnetverbinung ein Management Modul angelegt werden, das es erlaubt kleiner

Probleme, die auftauchen, ohne PC zu beheben.
Für weitere Projekte ist es denkbar, dass weitere Komponenten entwickelt werden die speziell auch die mobilen Eigenschaften des Smartphones zugeschnitten sind. Dadurch könnte ein Benutzer zum Beispiel zu Hause am PC oder Fernseher eine Einkaufsliste erstellen. Sobald festgestellt wird, dass die Person das Haus verlässt, wird diese aufs Smartphone übertragen und eine Erinnerung auslöst, wenn die Person den Supermark erreicht. Auch die technischen Möglichkeiten, die ein modernes Smartphone bietet, sind bei weitem noch nicht ausgenutzt. Für die Rotation des Bildschirminhalts besitzt nahezu jedes Smartphone einen Beschleunigungssensor. Dieser könnte als zusätzlicher Sensor für die Sturzerkennung genutzt werden. In Verbindung mit einem größeren AAL System kann das Smartphone auch für die Notfall Kommunikation genutzt werden. Dadurch könnte zum Teil auf teure Spezialgeräte verzichtet werden. Durch den Aufbau der Android Architektur ist auch eine Umgestaltung der standard Benutzeroberfläche sinnvoll. Dadurch wird es speziell älteren Menschen erleichtert, das Endgerät zu bedienen.
Abschließend kann gesagt werden, dass durch dieses Projekt erste Ansätze für eine Nutzung von mobilen Endgeräten in AAL System gewonnen wurden. Das entwickelte System kann nun dafür genutzt werden, weitere Erfahrungen auf diesen Sektor zu erlangen. Für den alltäglichen Einsatz ist es allerdings notwendig, die Benutzerfreundlichkeit und Stabilität weiter zu erhöhen.

Literaturverzeichnis

[1] Dipl. Inform.(FH) Jan Schäfer M. Sc., Dipl. Inform.(FH) Marcus Thoss M. Sc., Prof. Dr. Reinhold Kröger Oliver v. Fragstein B. Sc, Fabian Pursche B. Sc., Prof. Dr. Wolfgang Lux, Prof. Dr. Ulrich Schaarschmidt: *WieDAS - Pflichtenheft.* Hochschule RheinMain & Fachhochschule Düsseldorf, 2010.

[2] Thomas Schmitz: *Entwicklung einer mobilen Software zum Steuern und Überwachen von Wohnungstüren auf Basis von Android im Umfeld von Ambient Assisted Living.* Praxisprojekt, Fachhochschule Düsseldorf, 2011.

[3] Andreas Karp: *Entwicklung eines Frameworks für dynamisch ladbare Benutzeroberflächen im Umfeld von Ambient Assisted Living.* Bachelor Thesis, Fachhochschule Düsseldorf, 2010.

[4] NXP Software: *NXP Software joins the Open Handset Alliance.* 2010. http://www.openhandsetalliance.com/press_releases.html.

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

[6] Open Handset Alliance: *Memberlist.* 2011. http://www.openhandsetalliance.com/oha_members.html.

[7] Google Inc.: *Android Developer Homepage.* 2011. http://developer.android.com/guide/basics/what-is-android.html.

[8] Arno Becker, Marcus Pant: *Android 2 Grundlagen und Programmierung.* dpunkt Verlag, 2010.

[9] Chip Online: *T-Mobile G1 Das Google-Handy ist da.* 2008. http://www.chip.de/news/T-Mobile-G1-Das-Google-Handy-ist-da_32913718.html.

[10] Wikipedia. 2011. http://www.wikipedia.org.

[11] Google Inc.: *Android Developer Guide.* 2011. http://developer.android.com/guide/index.html.

[12] Apache Foundation: *Apache Felix Framework and Google Android.* 2011. http://felix.apache.org/site/apache-felix-framework-and-google-android.html.

[13] Gerd Wütherich, Nils Hartmann, Bernd Kolb Matthias Lübken: *Die OSGi Service Platform.* dpunkt Verlag, 2008.

[14] Apache Foundation: *Apache Felix Framework.* 2011. http://felix.apache.org/site/index.html.

[15] Apache Friends: *XAMPP Project Page.* 2011. http://www.apachefriends.org/de/xampp.html.

[16] Dipl. Inform.(FH) Jan Schäfer M. Sc., Dipl. Inform.(FH) Marcus Thoss M. Sc., Prof. Dr. Reinhold Kröger Oliver v. Fragstein B. Sc, Fabian Pursche B. Sc., Prof. Dr. Wolfgang Lux, Prof. Dr. Ulrich Schaarschmidt: *WieDAS - Eine AAL-Dienstplattform für verteilte Assistenzsysteme.* Hochschule RheinMain & Fachhochschule Düsseldorf, 2010. internes Arbeitspapier.

Abbildungsverzeichnis

Listings

Erklärung

Ich erkläre an Eides Statt, dass ich die vorgelegte Bachelor-Arbeit selbständig angefertigt und keine anderen als im Schrifttumsverzeichnis angegebene Literatur benutzt habe.

Düsseldorf, 12. August 2011 Thomas Schmitz