Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Bachelorarbeit

Viktor Kolbaja

Generischer Smart Home Controller auf Android™ OS

Fakultät Technik und Informatik
Studiendepartment Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Viktor Kolbaja

Generischer Smart Home Controller auf Android™ OS

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung

im Studiengang Bachelor of Science Technische Informatik
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr. Gunter Klemke
Zweitgutachter: Prof. Dr. Birgit Wendholt

Eingereicht am: 22.08.2012

Viktor Kolbaja

Thema der Arbeit
Generischer Smart Home Controller auf Android™ OS

Stichworte
Android™, Smart Home, Controller, Fernbedienung, SmartHomeController, Living Place, Plug-in, generisch

Kurzzusammenfassung
Der steigende Trend von Smarttechnologien in einer Smart Home Umgebung erfordert neue Bedienmöglichkeiten und Bedienkonzepte, was oft zur Vernachlässigung der konventionellen Steuerung führt, besonderes innerhalb einer Laborumgebung, wie Living Place an der HAW-Hamburg. Diese Bachelorarbeit befasst sich mit der Entwicklung einer Software, die die Steuerung der Komponenten des Living Place übernimmt. Dabei spielt die Erweiterung der Funktionalität mittels Plug-ins eine zentrale Rolle.
Viktor Kolbaja

Title of the paper
Generic Smart Home Controller on Android™ OS

Keywords
Android™, Smart Home, Controller, Remote, SmartHomeController, Living Place, Plug-in, generic

Abstract
The rising trend of smart technologies in a smart home environment requires new control options and control concepts, which often leads to neglection of conventional control, particularly within a laboratory environment, such as the Living Place at the HAW in Hamburg. This bachelor thesis deals with the development of software which controls the components of the Living Place. Here the extension of functionality through plugins plays a central role.

Inhaltsverzeichnis

Abbildungsverzeichnis

Listings

1. Einleitung

Als am 12.08.1981 der IBM 5150 den Personal Computer Markt eroberte und der Computer den Einzug ins Wohnzimmer feierte, konnten sich viele Menschen nicht vorstellen, wie eng das Leben des Menschen 31 Jahre später mit einem Computer verknüpft sein wird. Auch wenn es damals noch sehr viele Skeptiker bezüglich der Zugehörigkeit eines Computers zum modernen Leben gab, zweifelt heute kaum einer daran. Computer sind in den Jahren kleiner und unsichtbarer geworden. Viele von denen werden von uns nicht mehr als solche wahrgenommen. Computer sind überall zu finden, in Fahrkarten- oder Parkscheinautomaten, in den Mobiltelefonen, Fernseher oder auch in den Kaffeeautomaten. Sogar die Plastikkarten, wie die Kreditkarten oder Krankenversichertenkarte, sind im Grunde Computer. Eingebettete Systeme (Embedded Systems) sind der Technologietrend des 21. Jahrhunderts.

> „Unter Embedded Systems (ES) versteht man Computersysteme, die in Geräten, Anlagen und Maschinen eingebettet sind und spezielle Anwendungen abarbeiten...
> ...Ein Embedded System hat genau definierte Aufgaben; es bildet soft- und hardwaremäßig eine funktionale Einheit, die nur diese definierten Aufgaben erfüllt..."[1]

Eingebettete Systeme ermöglichen mehr Einsatzgebiete für den Computer. Einer davon ist das Eigenheim. Smart Home oder Intelligente Wohnung einer der Trends der letzten Jahren, der auf eingebetteten Systemen aufbaut. Computer und Mikrocontroller übernehmen dort unterschiedlichste Aufgaben: von der Klimasteuerung bis zum Kaffeekochen ist alles möglich.

Auch das Living Place an der HAW-Hamburg beschäftigt sich mit den Themen Smart Home und Ambient Intelligence.

1.1. Zielsetzung

Das Ziel dieser Arbeit ist es einen generischen Smart Home Controller für das Living Place zu entwickeln. Der Controller soll eine Art Fernbedienung für alle schon vorhandenen, sowie

[1] http://www.itwissen.info/definition/lexikon/Embedded-System-ES-embedded-system.html, abgerufen am 27.07.2012

zukünftigen Installationen werden. Die zu entwickelnde Software soll ein Plug-in Host werden, der Plug-ins aufnimmt und dadurch die Steuerung neuer Komponenten des Living Place übernimmt. Es soll ein Plug-in-Konzept erarbeitet werden, der eine Plug-in-Entwicklung ohne besondere Programmierkenntnisse voraussetzt.

Um die Umsetzbarkeit demonstrieren zu können, soll ein Prototyp der Anwendung entwickelt werden, der grundlegende Funktionalität realisiert. Zudem sollen Plug-ins, für die im Living Place bereits vorhandene Komponente bzw. Installationen, erstellt werden. Das Zusammenspiel der Plug-ins mit dem Prototyp soll anhand eines Funktionstest demonstriert werden.

1.2. Gliederung der Arbeit

Die Arbeit besteht aus sechs Kapiteln. Die Einleitung gibt einen kurzen Überblick über das Themengebiet, beschreibt die Zielsetzung, sowie die Gliederung der Arbeit.

Das Kapitel Grundlagen, vermittelt das Basiswissen über die für die Arbeit wichtigen Themengebiete.

Im Kapitel Analyse wird die Problemstellung, sowie Rahmenbedienungen analysiert und Anforderungen formuliert, eine Basis für das Projekt gefunden und die Machbarkeitsstudie durchgeführt. Zuletzt wird das Kapitel im Fazit zusammengefasst.

Das Kapitel Design befasst sich mit der Gestaltung des User Interfaces und der Kommunikation. Ein Plug-in Modell wird erarbeitet, sowie über die Persistenz und Konsistenz der Daten diskutiert, sowie das Kapitel im Fazit zusammengefasst.

Im Kapitel Realisierung wird die entwickelte Software beschrieben, die Funktionsweise erklärt und Vorschläge zur Erweiterung und Verbesserung vorgestellt.

Zum Schluss wird die Arbeit noch einmal zusammengefasst, über die gesammelten Erfahrungen diskutiert und ein Ausblick in die Zukunft gegeben.

2. Grundlagen

In diesem Kapitel werden Grundlagen zu den für diese Arbeit relevanten Themen vermittelt, unter anderem zur Android™-Plattform, Smart Home und Living Place.

2.1. Android™

Android™ ist eine von Google und Open Handset Alliance entwickelte Softwareplattform für mobile Endgeräte, wie z.B Smartphones. Diese Softwareplattform besteht aus einem auf Linux basierendem Betriebssystem, einer Laufzeitumgebung[1], umfangreichen Bibliotheken, sowie fertigen Schlüsselapplikationen, wie bspw. Telefon- oder SMS-App. Die Abbildung 2.1 zeigt nochmals die wichtigsten Komponente einer Android™-Softwareplattform. Die Benutzung von Android™ ist kostenlos und große Teile der Plattform sind Open-Source. Die Applikationen der Drittanbieter sind mit Plattformapplicationen gleichberechtigt. Als Programmiersprache wird Java eingesetzt. Der Start von Android™ wurde offiziell im November 2007 angekündigt. Das erste mobile Gerät mit Android™ OS war das HTC bzw. T-Mobile G1, der am 22. Oktober 2008 in der USA und am 02. Februar 2009 in Deutschland auf den Markt gekommen ist.(Mosemann und Kose, 2009, Kapitel 1)

Zum Zeitpunkt der Erstellung dieser Arbeit ist die aktuelle Android™ Version 4.1, die den Namen „Jelly Bean“ trägt. Die Aktuelle API hat den Level 16 erreicht.[2]

[1] Dalvik VM
[2] http://developer.android.com/about/dashboards/index.html, abgerufen am 02.08.2012

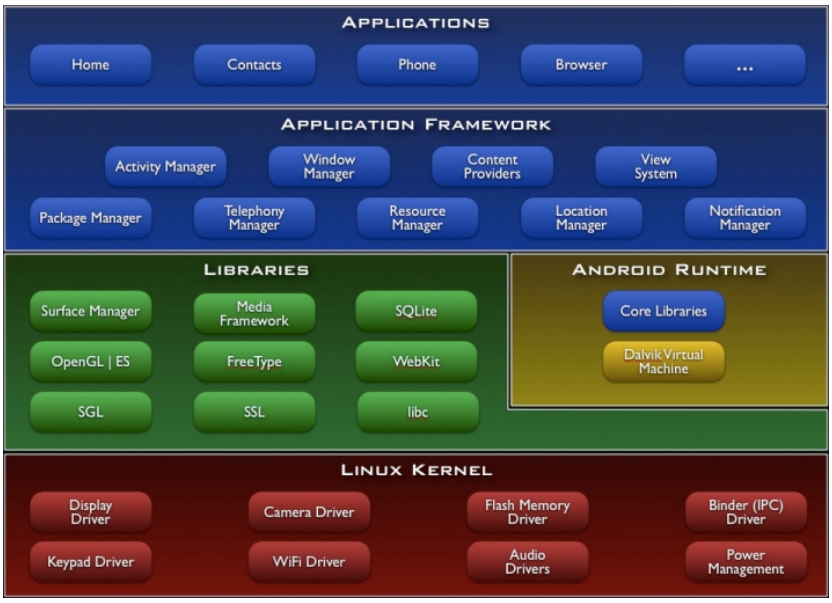


Abbildung 2.1.: Wichtigste Komponenten eines Android™ OS
Quelle: http://developer.android.com/images/system-architecture.jpg

2.1.1. Activities und deren Lebenszyklus

> „Aktivitäten (Unterklassen der *android.app.Activity*) stellen den sichtbaren Teil einer Android™-Applikation dar. Eine Aktivität repräsentiert eine Interaktion mit dem Benutzer, und oftmals kann man die Screens einer Android™-Applikation 1:1 auf Activity-Klassen abbilden [...] Sobald eine Aktivität mittels der Callback-Methode *onCreate()* vom System und meist vom Anwender durch direkte Interaktion gestartet wurde, durchläuft Sie einen Aktivitätenlebenszyklus, der durch mehrere Callback-Methoden gesteuert wird...“ (Haiges, 2011, Kap. 1.5, S.23)

Ein Aktivitätenlebenszyklus sieht, wie in der Abbildung 2.2 dargestellt aus. Jede Applikation durchläuft diesen Zyklus. Dabei ist aber zu bedenken, dass allein das Android™ System entscheidet, wann welche Aktivität mittels dieser Callback-Methoden in den Vordergrund oder Hintergrund verschoben, von einer anderen Aktivität abgelöst oder gar beendet wird. Aus diesem Grund ist es in der Android™-Programmierung üblich die grafischen Elemente einer Aktivität in der *onCreate()*-Methode neu zu erstellen bzw. wiederherzustellen und die *onSaveInstanceState()*-Methode zu benutzen um den Zustand der Aktivität zu speichern. Die Abbildung 2.3 stellt diesen Ablauf grafisch dar.

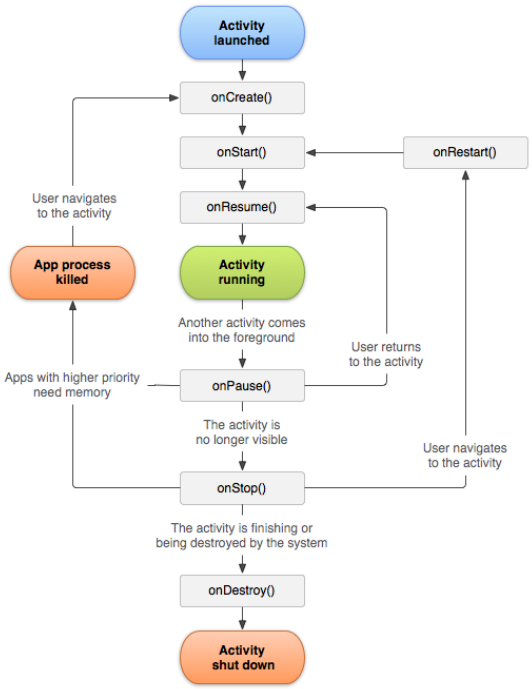


Abbildung 2.2.: Aktivitätenlebenszyklus
Quelle: http://developer.android.com/images/activity_lifecycle.png

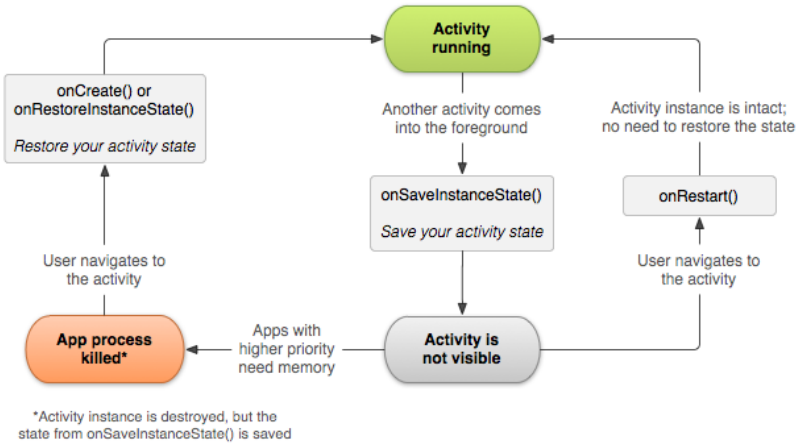


Abbildung 2.3.: Wiederherstellung des Zustandes in Android™
Quelle: http://developer.android.com/images/fundamentals/restore_instance.png

2.1.2. Tabs und Action Bar ab Android™ API Level 11

Bis API Level 11 war es für die Programmierung der Tabs üblich auf den Tab-Host und Tab-Widget Konzept zurückzugreifen. Dabei repräsentiert eine Aktivität einen Tab-Widget und wird vom Tab-Host aufgenommen und verwaltet. Seit API Level 11 ist dieses Konzept allerdings veraltet und sollte nicht mehr genutzt werden. Stattdessen gibt es ein neues Konzept der Action Bar. Dabei sollte jedes Tab nicht mehr einer Aktivität entsprechen, sondern einem Fragment und wird von der Action Bar aufgenommen und verwaltet. Action Bar ist mehr als nur ein Tab-Host. Diese verbindet mehrere Bedienelemente, wie Name und Logo einer Application, Tabbereich, Menübereich und Bereich für Action Buttons. Von API 11 bis API 13 war die Action Bar nur den Tablets vorbehalten, seit API 14 ist die Action Bar auch für Smartphones verfügbar. Die Abbildung 2.4 zeigt das alte Tab-Konzept, die Abbildung 2.5 stellt dagegen das neue Action Bar Konzept dar.

Abbildung 2.4.: Altes Tab Host Tab Widget Konzept
Quelle: http://developer.android.com/resources/tutorials/views/images/hello-tabwidget.png

Abbildung 2.5.: Das neue Action Bar Konzept
Quelle: http://developer.android.com/images/ui/actionbar.png

Eine typische Action Bar stellt mehr als nur einen Tab Host dar und besteht aus mehreren Bereichen (Komatineni und MacLean, 2012, Kapitel 10, Seite 283):

- **Title area**
  Dieser Bereich ist für die Anzeige des Programmnamens zuständig
- **Tabs area**
  Dieser Bereich beherbergt die Tabs
- **Action Icon area**
  In diesem Bereich befinden sich manche der Menüpunkte als Icon dargestellt
- **Menu Icon area**
  Hier wird das Programmmenü beherbergt

Das Konzept der Action Bar macht die Tab-Programmierung attraktiver und vielseitiger. Außerdem passt sich das Design dem Design des neuen Android™-Betriebssystems der Version 4.x.x an.

2.1.3. Fragments

> „Fragments are not widgets, like *Button* or *EditText*. Fragments are not containers, like *LinearLayout* or *RelativeLayout*. Fragments are not activities. Rather, fragments aggregate widgets and containers. Fragments then can be placed into activities—sometimes several fragments for one activity, sometimes one fragment per activity. And the reason for this is the variation in Android™ screen sizes.“ (Allen und Murphy, 2012, S.307)

Die Idee der Fragmente ist es die Darstellung auf unterschiedlichen Bildschirmen der Android™-Geräte zu ermöglichen. Vor allem sind die Unterschiede zwischen den Tablets und Smartphones, was die Bildschirmgröße angeht, sehr groß. Um den Programmieraufwand zu verkleinern und keinen doppelten Code für beide Gerätegruppen zu erzeugen, werden Fragmente eingesetzt. Die Abbildung 2.6 stellt nochmal die Verwendung von Fragments bildlich dar. Dabei beinhaltet die Aktivität eines Tablets zwei Fragmente und die des Smartphones lediglich einen. Die Fragmente selbst sind jedoch in beiden Fällen identisch.

Eine weitere Besonderheit, die die Fragmente mit sich bringen, ist die Tatsache, dass man jetzt Applikationen erstellen kann, die aus lediglich einer einzelnen Aktivität bestehen.

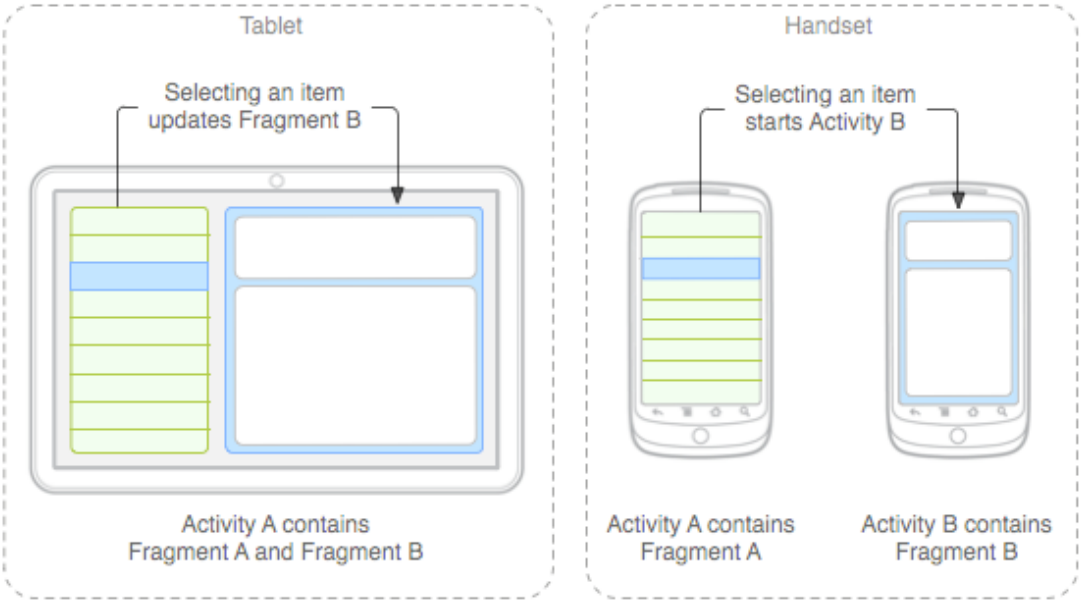


Abbildung 2.6.: Beispiel für die Verwendung von Fragments
Quelle: http://developer.android.com/images/fundamentals/fragments.png

„Fragments always run within the context of an existing activity. The life cycle of a fragment is similar to that of an activity, but with a few added callbacks that handle events such as attaching to the host activity." (Ostrander, 2012, Kapitel 5, Seite 154)

Die Abbildung 2.7 verdeutlicht nochmal die Ähnlichkeit, Abhängigkeit, sowie die Unterschiede zwischen dem Aktivitätslebenszyklus und dem Lebenszyklus der Fragmente.

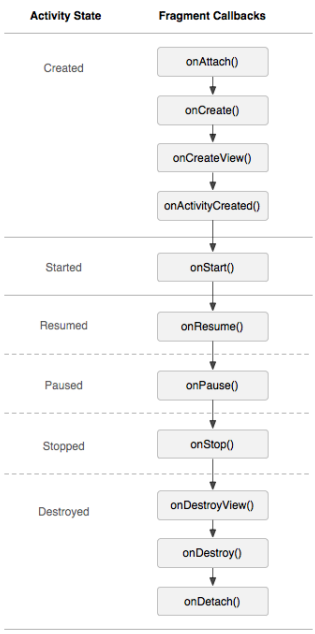


Abbildung 2.7.: Abhängigkeit der Lebenszyklen zwischen Aktivitäten und Fragmenten
Quelle: http://developer.android.com/images/activity_fragment_lifecycle.png

Die Fragmente sind ein wichtiges Bestandteil des Action Bar Konzepts und sollten in diesem Zusammenhang stets verwendet werden. Die Universalität des Fragment-Codes stellt einen weiteren Vorteil bei der Nutzung der Fragmente dar.

2.2. Smart Home

Schon im Jahr 1988 verwendete Mark Weiser erstmals den Begriff „Ubiquitous Computing“. In dem Artikel „The Computer for the 21st Century“ (Weiser, 1991) beschreibt Weiser seine Vision von der „Allgegenwärtigkeit der Computer im 21. Jahrhundert“. Weisers Vision beschreibt das Leben eines Menschen umgeben von Computern, die sich der Umgebung anpassen und sich in das Leben eines Menschen integrieren, um dieses zu erleichtern und zu unterstützen.

> „Ubiquitous computing names the third wave in computing, just now beginning. First were mainframes, each shared by lots of people. Now we are in the personal computing era, person and machine staring uneasily at each other across the desktop. Next comes ubiquitous computing, or the age of calm technology, when technology recedes into the background of our lives.“ –Mark Weiser[3]

[3] http://www-sul.stanford.edu/weiser/Ubiq.html, abgerufen am 05.06.2012

Heute, nach über 20 Jahren, ist die Vision von Weiser Realität geworden. Begriffe, wie Smartphone, Smart-TV oder Smart Home prägen unseren Alltag. Dabei bedeutet der neue Wortzusatz „Smart“-clever[4]. Durch das Integrieren von Computern in die uns bekannte Gegenstände wird diesen eine gewisse Intelligenz gewährt. Solche Geräte sind dann in der Lage selbständig zu agieren bzw. auf äußere Einflüsse zu reagieren.

> „Das Smart Home ist ein privat genutztes Heim (z. B. Eigenheim, Mietwohnung), in dem die zahlreichen Geräte der Hausautomation (wie Heizung, Beleuchtung, Belüftung), Haushaltstechnik (wie z. B. Kühlschrank, Waschmaschine), Konsumelektronik und Kommunikationseinrichtungen zu intelligenten Gegenständen werden, die sich an den Bedürfnissen der Bewohner orientieren. Durch Vernetzung dieser Gegenstände untereinander können neue Assistenzfunktionen und Dienste zum Nutzen des Bewohners bereitgestellt werden und einen Mehrwert generieren, der über den einzelnen Nutzen der im Haus vorhandenen Anwendungen hinausgeht“ (Strese u. a., 2010, Seite 8)

Die Abbildung 2.8 stellt die wichtigsten Einheiten eines Smart Homes grafisch dar.

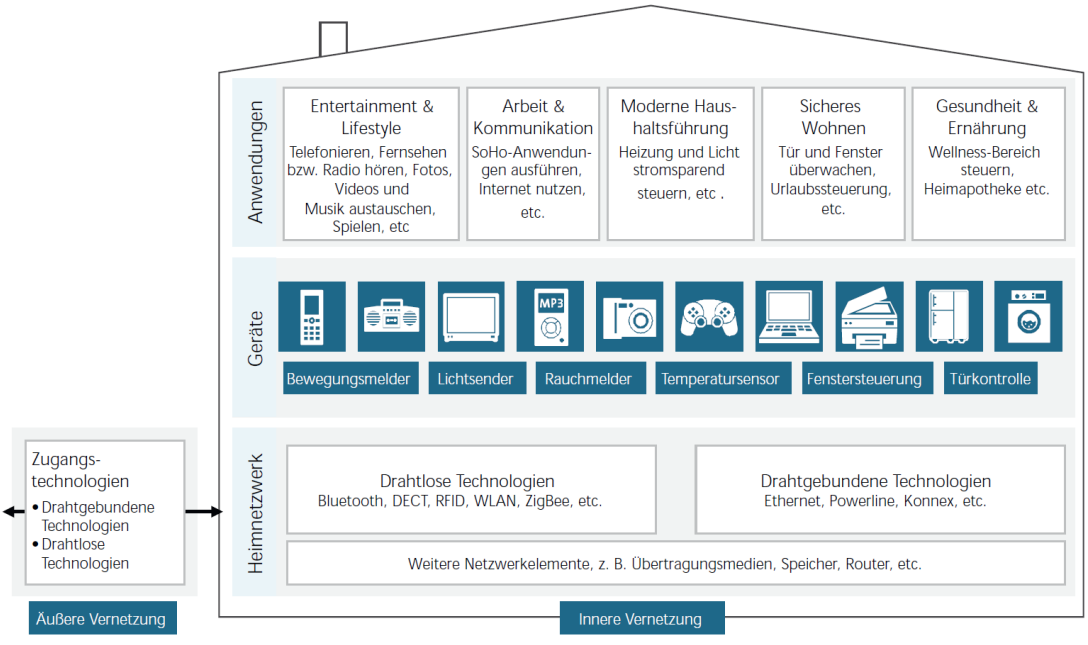


Abbildung 2.8.: Graphische Darstellung eines Smart Homes
Quelle: BITKOM AK Digital Home: Leitfaden zur Heimvernetzung, 2009; S. 5

[4] http://www.duden.de/rechtschreibung/smart, abgerufen am 05.06.2012

Wie man aus der Grafik 2.8 erkennt, baut ein Smart Home auf einem Heimnetzwerk auf. Das Netzwerk kann dabei unterschiedliche Übertragungsmedien und Protokolle verwenden. Des Weiteren benötigt jedes Smart Home die smarten Geräte, die miteinander mittels Netzwerk verbunden werden. Zuletzt sind es die Anwendungen, die die Intelligenz und Funktionalität eines Smart Homes ausmachen.

Der Trend des Smart Homes entwickelt sich im Wesentlichen in vier Marktsegmenten (Strese u. a., 2010, Seite 13):

- Energiemanagement
  Die Richtlinien des EU-Parlaments fordern einen effizienten Umgang mit Energie, sowie eine genau Erfassung des Verbrauchs.

- Ambient Assisted Living
  Die demografische Situation in Deutschland (Tendenz: steigender Durchschnittsalter) fordert Lösungen, die ältere Menschen im Alltag unterstützen werden.

- Sicherheit
  Das steigende Bedürfnis nach Sicherheit, sowie Vorschriften seitens der Regierung erfordern die Einbindung von Warnsystemen, wie Rauchwarnmelder oder Notrufsystemen, in die Smart Home Umgebung.

- Komfort
  Unverändert bleibt das menschliche Verlangen nach Komfort.

Die steigende Nachfrage fordert bekanntlich die Entwicklung neuer Technologien, sowie technischer Geräte, die Smart Home als Einsatzort haben. Die Industrie reagiert bereits mit neuen Geräten auf den steigenden Trend. Die Abbildung 2.9 zeigt einen Smart-Kühlschrank „LG DIOS“ bei der Präsentation. Der intelligente Kühlschrank verfügt über die „Smart Grid“ Technologie und ist in der Lage den eigenen Stromverbrauch den Bedürfnissen anzupassen.

Abbildung 2.9.: LG DIOS Smart Grid-Ready bei der Präsentation
Quelle: http://en.akihabaranews.com/91586/environment/lg-introduces-its-first-smart-grid-ready-refrigerator-the-dios, abgerufen am 10.08.2012

2.2.1. Entwicklung und Trend

Die Komplettlösungen für das Smart Home werden bereits seit vielen Jahren von mehreren Initiativen in paralleler Arbeit entwickelt. Eine gemeinsame Strategie oder ein Gremium, der die Entwicklungsrichtung angibt oder Standards und Richtlinien definiert, fehlt bislang. Dies hat zur Folge, dass viele „Insellösungen“ existieren, die untereinander meistens nicht kompatibel sind und oft nicht alle Bereiche abdecken. Dies bremst zwar die Entwicklung ab, ein steigender Trend ist jedoch in den letzten Jahren auf dem Smart Home Markt zu beobachten. Die Entwicklung des Smart Homes spielt sich in drei Kernbereichen ab:

- Konsumelektronik
  TV, PC, DVD, Telefon, Personenwaage etc.
- Haushaltstechnik
  Kühlschrank, Herd, Mikrowelle, Waschmaschine etc.
- Hausautomation
  Beleuchtung, Heizung, Lüftung, Alarmanlage etc.

Heutzutage ist eine klare Trennung dieser Bereiche zu erkennen. Die Trendprognosen versprechen in der nahen Zukunft den Wegfall der Grenzen und eine Verschmelzung der Kernbereiche. Ein denkbares Beispiel dazu wäre, dass das Licht automatisch gedimmt wird, wenn man eine DVD anschauen möchte. Mittelfristig bleibt dieser Trend ebenfalls bestehen. Eine noch stärkere Verschmelzung der Kernbereiche findet statt. Smart Home wird dienstorientiert. Beispielsweise kann ein Dienst „DVD anschauen“ viel mehr als nur Fernsehgerät oder DVD-Player anschalten. Dieser kann dafür sorgen, dass die Telefonanlage die Anrufe auf den Anrufbeantworter weiterleitet, das Handy stumm bleibt, die Klimatisierung den Bedürfnissen angepasst wird und die Rollläden heruntergelassen werden. Gestartet wird der Dienst eventuell über eine definierte Geste. Die Mensch-Maschine-Kommunikation wird zu einem wichtigen Thema in einer Smart Home Umgebung, wobei die Schnittstellen, wie Sprache und Gestik verstärkt eingesetzt werden. Usability wird zum Schwerpunkt der Entwicklung.(Strese u. a., 2010)

2.3. Living Place

2.3.1. Beschreibung

Seit Januar 2009 existiert an der HAW das Living Place. Living Place ist ein Labor für angewandte Forschung in verschiedenen Bereichen der Ambient Intelligence und stellt ein ca. 140

m^2 großes Loft mit ca. 100 m^2 angrenzenden Labor und Kontrollräumen dar. Die Abbildung 2.10 zeigt einen Grundriss der Räumlichkeiten. Das Loft ist in folgende Bereiche aufgeteilt: Essbereich, Wohnbereich, Küche, Schlaf- und Arbeitsbereich, sowie ein separates Badezimmer. (von Luck u. a., 2010) Die Abbildung 2.11 stellt die Aufteilung des Lofts bildlich dar.

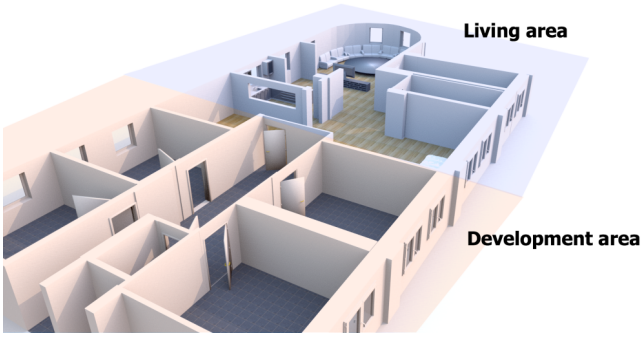


Abbildung 2.10.: Living Place Hamburg
Quelle: von Luck u. a. (2010)

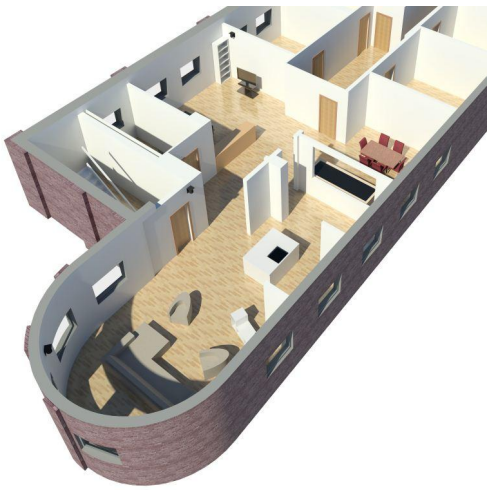


Abbildung 2.11.: Aufteilung des Lofts im Living Place Hamburg
Quelle: Living Place Hamburg

2.3.2. Ausstattung

An dieser Stelle wird ein Überblick über die bereits vorhanden Installationen des Living Place gegeben. Hier wird nicht die gesamte Hardwareaufstellung benannt, da diese ständig moder-

nisiert und erweitert wird, sondern nur für diese Arbeit relevanten Hardwareinstallationen erwähnt.

Licht

Die wohl bis jetzt aufwendigste Installation betrifft das Licht. Die Lichtinstallation wurde von der Schwedischen Firma „Ljusarkitektur“ vorgenommen und beinhaltet: „... unterschiedlichste Beleuchtungselemente wie zum Beispiel vertikale Beleuchtung, Deckenbeleuchtung, integrierte Beleuchtung in der Möblierung, jeweils mit Farbwahl (RGB LED) und Akzentbeleuchtung...“[5]. Die Lichtinstallation ermöglicht es die gesamte Wohnung in unterschiedlichste Farben einzufärben, um somit unterschiedliche Stimmungen an die Bewohner zu vermitteln, siehe Abbildung 2.12. Die Leuchten werden mittels netzwerkbasierten Pharos-Controllern gesteuert.

Abbildung 2.12.: Beispiel Beleuchtung
Quelle: http://livingplace.informatik.haw-hamburg.de/blog/?p=484

Gardinen und Rollläden

Gardinen und Rollläden sind ein Teil der Lichtinstallation und werden ebenfalls zur Erzeugung der Lichtstimmung genutzt. Gesteuert werden diese ebenfalls über die Pharos-Controller. Die Gardinen lassen sich einzeln ansteuern, die Rollläden jedoch lediglich alle gemeinsam.

[5] http://livingplace.informatik.haw-hamburg.de/blog/?p=484, abgerufen am 12.06.2012

Fenster und Heizung

Im gesamten Living Place ist es möglich sowohl jedes einzelne Fenster, als auch Fenstergruppen (Alle, Esszimmer, Küche, Wohnzimmer) auf bis zu 20 cm Weite zu öffnen. Die Ausnahme stellen die Fenster im Schlafbereich dar, da diese sich nicht öffnen lassen. Es ist ebenfalls möglich die Heizungsventile anzusteuern. Die Ansteuerung erfolgt gruppenweise: beide Ventile im Essbereich, vier Ventile im Wohnbereich jeweils paarweise links und rechts, ein Ventil im Bad, sowie zwei Ventile im Schlafbereich.

Weitere Installationen

Zu den weiteren Highlights des Living Place gehören mehrere Kameras und Mikrophone, die beispielsweise zum Suchen von Objekten eingesetzt werden. Ein drahtloses Sensornetzwerk, welches das kabellose Abfragen der Sensoren im Living Place ermöglicht, *Indoor Spatial Information Service*, mit dem man die Position im Living Place ermitteln kann, sowie ein intelligentes Bett sind weitere Bestandteile des Living Place.

2.3.3. Steuerung

Fast jede Installation setzt Controller ein, die auf einem niedrigen Abstraktionslevel, meist mittels Hersteller API, die Steuerung der einzelnen Komponenten übernehmen. Um für andere die Benutzung der Komponente zu erleichtern, werden Services programmiert, die auf einem Server laufen, Aufträge entgegennehmen und damit die Steuerung „von außen“ realisieren. Bei der Programmierung von Services für das Living Place gibt es keine Vorschriften, Standards, Protokolle oder Programmiersprachen, die man unbedingt benutzen muss oder an die man gebunden wäre. Jedem Entwickler ist es selbst überlassen, wie die Steuerung intern aussieht. Um die Kommunikation zwischen den einzelnen Komponenten dennoch möglich zu machen, hat man sich im Living Place auf asynchrone Kommunikation mittels Nachrichten geeinigt.

2.3.4. Kommunikation

Als Message Oriented Middleware (MOM) für die Kommunikation wird *Apache ActiveMQ*, eine Implementierung der *Apache Software Fundation*[6] eingesetzt. ActiveMQ ist ein quelloffener *Message Broker*, der eine vollständige Implementierung des JMS beinhaltet. Dies ermöglicht eine sehr einfache Benutzung in Java. Neben Java werden aber auch noch weitere Programmiersprachen von ActiveMQ unterstützt. Mittels „Cross Language Clients“, basierend auf dem

[6] http://www.apache.org/, abgerufen am 12.06.2012

OpenWire-Protokoll, werden Sprachen, wie *C, C++, C#* unterstützt. *C, Ruby, Perl, Python, PHP, ActionScriptFlash, Smalltalk*, basierend auf dem *Stomp-Protokoll*, finden ebenfalls Unterstützung in ActiveMQ. Zu den weiteren Vorteilen des ActiveMQ kann die Unterstützung der *REST-API*, sowie die Unterstützung von *Ajax* und *Spring* gezählt werden[7].

ActiveMQ implementiert zwei Arten der Kommunikation. Die erste Art basiert auf dem sog. „Topic“, der eine *publish and subscribe* Semantik besitzt. Dienste, die etwas mitzuteilen haben, veröffentlichen die Information mittels ActiveMQ in Form einer Nachricht. ActiveMQ wiederum verteilt diese an alle Interessenten, die sich für diese Information am Broker registriert haben. Es können mehrere oder gar keine sein, was einer $1 : n^*$ Beziehung entspricht. Die zweite Art basiert auf der sog. „Queue“, die eine *load balancer* Semantik besitzt. Hier veröffentlichen Dienste, die etwas mitzuteilen haben, ebenfalls ihre Information mittels ActiveMQ in Form einer Nachricht. In diesem Fall jedoch verteilt ActiveMQ die Nachricht nicht an alle Interessenten, sondern an einen Einzigen. Dies entspricht einer $1 : 1$ Beziehung[8].

Das Nachrichtenformat, der im Living Place eingesetzt wird kann weitestgehend frei gestaltet werden, jedoch befolgt es gewisse Regeln und Vorschriften. Es werden JSON-Nachrichten versendet. Jede Anfrage muss dabei folgende Pflichtfelder besitzen: Version und ID und jede Antwort zusätzlich noch das Feld Status. Sonst sind die Nachrichten frei im Rahmen der JSON-Notation gestaltbar (of Living Place, Nachrichtenformat)

2.4. Begriffsklärung

Im Laufe der Arbeit werden oft die Begriffe „Komponente“ oder „Installationen“ im Zusammenhang mit Living Place verwendet. Dabei sind die Akteure des Living Place gemeint. Es sind unterschiedliche Elemente, wie Licht, Heizung, Rollläden und Vorhänge oder Fenster, die angesteuert werden können.

[7] http://activemq.apache.org/, abgerufen am 12.06.2012
[8] http://activemq.apache.org/how-does-a-queue-compare-to-a-topic.html, abgerufen am 12.06.2012

3. Analyse

In diesem Kapitel werden Rahmenbedingungen erläutert, mögliche Szenarien durchgespielt und Anforderungen an die Hard- und Software gestellt. Es wird über die einzusetzende Plattform entschieden, sowie die Machbarkeit des Projekts analysiert. Vergleichbare Projekte werden in diesem Kapitel ebenfalls vorgestellt und am Ende das Kapitel im Fazit zusammengefasst.

3.1. Rahmenbedingungen

Wie bereits im Kapitel 2.3 erwähnt, sind im Living Place bereits Installationen vorhanden, die über das ActiveMQ mit ihrer Umgebung kommunizieren. Die zu entwickelnde Software muss also als Kommunikationsschnittstelle ActiveMQ benutzten. Des Weiteren darf die Hardware, sowie die Software der bereits bestehenden Komponenten nicht geändert werden, um Kompatibilitätsprobleme mit anderen Komponenten und Projekten des Living Place zu vermeiden.

3.2. Zielsetzung

Das Ziel dieser Arbeit ist die Entwicklung eines generischen Smart Home Controllers. Es soll eine geeignete Hard- sowie Softwarebasis für die Realisierung des Projekts gefunden werden. Es soll eine Software entwickelt werden, die eine Erweiterung der Funktionalität mittels Laden von Plug-ins unterstützt. Es soll ein Plug-in Modell eingesetzt werden, welches eine Erstellung der Plug-ins ohne Programmierkenntnisse ermöglicht. Die Software muss in der Lage sein die Benutzeroberfläche, abhängig von der Definition des Plug-ins, aufzubauen. Es sollen Plug-ins für bereits vorhandene Komponente des Living Place erstellt und die Funktionalität der Software anhand dieser demonstriert werden.

3.3. Szenarien und Anforderungen

3.3.1. Szenarien

Um die Anforderungen an die Software formulieren zu können, müssen folgende Szenarien untersucht werden:

- **Plug-in erstellen**
  Der User erstellt ein neues Plug-in
- **Plug-in laden**
  Das Programm lädt ein fertiges Plug-in
- **Oberflächenkomponente generieren**
  Das Programm interpretiert das Plug-in und erstellt neue Bedienelemente
- **Komponente steuern**
  Der User steuert mit den neuen Bedienelementen die Komponente

3.3.2. Anforderungen

Nach der Analyse der Szenarien wurden folgende funktionale, sowie nicht funktionale Anforderungen formuliert, die die Software erfüllen muss.

Funktionale Anforderungen

Funktionale Anforderungen beschreiben die Funktionalität, die das Programm implementieren muss, um seiner Bestimmung gerecht zu werden.

- Die Software muss eine Möglichkeit bieten die Plug-ins dafür erstellen zu können.
- Die Software muss eine Möglichkeit bieten die Plug-ins in das Programm einbinden zu können.
- Die Software muss im Stande sein die GUI anhand der Plug-ins aufzubauen.
- Die Software muss im Stande sein die Installationen des Living Place mithilfe der GUI zu steuern.

Nicht funktionale Anforderungen

Die nicht funktionalen Anforderungen beschreiben wie ein Programm funktionieren muss und sind ein Indikator für die Qualität der Software.

- Mobilität
  Die Steuerung muss von einem mobilen, nicht ortsgebundenem, Gerät erfolgen
- Aussehen und Design
  Das UI soll modern und zeitgemäß aussehen. Die nachträglich eingefügte Module müssen sich nahtlos der Software anpassen.
- Benutzbarkeit
  Die Steuerung muss schnell einsatzbereit und intuitiv zu bedienen sein
- Wirtschaftlichkeit
  Die Steuerung muss möglichst günstig sein
- Erweiterbarkeit
  Die Steuerung muss leicht zu erweitern sein, ohne Programmierkenntnisse vorauszusetzen.

3.4. Plattform

3.4.1. Hardware

Um Aussagen über die Hardware treffen zu können, müssen die funktionale, sowie nicht funktionalen Anforderungen geprüft und analysiert werden. Nach der Analyse ergeben sich folgende Kriterien, die von der Hardware erfüllt werden müssen:

- ein modernes Betriebssystem, welches das Programmieren auf einem hohen Abstraktionslevel ermöglicht
- drahtlose Kommunikationsmöglichkeit
- ein Bildschirm als Ausgabemedium
- intuitive Bedienmöglichkeit per Touchscreen
- handliche Größe
- möglichst lange Bereitschaft- bzw. Akkulaufzeit

Da eine der nicht funktionalen Anforderungen des Projekts die Wirtschaftlichkeit ist, wird eine Eigenentwicklung der Hardware dieser nicht genügen und an dieser Stelle ausgeschlossen. Daher gilt es herauszufinden, welche auf dem Markt verfügbaren Geräte für die gestellte Aufgabe am besten geeignet sind. Folgende Gerätetypen werden untersucht:

- Laptops bzw. Notebooks
- Smartphones
- Tablet PC's

Die Tabelle 3.1 vergleicht die Hardware bezüglich der für das Projekt relevanten Punkte. Die Bereitschaftszeit bzw. Akkulaufzeit lässt sich nicht objektiv vergleichen, da die Geräte für unterschiedliche Hauptaufgaben entwickelt worden sind und die Akkulaufzeit je nach Nutzung sehr variiert. Um dennoch ein ungefähres Bild von der Akkulaufzeit haben zu können, wurde hier auf die aktuellen Tests von www.chip.de zurückgegriffen[1]. Dabei wurde jeweils ein Gerät, der die höchste Bewertung in der Kategorie „Mobilität" bei Notebooks bis 13.3" und Tablets, sowie in der Kategorie „Telefon und Akku" der Smartphones herausgesucht und die längsten Akkulaufzeiten miteinander verglichen.

Der Vergleich zeigt, dass jeder der untersuchten Geräte eine Basis für einen generischen Smart Home Controller bietet, da alle diese den zuvor formulierten Anforderungen entsprechen. Die Entscheidung fällt jedoch zugunsten eines Tablets aus. Gegen den Einsatz eines Note- bzw. Netbooks spricht das Bedienkonzept. Die meisten dieser Geräte sind für die Bedienung mit der Tastatur und Maus ausgelegt und setzen entsprechende Betriebssysteme ein. Es ist zwar möglich ein Notebook mit Touchscreen mit einem für Touchscreens optimierten Betriebssystem zu bespielen, dies hat aber die Industrie mit der Einführung von Tablets bereits getan. Gegen den Einsatz von Smartphones spricht der kleine Bildschirm. Die Bildschirmgröße des größten Smartphones (Samsung Galaxy Note) ist teilweise kleiner als so manch eine Universalfernbedienung (vgl. Logitech Harmony 1100). Aus den obengenannten Gründen bieten Tablets die beste Mischung aus Displaygröße, Bedienkonzept, Mobilität, sowie der Vielfalt der Betriebssysteme.

[1] http://www.chip.de/bestenlisten/Bestenliste-Notebooks-bis-13-3-Zoll–index/index/id/879/ - Notebooks bis 13,3",
http://www.chip.de/bestenlisten/Bestenliste-Handys–index/index/id/900/- Smartphones,
http://www.chip.de/bestenlisten/Bestenliste-Tablets–index/index/id/970/ - Tablets,
abgerufen am 20.06.2012

Eigenschaft	Note- bzw. Netbook	Smartphone	Tablet PC
Betriebssystem	Windows OSX Linux Android	iOS Android SymbianOS Windows Mobile BlackBerry OS WebOS Linux(Maemo)	Android iOS Windows WebOS PlayBook OS
Bedienung	primär: Tastatur und Maus zusätzlich: Touchscreen	primär: Touchscreen selten: Tastatur	primär: Touchscreen selten: Tastatur
Kommunikation	LAN WLAN Bluetooth UMTS	WLAN Bluetooth UMTS	WLAN Bluetooth UMTS
Bildschirmgröße	ab 7“ (Asus eee pc 701) bis 18,4“ (Asus K93SM)	ab 2,6“ (HP Veer) bis 5,3“ (Samsung Note)	ab 5“ (Dell Streak) bis 11,6“ (Neofonie WeTab)
Akkulaufzeit	Toshiba Portégé R830-10V Bewertung: 100% Laufzeit: Office 15:24 h	Samsung Galaxy S Plus i9001 Bewertung: 100% Laufzeit: Online 9:25 h	Samsung Galaxy Tab 7.0 Plus N Bewertung: 100% Laufzeit: Websurfen 7:01 h

Tabelle 3.1.: Vergleich der für den Controller relevanter Hardwareeigenschaften

3.4.2. Betriebssystem

Da die Hardwareentscheidung auf das Tablet gefallen ist, gilt es zu analysieren, welcher der für Tablet PC verfügbaren Betriebssysteme als Plattform für Smart Home Controller am besten geeignet sind. Wie man aus der Abbildung 3.1 unschwer erkennen kann, wird der Markt der Mobilen Betriebssysteme unter Android™ von Google und iOS von Apple aufgeteilt. Die Gerätevielfalt, das Open Source Aspekt, sowie die persönliche Überzeugung begünstigen die Entscheidung für Android™. Apples strenge und kostenpflichtige Entwicklungspolitik, sowie die Gerätebeschränkung auf iPad erleichtern ebenfalls die Entscheidung zugunsten von Android™ OS.

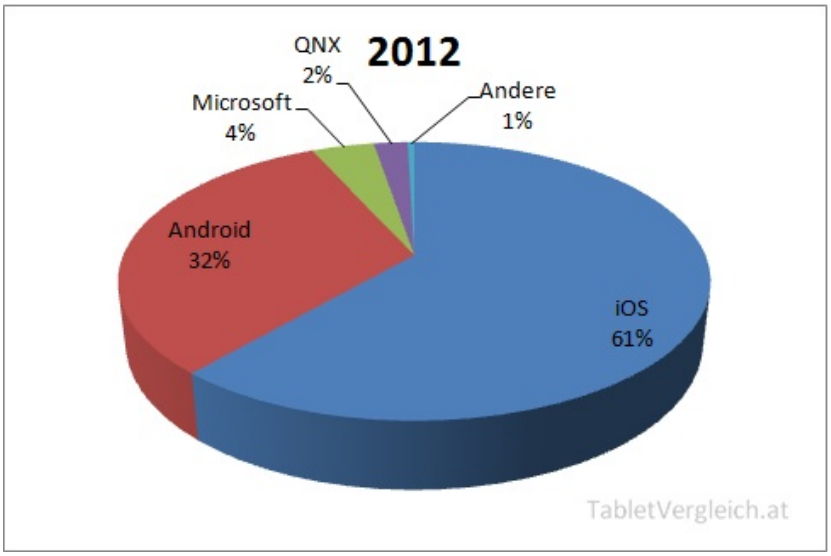


Abbildung 3.1.: Überblick über Marktanteile Mobiler Betriebssysteme für Tablets
Quelle: http://www.tabletvergleich.at/tablet-pc-marktanteile-bis-2016-marktforschungsinstitut-gartner, abgerufen am 20.06.2012

Android™ Version

Da die Entscheidung zugunsten des Android™ von Google fiel, welches es in unterschiedlichen Versionen gibt, gilt es eine Entscheidung für eine der Versionen zu treffen. Die Abbildung 3.2 zeigt die Aufteilung der Android™ Versionen auf. Eine Dominanz von „Gingerbread“ (Android™ 2.3.x) ist klar zu erkennen, jedoch bleibt diese Version bis auf ein paar Ausnahmen die Domäne der Smartphones. Die Version 3.x des Betriebssystems dagegen setzt die Verwendung auf Tablet PC's voraus. Mit „Ice Cream Sandwich“ (Android™ 4.0.x) wurde erstmals ein Betriebssystem herausgebracht, das sowohl für Smartphones, als auch für Tablets optimiert ist. Diese Version bringt viele Neuerungen in den Bedienkonzepten, wie z.B. Action Bar (siehe Kapitel 2.1.2) oder Navigation Bar[2], erstmals auch für die Smartphones. Da ein eventueller Einsatz der Controller Software zu einem späteren Zeitpunkt auf einem Smartphone nicht ausgeschlossen werden kann, bietet sich der Einsatz einer Version, die auf beiden Gerätetypen lauffähig ist, also Android™ 4.0.x „Ice Cream Sandwich“ an. Die Abwärtskompatibilität von Android™ sollte theoretisch auch den Einsatz unter „Jelly Bean“ (Android™ 4.1.x) ermöglichen.

[2] http://developer.android.com/design/patterns/new-4-0.html, abgerufen am 20.06.2012

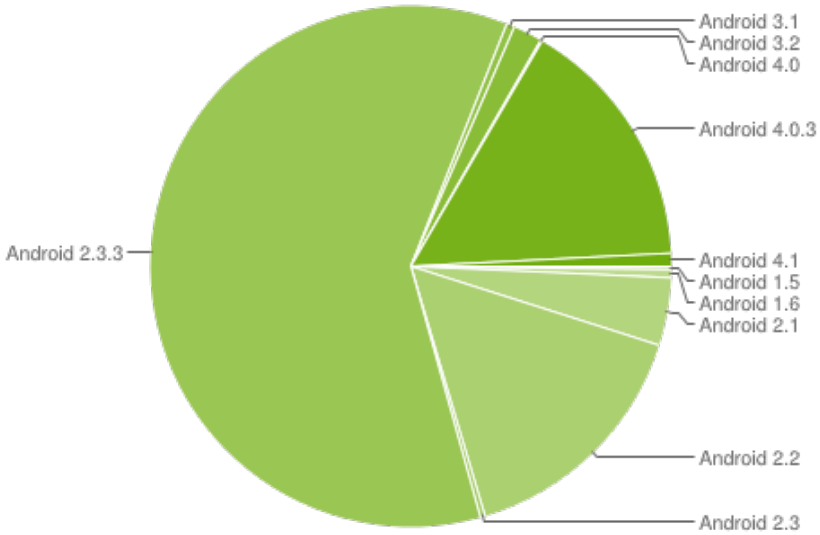


Abbildung 3.2.: Überblick über Benutzung unterschiedlicher Android™ Versionen
Quelle: http://developer.android.com/resources/dashboard/platform-versions.html, abgerufen am 06.08.2012

3.5. Machbarkeitsstudie

An dieser Stelle gilt es zu überprüfen, ob es überhaupt möglich ist das Projekt zu realisieren. Hardware- und Softwaregrundlage bietet ein Tablet PC mit dem Betriebssystem Android™ Ice Cream Sandwich. Hardwareseitig sind alle für die Realisierung benötigten Komponenten, wie WLAN und Touchscreen vorhanden. Softwareseitig bietet das Android™ ein sehr mächtiges SDK, ein sehr ausgereiftes Betriebssystem auf Linux-Basis und Unterstützung der Programmiersprache Java, sodass auch softwareseitig einem erfolgreichen Projektabschluss nichts im Wege steht.
Dennoch gibt es zwei „Problembereiche“, die man an dieser Stelle ansprechen sollte:

- Zum Einen ist die Funktionalität der Plug-ins, durch die Controller-API beschränkt.
  Da laut der Rahmenbedienungen die Änderung von Hard- bzw. Software der vorhandenen Komponenten nicht gestattet ist und die Kommunikation über ActiveMQ stattfinden soll, bietet sich lediglich die Möglichkeit an die Komponenten über die vorhandene API mittels Nachrichten an ActiveMQ zu steuern.

- Zum Anderen stellt die zwingende Kommunikation mit dem ActiveMQ ein Problem dar. Obwohl Android SDK zum größten Teil dem Java JDK entspricht, fehlen dem Android SDK manche Pakete, wie z.B. `java.awt` oder `java.swing` (Becker und Pant, 2009,

Kapitel 21.1), auch das Paket `javax.jms`, welches Bestandteil des JavaEE ist, ist nicht dabei. Dies stellt ein Problem dar, da dieses Paket für die direkte Kommunikation mit dem ActiveMQ unbedingt notwendig ist.

Auf die Lösung dieser Probleme wird im Kapitel 4 näher eingegangen.

3.6. Vergleichbare Projekte

Die Recherchen ergaben, dass es mittlerweile fertige Smart Home Lösungen auf dem Markt gibt, die unter anderem Tablets oder Smartphones als einen der Controller einsetzen.

> „**Intuitive Bedienbarkeit von Daheim und unterwegs**
> Einfachste Bedienung über PC oder Smartphone durch selbsterklärende, graphische Benutzeroberfläche. Per Internet und Smartphone Zugriff auch aus der Ferne“[3]

So lautet einer der Werbeslogans des „RWE SmartHome“, einer der bekanntesten Anbieter auf dem deutschen Markt für nachträgliche Vernetzung des Hauses. Folgendes Zitat aus der RWE SmartHome Werbung erklärt am besten die Funktionsweise des Produkts:

> „RWE SmartHome ist eine Produktfamilie intelligenter Geräte, die Sie ohne technisches Vorwissen mit minimalem Zeitaufwand in Ihren Wohnräumen anbringen können. Ein hausinternes Funknetzwerk verbindet Haushaltsgeräte Ihrer Wahl mit einer zentralen Steuereinheit – die RWE SmartHome Zentrale – und ermöglicht darüber hinaus eine intelligente Heizungssteuerung.“[4]

Weitere Anbieter für Hausautomation sind z.B. EnOcean[5], HomeEasy[6] oder Loxone[7]. Alle diese Anbieter bieten Smartphone oder Tablet PC Apps als eine Art der Steuerung an. Obwohl diese Anbieter Hardware unterschiedlicher Hersteller verbauen, ist die Funktionsweise immer gleich. Die Abbildung 3.3 stellt eine Bedienoberfläche für die in der Abbildung 3.4 dargestellten Controller dar. Die Abbildung 3.5 gibt den Überblick über die Architektur. Die Abbildungen 3.3 bis 3.5 stellen ein Produkt der Firma „Enexoma AG“[8] dar, die auf dem Gebiet smarter Innovationen tätig ist.

[3] http://www.rwe-smarthome.de/web/cms/de/457156/smarthome/informieren/was-ist-rwe-smarthome/, abgerufen am 15.07.2012
[4] http://www.rwe-smarthome.de/web/cms/de/457156/smarthome/informieren/was-ist-rwe-smarthome/, abgerufen am 15.07.2012
[5] http://www.enocean.com/de/home/, abgerufen am 15.07.2012
[6] http://www.homeeasy.eu/german/home.php, abgerufen am 15.07.2012
[7] http://www.loxone.com/, abgerufen am 15.07.2012
[8] http://www.enexoma.de/, abgerufen am 13.08.2012

Abbildung 3.3.: Bedienoberfläche
Quelle:http://enexoma.de/newsletter/2011/2011-03-22/img/smart-home-web-iPhone.png, abgerufen am 13.08.2012

Abbildung 3.4.: Mögliche Controller
Quelle:http://enexoma.de/newsletter/2011/2011-03-22/img/smart-home-leiste.png, abgerufen am 13.08.2012

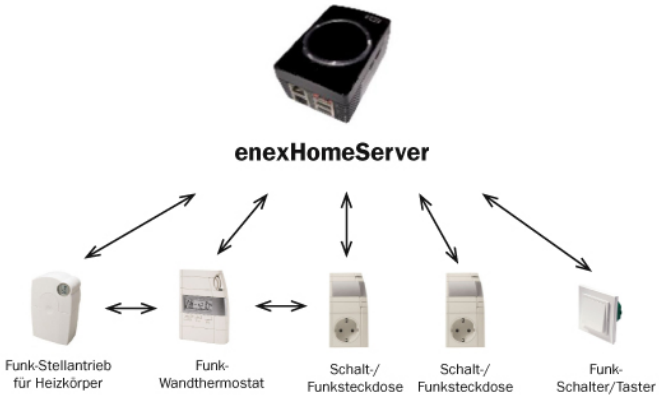


Abbildung 3.5.: Akteure und Server
Quelle:http://enexoma.de/newsletter/2011/2011-03-22/img/smart-home-enexhomeserver.jpg, abgerufen am 13.08.2012

Es gibt Endgeräte (Lampe, Klimaanlage, Heizung), zu den Endgeräten gibt es Akteure, wie z.B. Schalter, Dimmer, Steckdosen, Thermostate usw. Diese Akteure werden entweder per Funk, Ethernet oder eigenes Verbindung- bzw. BUS-System mit einer Steuereinheit/Server verbunden. Die Controller (Computer, Smartphone oder Tablet PC) senden Steuerbefehle an die Steuereinheit und diese wiederum steuert die Akteure an und somit die Endgeräte. Diese Funktionsweise stellt ein klassisches Server-Client-System dar, welches es von dem hier zu entwickelnden System konzeptuell sehr unterscheidet. Durch das Wegfallen des klassischen Servers, kann hier keine standardisierte Kommunikation zwischen Client und Server stattfinden, sondern eine Lösung gefunden werden muss, die es möglich macht unterschiedliche Kommunikationsformate mit unterschiedlichen Services mittels ActiveMQ zu verbinden.

Ein weiteres Produkt, der an dieser Stelle vorgestellt werden sollte, ist „Android@Home“.

> „Eines der neuen Projekte zur Erweiterung des Google-Betriebssystems nennt sich Android@Home. Dabei handelt es sich um eine Heimsteuerung, mit der etwa die Temperatur geregelt, das Licht an- und ausgeschaltet oder Musik bedient werden kann. Im Zentrum des Google-Betriebssystems für das eigene Zuhause steht dabei das jeweilige Android-Gerät, also etwa ein Tablet, über das sich Heimanwendungen gezielt kontrollieren lassen.“ (Zollondz, 2011)

Bei dem Projekt handelt es sich um eine Demonstration des ADK[9] (Android Accessory Development Kit). Interessant wird es seit ADK 2012, welches für die Verwendung mit der Arduino-Plattform[10] ausgelegt ist.[11] Diese Kombination macht es möglich selbständig komplexe Steuerungen für Hausautomation kostengünstig und auf Open Source Basis zu entwickeln. Die Abbildung 3.6 zeigt die Verwendung des Android™ Smartphones mit dem Arduino Entwicklungsboard und der ADK Demo-App.

Trotz der äußerlichen Ähnlichkeit der hier vorgestellten Projekte mit dem Schwerpunkt dieser Arbeit, gibt es zwischen diesen gravierende konzeptuelle Unterschiede.

[9] http://developer.android.com/tools/adk/adk2.html, abgerufen am 15.07.2012
[10] http://arduino.cc/, abgerufen am 15.07.2012
[11] http://developer.android.com/tools/adk/adk2.html, abgerufen am 15.07.2012

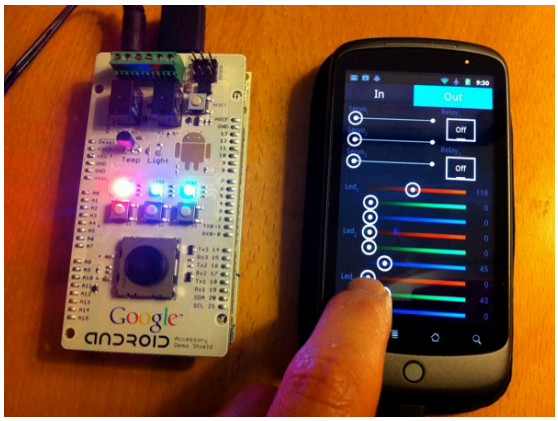


Abbildung 3.6.: ADK Demo-App auf Android™ Smartphone und Arduino Entwicklerboard
Quelle: http://makezineblog.files.wordpress.com/2011/06/demokitoutput.jpg, abgerufen am 13.08.2012

3.7. Fazit

Nach der durchgeführten Analyse wurden die Anforderungen an die zu entwickelnde Software gestellt, sowie eine Entscheidung bezüglich der einzusetzender Hardware getroffen. Die Hardwarebasis wird ein Tablet PC mit Android™ OS Ice Cream Sandwisch bieten. Für die Entwicklung des Prototypen wird ein 10“ Tablet PC: „Asus Eee Pad Transformer TF-101[12]“ (siehe Abbildung 3.7) mit Android™ 4.0.3 verwendet.

Abbildung 3.7.: Asus Eee Pad Transformer TF-101
Quelle: http://www.sparblog.com/wp-content/uploads/2011/10/Asus-EeePad-Transformer.jpg, abgerufen am 06.08.2012

[12] http://www.asus.de/Eee/Eee_Pad/Eee_Pad_Transformer_TF101/, abgerufen am 05.07.2012

4. Design

In diesem Kapitel werden grundsätzliche Designentscheidungen formuliert und getroffen. Es werden unterschiedliche Alternativen analysiert und zum Schluss das Design zusammengefasst.

4.1. UI Gestaltung

Für die Gestaltung des User Interfaces gelten keine Vorschriften, an die man sich unbedingt halten muss. Das User Interface ist also frei gestaltbar, jedoch muss dieses möglichst bequem zu bedienen sein. Am Beispiel vieler Programme (z.B Windows Explorer), deren Bedienkonzept sich über Jahre in der Computerwelt etabliert hat, wurde entschieden die sogenannte „Exploreransicht“ zu verwenden. Dafür wurde der Hauptbildschirm in zwei Teile (ca. $\frac{1}{3}$ und $\frac{2}{3}$) vertikal aufgeteilt. Der linke Teil des Bildschirms beherbergt die räumliche Navigation während der rechte Teil für die Darstellung der Bedienelemente verantwortlich ist. Zwischen den einzelnen Plug-ins soll mittels Tabs umgeschaltet werden, die ihren Platz im oberen Bereich des Bildschirms finden. Die Abbildung 4.1 skizziert die Bildschirmaufteilung.

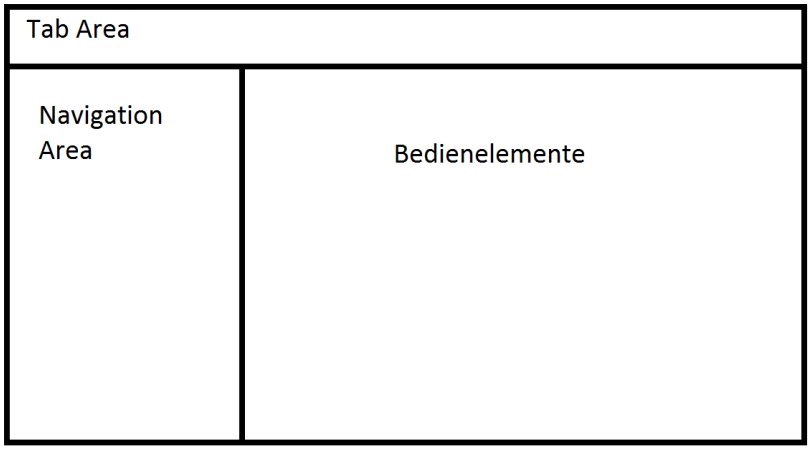


Abbildung 4.1.: UI-Aufteilung, Skizze

Folgendes Verhalten des Interfaces wäre denkbar:

1. Action: User wählt ein Plug-in durch Wählen des entsprechenden Tabs
   Reaction: Eine entsprechende Navigationsliste wird aufgebaut und angezeigt
   Bedienelemente des Root-Knotens der Navigationsliste werden, falls vorhanden, angezeigt

2. Action: User wählt einen Ort/Eintrag aus der Navigationsliste aus
   Reaction: Der gewählte Knoten der Navigationsliste wird zum Root-Knoten und seine Kindesknoten, falls vorhanden, aufgeklappt
   Bedienelemente des gewählten Knotens werden angezeigt

3. Action: User betätigt Bedienelement
   Reaction: Bedienelement ändert sein Zustand entsprechend der Action und löst ein Ereignis aus

4. Action: User Betätigt den Zurückbutton
   Reaction: Elternknoten des aktuellen Knotens, falls vorhanden, wird zum Root-Knoten und seine Kindesknoten aufgeklappt
   Bedienelemente des aktuellen Root-Knotens werden, falls vorhanden, angezeigt.

4.2. Bedienkonzept

Neue Technologien erfordern oft neue Bedienkonzepte. Die Mensch-Computer-Interaktion bildet deshalb einen eigenen wissenschaftlichen Zweig.

> „Human-computer interaction is a discipline concerned with the design, evaluation and implementation of interactive computing systems for human use and with the study of major phenomena surrounding them."(Hewett u. a., 2007, Kapitel 2.1)

Dabei beteiligen sich an diesem Bereich nicht nur Wissenschaftler aus der Informatik, sondern auch Wissenschaftler aus den Bereichen Psychologie und Verhaltensforschung.

Ein sehr verbreitetes Interaktionparadigma auf dem Bereich Ubiquitos Computing, zu dem auch Smart Homes gezählt werden, ist die unsichtbare Steuerung. Es wird nach Möglichkeit keine gesonderte Hardware zur Steuerung eingesetzt, sondern die Umgebung reagiert auf das menschliche Verhalten und die äußeren Einflüsse.(Hußmann, 2006, Kapitel 6.2) Die Idee dahinter ist nicht nur die Computer unsichtbar zu machen, sondern auch deren Steuerung, indem die Mensch-Computer-Interaktion so nah wie nur möglich an die Mensch-Mensch-Interaktion angelehnt wird. Dazu werden die menschlichen Kommunikationsformen, wie

Gestik, Sprache oder Handlungen verwendet. Dabei werden unterschiedlichste Medien zur dieser Erkennung eingesetzt. Nebst Kameras und Mikrofonen, werden Sensoren (Infrarot, Ultraschall usw.), Gegenstände oder Bekleidung verwendet. Auch an der HAW-Hamburg wird auf dem Gebiet Gestenerkennung geforscht. Die Masterarbeit von Arne Bernin „Einsatz von 3D-Kameras zur Interpretation von räumlichen Gesten im Smart Home Kontext“ (Bernin, 2011) beschäftigt sich mit dem Einsatz von 3D-Kameras zur Gestenerkennung im Kontext des Smart Homes.

Jedoch eignen sich nicht nur interpersonelle Kommunikationsformen für die Mensch-Computer-Interaktion. Computersteuerung mittels Gedanken ist bereits möglich. Die ersten Produkte für die breite Masse sind ebenfalls schon vorhanden. Die Abbildung 4.2 zeigt einen Neuroheadset der Firma EMOTIV[1].

Abbildung 4.2.: Neuroheadset der Firma EMOTIV
Quelle: http://www.emotiv.com/upload/iblock/6ac/header_set.gif, abgerufen am 16.08.2012

Es gibt auch weniger futuristische Interaktionsmöglichkeiten mit dem Computer, die den Menschen sehr vertraut sind. Dazu gehört unter anderem das Desktop Interaktionsparadigma. Dabei befindet sich der Mensch direkt vor einem Computerinterface, der meist in Form eines Bildschirms mit einer Benutzeroberfläche repräsentiert wird.(Hußmann, 2006, Kapitel 6.2) Die Kommunikation kann dabei auf unterschiedliche Art und Weise stattfinden, wie z.B. mittels Maus und Tastatur, per Touchscreen oder Kamera und Mikrofon. Eine weitere Interaktionsmöglichkeit stellen herkömmliche Fernbedienungen dar. Eine Symbiose aus der Benutzeroberfläche

[1] http://www.emotiv.com, abgerufen am 14.08.2012

einer Software und der Mobilität, sowie dem Konzept einer Fernbedienung, bieten neuartige Steuerungsmöglichkeiten. Diese bieten die Möglichkeit komplexe Steuerungen mit der „Usability“ einer herkömmlichen Fernbedienung zu erstellen. Solche Lösungen werden sehr oft in der Smart Home Umgebung verwendet. Eine Hardwarebasis bieten dabei meist portable Computer, wie Tablets oder Smartphones oder andere Geräte, die eine Touchbedienung unterstützen. Auch in dieser Arbeit wird das Desktopparadigma verfolgt und auf die Bedienung mittels Touchscreen gesetzt.

4.3. Kommunikation

Wie bereits im Kapitel 2.3.4 beschrieben läuft die Kommunikation im Living Place über ActiveMQ, der eine JMS-Implementierung ist. Unter Android™ wird in Java (siehe Kapitel 2.1) programmiert. Obwohl Android™ SDK zum größten Teil dem Java JDK entspricht, fehlen dem Android™ SDK manche Pakete, wie z.B. `java.awt` oder `java.swing` (Becker und Pant, 2009, Kapitel 21.1). Auch das Paket `javax.jms`, welches Bestandteil des JavaEE ist, ist nicht dabei. Dies stellt ein Problem dar, da dieses Paket für die direkte Kommunikation mit dem ActiveMQ unbedingt notwendig ist. Das Problem lässt sich aber umgehen, indem man z.B. einen Proxy einsetzt, der über eine Socket-Verbindung mit dem Android™ Tablet kommuniziert und die Anfragen an das ActiveMQ weiterreicht. Ein solches Proxy wurde bereits erstellt und wird im Living Place genutzt. Die Abbildung 4.3 soll die Funktionsweise nochmals verdeutlichen. Für die weitere Implementierung wird der Android™ Publisher verwendet, eine Bibliothek, die das komplette Verbindungsaufbau zum Android™ Proxy übernimmt. Zur ausführlichen Information über Android™ Proxy und Android™ Publisher sei an dieser Stelle auf die Bachelorarbeit von Sven Boris Bornemann verwiesen (Bornemann, 2011, Kapitel 5.1.3).

Alternativ könnte man für die Kommunikation das „simple Java Stomp-API“ verwenden, welches seit Version 5.2 im ActiveMQ zur Verfügung steht, jedoch nur zur Testzwecken genutzt werden sollte. [2]

[2] http://activemq.apache.org/stomp.html#Stomp-JavaAPI, abgerufen am 05.07.2012

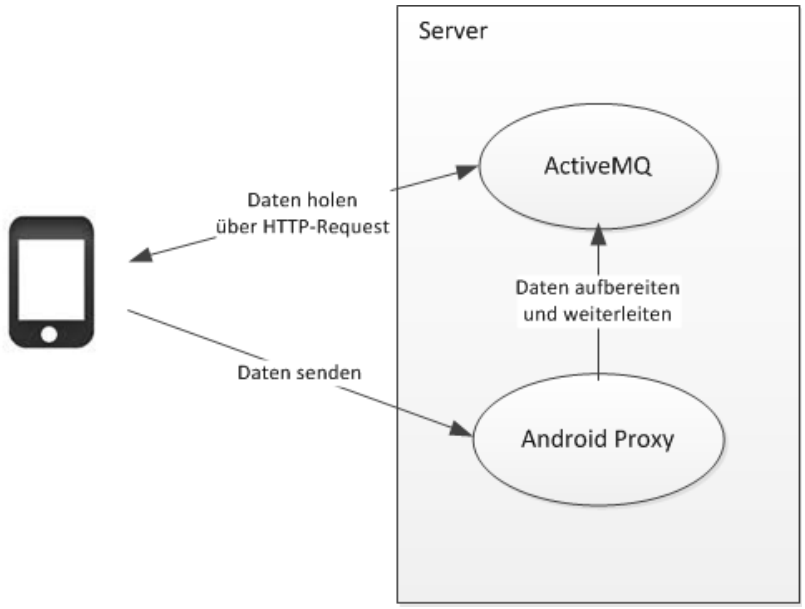


Abbildung 4.3.: Kommunikation zwischen Android™ und dem ActiveMQ
Quelle: (Bornemann, 2011, Abbildung 5.2)

4.3.1. Nachrichtenformat

Wie bereits im Kapitel 2.3.4 beschrieben, ist JSON das Standardnachrichtenformat im Living Place. Aus diesem Grund wird hier ebenfalls JSON für die Kommunikation zwischen dem ActiveMQ und dem Controller verwendet.

4.4. Plug-in Frameworks

Da die zu entwickelnde Software eine Plug-in-Funktionalität besitzen soll, bietet sich der Einsatz eines Plug-in Frameworks an. Nachfolgend werden zwei Frameworks vorgestellt.

4.4.1. JPF

JPF ist eine Bibliothek, die die Implementierung einer Plug-in Architektur in Java-Anwendungen ermöglicht.

> „JPF provides a runtime engine that dynamically discovers and loads "plug-ins".
> A plug-in is a structured component that describes itself to JPF using a "manifest".

> JPF maintains a registry of available plug-ins and the functions they provide (via extension points and extensions).“[3]

JPF ist eine Open Source Implementierung und unterliegt der LGPL-Lizenz. JPF ist in der Version 1.5.1 vom 19.05.2007 verfügbar.

4.4.2. OSGi

> „OSGi technology is a set of specifications that defines a dynamic component system for Java. These specifications reduce software complexity by providing a modular architecture for large-scale distributed systems as well as small, embedded applications.“[4]

OSGi Spezifikation wurde von der OSGi Alliance (gegründet 1999) veröffentlicht. Einige der bekanntesten Implementierungen der OSGi Spezifikationen sind „Eclipse Equinox“, „Apache Felix“ oder „Knopflerfish“.

4.4.3. Fazit Plug-in Framework

Die Verwendung eines Plug-in Frameworks bei der Entwicklung einer Software mit Plug-In-Funktionalität ist sehr sinnvoll, da dadurch die Programmierung, sowie die Erweiterung erleichtert wird. Hier jedoch ist eine Verwendung des Frameworks nicht möglich, da dies die Kenntnis über das Framework, das Programm, sowie gewisse Programmierkenntnisse erfordert und damit gegen die nicht funktionale Anforderung „Erweiterbarkeit“ (siehe 3.3.2) verstößt. Diese erfordert eine Erweiterung des Programms ohne Programmierkenntnisse.

Da die Verwendung eines Frameworks nicht möglich ist, wurde entschieden ein eigenes Plug-in Modell zu entwickeln, welches im Kapitel 4.5 vorgestellt wird.

4.5. Plug-in Modell

Grundsätzlich besteht das Plug-in Modell, wie in der Abbildung 4.4 dargestellt, aus zwei Phasen. Die erste Phase beschäftigt sich mit der Erstellung eines Plug-ins, die zweite mit der Einbindung an das Programm.

- In der ersten Phase wird überlegt, wie ein Plug-in für eine konkrete Komponente aussehen muss, wie die Kommunikation mit seinem Service aussieht und welche Bedienelemente dafür verwendet werden können. Danach wird dieser Plug-in formuliert,

[3] http://jpf.sourceforge.net/, abgerufen am 31.07.2012
[4] http://www.osgi.org/Technology/HomePage, abgerufen am 31.07.2012

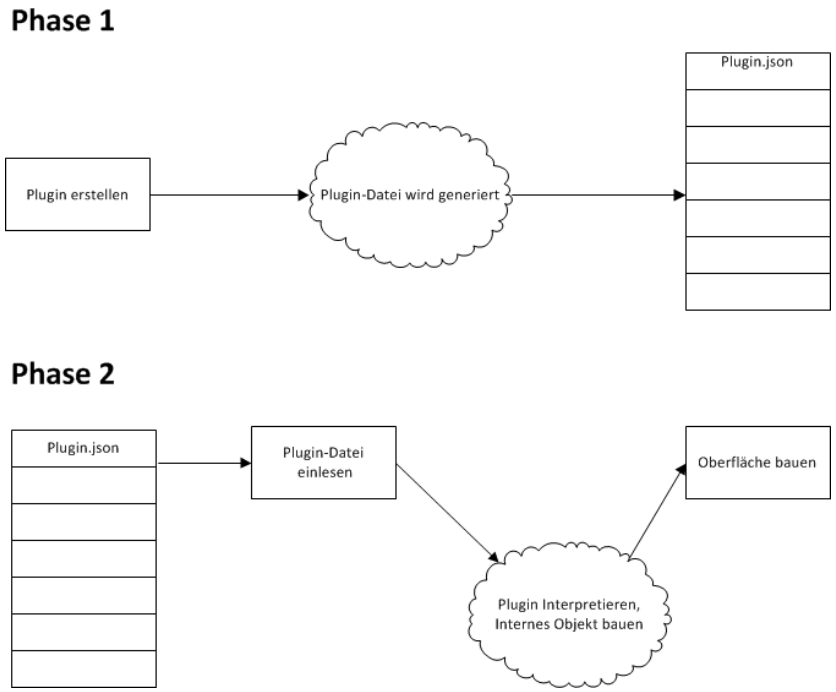


Abbildung 4.4.: Phasen des Plug-in Modells

entweder manuell oder mit Hilfe eines Tools. Am Ende entsteht ein File, der eine wohlgeformte Definition des Plug-ins enthält.

- In der zweiten Phase wird das erstellte Plug-in eingelesen und ein internes Objekt als sein Abbild erstellt. Abhängig von der Definition wird die Oberfläche des Plug-ins gebaut und in das Hauptprogramm eingebunden.

4.5.1. Plug-in Format

Damit die Plug-in-Files gelesen und interpretiert werden können, müssen diese vom System „geparst“ werden. Wenn man sich die Arbeit ersparen und keine neue Sprache inklusive Parser entwickeln möchte, bieten sich schon vorhandene Alternativen an, wie z.B. XML, YAML oder JSON.

XML

Extensible Markup Language (XML) ist eine von SGML(ISO 8879) abgeleitete Metasprache. Die erste Ausgabe der XML-Spezifikation „XML 1.0 Recommendation“ wurde am 10. Februar

1998 von W3C veröffentlicht. Seit dem 26. November 2008 ist die fünfte und aktuellste Version von XML verfügbar. (W3C, 2008)

XML ist eine Metasprache, die zur Beschreibung weiterer Sprachen, wie z.B. XHTML oder zur Erstellung neuer, eigener Sprachen genutzt wird. XML ist eine sehr mächtige und weit verbreitete Sprache. Ein XML-Dokument wird anhand von Tags geparst, woraus folgt, dass alle Elemente einen Öffnungs- und einen Endtag besitzen müssen. XML-Elemente können Attribute, Anweisungen oder Kommentare enthalten und verschachtelt werden. Im Listing 4.1 ist ein Beispiel einer Bestellung in XML aufgeführt. [5]

```
<order>
  <id>731</id>
  <date>16th of May 2011</date>
  <customer>17</customer>
  <items>
    <item>
      <quantity>5</quantity>
      <description>Oolong</description>
      <price>5.98</price>
      <in-stock>true</in-stock>
    </item>
    <item>
      <quantity>2</quantity>
      <description>Assam</description>
      <price>2.95</price>
      <in-stock>false</in-stock>
    </item>
  </items>
</order>
```

Listing 4.1: Bestellung in XML

Die weite Verbreitung von XML hat dafür gesorgt, dass XML unter Android™ unterstützt wird. Die Android™-Plattform benutzt intern XML, um zum Beispiel Oberflächenlayouts beschreiben zu können. Mit den Paketen wie `org.w3c.dom`, `org.xml.sax` oder `org.xmlpull.v1` bietet Android™ von Haus aus eine sehr gute XML-Unterstützung.

[5] http://predic8.de/xml-json-yaml.htm, abgerufen am 11.07.2012

YAML™

YAML Ain't Markup Language (YAML™) ist eine Sprache, die primär zur Datenserialisierung eingesetzt wird. Die erste historische Version von YAML™ wurde am 30. März 2001 veröffentlicht, aktuell ist YAML™ in der Version 1.12 3^{rd} Edition vom 01. Oktober 2009 verfügbar. Einer der Vorteile von YAML™ im Vergleich zu XML ist die bessere Lesbarkeit für den Menschen. (YAML™, 2009)

„Dies wurde durch einen Verzicht auf Klammern, Anführungszeichen und Tags erreicht. Für die Darstellung von hierarchischen Strukturen verwendet YAML Einrückungen."[6]

Im Listing 4.2 ist ein Beispiel einer Bestellung in YAML™ aufgeführt. [7]

```
1  ---
2  id: 731
3  date: 16th of May 2011
4  customer: 17
5  items:
6  - quantity: 5
7    description: Oolong
8    price: 5.98
9    in-stock: true
10
11 - quantity: 2
12   description: Assam
13   price: 2.95
14   in-stock: false
```

Listing 4.2: Bestellung in YAML™

Entgegen dem XML ist YAML™ nicht so stark in der Android™-Entwicklung verbreitet. Dafür spricht auch die fehlende Unterstützung seitens der Android™-Plattform. Mit den zusätzlichen Bibliotheken, wie „snakeyaml"[8] lässt sich YAML™ jedoch auch unter Android™ nutzen.

JSON

> „JSON (JavaScript Object Notation) ist ein schlankes Datenaustauschformat, das für Menschen einfach zu lesen und zu schreiben und für Maschinen einfach zu parsen (Analysieren von Datenstrukturen) und zu generieren ist. Es basiert auf

[6] http://predic8.de/xml-json-yaml.htm, abgerufen am 11.07.2012
[7] http://predic8.de/xml-json-yaml.htm, abgerufen am 11.07.2012
[8] http://code.google.com/p/snakeyaml/wiki/howto#Android, abgerufen am 11.07.2012

> einer Untermenge der JavaScript Programmiersprache, Standard ECMA-262 dritte Edition - Dezember 1999.“[9]

JSON baut auf lediglich zwei Strukturen auf[10]:

1. Key-Value Paare
2. Eine geordnete Liste von Werten, Arrays

Die Abbildung 4.5 zeigt den Aufbau des Key-Value Paares, welches folgende Form besitzt: `{Key:Value}`. Die Abbildung 4.6 stellt den Aufbau einer geordneter Liste dar und die Abbildung 4.7 zeigt den möglichen Inhalt des Wertes. Die Arrays haben in JSON folgende Form: `{“arrayitems“:[value1, value2]}` und ein Value kann ein Objekt oder ein primitiver Typ sein. Im Listing 4.3 ist ein Beispiel einer Bestellung in JSON aufgeführt. [11]

```
{
    "id": 731,
    "date": "16th of May 2011",
    "customer": 17,
    "items" : [
        {
            "quantity": 5,
            "description": "Oolong",
            "price": 5.98,
            "in-stock": true
        },
        {
            "quantity": 2,
            "description": "Assam",
            "price": 2.95,
            "in-stock": false
        }
    ]
}
```

Listing 4.3: Bestellung in JSON

Ähnlich, wie XML wird auch JSON sehr oft unter Android™ verwendet. Android™ ist mit dem Paket `org.json` ausgestattet, der eine einfache Benutzung von JSON erlaubt. Wem es

[9] http://www.json.org/json-de.html, abgerufen am 11.07.2012
[10] http://www.json.org/json-de.html, abgerufen am 11.07.2012
[11] http://predic8.de/xml-json-yaml.htm, abgerufen am 11.07.2012

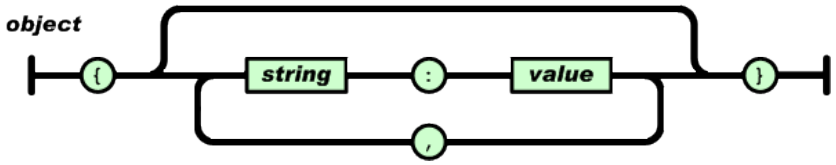


Abbildung 4.5.: Aufbau eines Objekts in JSON
Quelle: http://www.json.org/json-de.html, abgerufen am 11.07.2012

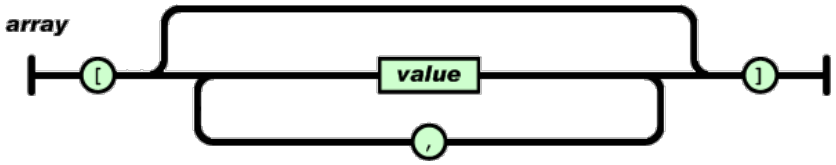


Abbildung 4.6.: Aufbau eines Arrays in JSON
Quelle: http://www.json.org/json-de.html, abgerufen am 11.07.2012

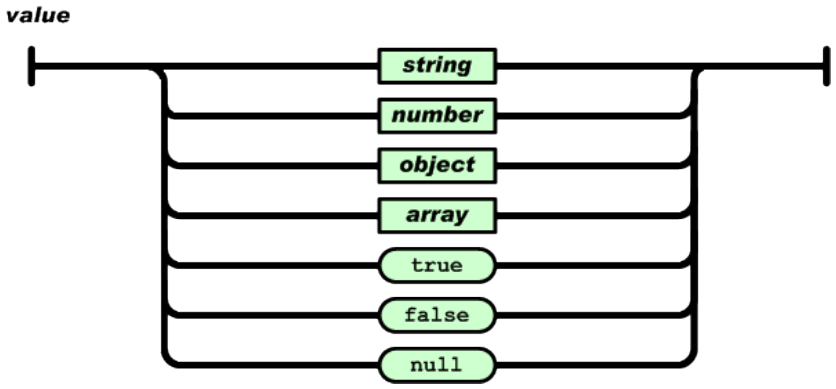


Abbildung 4.7.: Aufbau eines Values in JSON
Quelle: http://www.json.org/json-de.html, abgerufen am 11.07.2012

allerdings zu wenig ist, kann auf eine der vielen Implementierungen (GSON, Flexjson, Jackson usw.) von Drittanbietern zurückgreifen.

Fazit Plug-in Format

Da ein Plug-in letztendlich ein serialisiertes Objekt ist, bietet sich hier nur ein Format an, der am besten für die Serialisierung bzw. Deserialisierung von Daten geeignet ist. Demnach muss die Entscheidung zwischen YAML™ und JSON stattfinden. JSON hat im Gegensatz zu YAML™ eine viel bessere Android™-Unterstützung, was die Entscheidung zugunsten des JSON-Formats ausfallen lässt.

4.5.2. Plug-in Aufbau

Aufbaukonzept

Die Idee, die die Grundlage für das Aufbaukonzept bietet, wurde der Funktionsweise einer Fernbedienung entnommen. Beim Drücken einer Taste auf der Fernbedienung, wird ein bestimmter Code bzw. eine Nachricht an das zu steuernde Gerät verschickt. So wird auch hier beim Betätigen eines Bedienelements eine Nachricht an das ActiveMQ verschickt, welches diese Nachricht an den richtigen Controller-Service zustellt und somit eine Aktion auslöst. Die Umsetzung dieses Konzept bedarf einer Analyse der Komponenten-API‘s. Dabei ist es wichtig herauszufinden welche Bedienelemente benutzt werden können und wie die dazugehörigen Nachrichten aussehen.

Durch Analyse und Vergleich der im Living Place vorhandenen Komponenten- API’s wurde festgestellt, dass es grundsätzlich zwei unterschiedliche Nachrichtentypen gibt:

- Nachrichten, die keinen Parameter besitzen
- Nachrichten, die einen oder mehrere Parameter besitzen

Die unterschiedlichen Nachrichtentypen erfordern unterschiedliche Herangehensweise und Einsatz von unterschiedlichen Bedienelementen. Einfache Bedienelemente, wie zum Beispiel Buttons, haben keinen Zustand und werden mit den parameterlosen Nachrichten verwendet. Nach dem Betätigen eines solchen Bedienelements wird die Nachricht sofort versendet.

Bedienelemente entgegen, die mehrere Zustände besitzen, wie zum Beispiel Slider, werden stets mit Nachrichten benutzt, die einen oder mehrere Parameter besitzen. Hier wird die Nachricht zuerst entsprechend des Zustandes des Bedienelements manipuliert und erst dann versendet.

Eine Sonderform stellen die sogenannten Toggle-Buttons dar, wie zum Beispiel Switche. Dabei handelt es sich um Bedienelemente, die zwei Zustände besitzen. Die beiden Zustände können dabei unterschiedlich dargestellt werden:

- Durch zwei Unterschiedliche Nachrichten, die je einen Zustand repräsentieren
- Durch eine Nachricht mit einem sich abwechselnden Parameter, der je einen Zustand repräsentiert

Im ersten Fall gibt es zwei Lösungswege. Entweder wird die Problematik aufgeteilt und wie zwei Nachrichten ohne Parameter behandelt oder ein dem Switch ähnliches Bedienelement erstellt, der in der Lage ist zwei Nachrichten als Parameter aufzunehmen. Dafür kann die zweite Nachricht im Argumenten-Array (siehe 4.5.2) dem Bedienelement übergeben werden.

Im zweiten Fall kann die Nachricht, wie die Nachricht mit mehreren Parametern mit dem Bedienelement Switch verwendet. Dafür können die Parameter dem Switch im Argumenten-Array (siehe 4.5.2) übergeben werden. Sobald der Switch betätigt wird, ersetzt das Programm den Parameter in der Nachricht und versendet diese.

Eine weitere wichtige Information, die ein Plug-in enthalten muss, ist der Pfad zum Objekt, welches gesteuert wird. Diese Information ermöglicht den Aufbau einer Navigation.

Wenn man den Aufbaukonzept des Plug-ins grob zusammenfasst, erhält man eine Beschreibung der Pfade mit den dazugehörigen Bedienelementen-Nachrichten-Kombinationen für eine einzige Komponente des Living Place. Im nachfolgenden Kapitel 4.5.2 wird der detaillierte Aufbau eines Plug-ins beschrieben.

Aufbau

Durch Analyse der Dokumentation zu den vorhandenen Komponenten des Living Place, der Infrastruktur des Living Place, sowie des unterschiedlichen Aufbaus der Nachrichten, wurde ein Konzept entwickelt, welches eine universelle Darstellung von Plug-ins ermöglicht. Nachfolgend wird der Aufbau genauer erläutert.

Jedes Plug-in wird in der JSON-Syntax erstellt und hat stets den folgenden Aufbau:

- Plug-in Name
  Name des Plug-ins bzw. später Bezeichnung des Tabs
- Pointer
  Die Position in der das Plug-in starten soll, meistens Root-Knoten „LP“

- Server
  IP-Adresse bzw. Name des Servers, auf dem das Android Proxy läuft
- Port
  Port-Nummer des Servers, auf dem das entsprechende Android Proxy läuft
- Main-Array
  Das Objekt-Array, welches die Information über die gesamten Steuerelemente beinhaltet

Ein Main-Array besteht aus Objekten:

- Location-Pfad
  beinhalten die Information über die Location
- Controls-Array
  Das Objekt-Array, welches Information über unterschiedliche Steuerelemente, die aber einer Location angehören, beinhaltet

Ein Controls-Array besteht aus folgenden Objekten:

- Control-Item
  beschreibt ein Steuerelement, wie z.B. einen Button oder Schieber

Manche Steuerelemente kann man gruppieren und somit aus mehreren einfachen ein komplexes Steuerelement erstellen. Aus drei einfachen Slider kann man zum Beispiel einen RGB-Slider erstellen, wo jeder einzelner Slider einen der drei RGB-Parameter darstellt. Jede Gruppe kann also entweder einen oder mehrere einfache Steuerelemente besitzen. Eine Gruppe besitzt stets nur eine Nachricht. Ein Control-Item besteht aus:

- Gruppentitel
  Name der Gruppe der Elemente
- LP-Label
  Label, der im Living Place genutzt und für die Statusabfrage eingesetzt wird
- Group-Array
  Beinhaltet die Information über einen oder mehrere primitive Steuerelemente
- Nachricht
  Nachricht, die an den Service verschickt werden soll

Ein Group-Array-Objekt besteht aus:

- Group-Item-Typ
  der „primitive“ Typ, zur Zeit nur Slider, Button oder Switch
- Group-Item-Title
  Bezeichnung der einzelner Steuerelemente, kann bei nur einem Steuerelement leer sein
- Group-Item-Argument-Array
  Ein Array, der Konfigurationsparameter, wie untere oder obere Grenzen, sowie die Nullposition, für einzelne Steuerelemente beinhaltet

Eine Nachricht besteht aus:

- Message-Topic
  Topic-Name, an den die Nachricht adressiert wird
- Message-Typ
  Typ der Nachricht: Queue oder Topic
- Message-Value
  Die Nachricht selbst, wie diese vom Entwickler der Komponente definiert wurde

Falls eine Nachricht Werte enthält, die durch einzelne Steuerelemente einer Gruppe manipuliert werden, werden diese Stellen in dem Message-Value durch „paramX“ dargestellt, wobei „X“ der Nummer des Steuerelements entspricht. Im Listing 4.4 ist ein Auszug aus der Definition eines RGB-Sliders aufgeführt:

```
...
{
          "control_item" : {
            "group_title" : "Color",
            "lp_label" : "lounge_light_color",
            "group_array" : [{
                "group_item_typ" : "Slider",
                "group_item_title" : "Red",
                "group_item_argument_array" : ["0", "255", "0"]
              }, {
                "group_item_typ" : "Slider",
                "group_item_title" : "Green",
                "group_item_argument_array" : ["0", "255", "0"]
```

```
14              }, {
15                "group_item_typ" : "Slider",
16                "group_item_title" : "Blue",
17                "group_item_argument_array" : ["0", "255", "0"]
18              }
19            ],
20            "message" : {
21              "message_topic" : "LP.LIGHTCONTROL",
22              "message_typ" : "topic",
23              "message_value" : {
24                "values" : {
25                  "fadeTime" : "0",
26                  "red" : "param1",
27                  "blue" : "param3",
28                  "green" : "param2"
29                },
30                "action" : "lounge_light_color",
31                "Id" : "controller_1",
32                "Version" : null
33              }
34            }
35          }
36          ...
```

Listing 4.4: Definition eines RGB-Sliders in JSON

Wie man anhand des Beispiels erkennen kann, besteht ein RGB-Slider aus drei einfachen Slidern. Es ist jeweils ein Slider für die Farbwerte Rot, Grün und Blau zuständig. Jeder dieser Slider wird zu Beginn wie folgt konfiguriert: Untergrenze-0, Obergrenze-255, Startwert-0. In der Message wurden die entsprechenden Wertefelder durch param1 bis param3 markiert, wobei man erkennen kann, dass die Nummern der Reihenfolge der Steuerelemente entsprechen: Red-param1, Green-param2, Blue-Param3.

4.5.3. Hierarchische Darstellung

Eine der Aspekte bei der Erstellung eines Plug-ins ist die räumliche Navigation. Beginnend mit dem gesamten Living Place kann man die Navigation bis zum kleinsten Objekt granulieren. Diese Vorgehensweise erfordert eine hierarchische Darstellung im Plug-in. Sehr oft ist es aber so, dass die Plug-ins durch tiefe Hierarchieabbildung sehr unübersichtlich werden und man so leichter ein Fehler bei der Erstellung macht. Aus diesem Grund wurde entschieden

die Hierarchie auf eine andere Art und Weise darzustellen. Die Hierarchie in den Plug-ins wird mittels Punktnotation realisiert. Jedes Objekt bekommt einen sogenannten Location-Pfad. Dieser kann wie folgt aussehen: „*LP.Kitchen.Cooking*“, wobei „LP“ immer das Root-Knoten ist. Ein weiterer Kindesknoten von Kitchen kann wie folgt dargestellt werden: „*LP.Kitchen.Main*“.

Solche Notation stellt zwar ein gewisses Overhead bei der Programmierung dar, da man die Punktnotation auf eine Baumstruktur intern Abbilden muss, um besser navigieren zu können, jedoch lohnt sich dieses Overhead im Sinne der Übersichtlichkeit.

4.6. Alternatives Plug-in Konzept

Eine weitere Gestaltung des User Interfaces, sowie die Gestaltung der Plugins wäre denkbar. Das aktuelle Design sieht vor, dass jede Komponente des Living Place ein eigenes Plug-in erhält. Die Plug-ins unterstützen dann die Navigation beginnend mit dem Living Place als Root-Knoten über einzelne Räume bis hin zum kleinsten Objekt. Die Alternative wäre die Plug-ins pro Raum oder Objekt zu gestalten und die Bedienelemente für alle an dieser Stelle verfügbaren Komponenten zu gruppieren. Somit entfällt die Verwendung von Tabs. Die GUI wird nur in zwei Bereiche vertikal aufgeteilt, wobei sich links die Navigation befindet und rechts die Bedienelemente dargestellt sind.

Der Vorteil eines solchen Aufbaus ist die Verfügbarkeit aller Bedienelemente für das zu steuernde Objekt auf einem Platz, was die Steuerung erleichtert, da die unnötige Navigation entfällt. Dieses Konzept wird sehr gut mit einem Ortungssystem harmonieren, wenn die Software selbständig die Position erkennt und die entsprechende Bedienelemente darstellt.

Dieses Konzept hat aber auch Nachteile. Bei steigender Anzahl der Bedienelementen wird die Steuerung unübersichtlich und erschwert die Benutzung des Programms. Zudem erschwert sich auch die interne Verwaltung der Plug-ins. Das nachträgliche Einbinden der Funktionalität mittels Plug-in erfordert eine Möglichkeit die Bedienelemente so zu gruppieren, dass diese logisch zusammenhängend sind, was einen zusätzlichen Overhead bedeutet.

Angesichts der Unübersichtlichkeit mit steigender Anzahl der Bedienelemente sowie Komponenten kann dieses Modell mit dem aktuellen Modell leider nicht konkurrieren.

4.7. Persistenz der Daten

Wie Bereits im Kapitel 2.1.1 erwähnt, durchläuft jede Aktivität, aus denen ein Programm in Android™ besteht, einen bestimmten Lebenszyklus. So kann zum Beispiel ein Programm in Android™ in den Hintergrund verschoben werden, falls mitten in der Ausführung ein Anruf

kommt, da die Telefonapp in dem Moment logischerweise höhere Priorität hat. Falls zudem im System Speicherknappheit herrscht, kann das zuvor aktive Programm gar beendet werden, um Ressourcen frei zu legen. In so einer Situation ist es sehr wichtig einen aktuellen Stand festzuhalten, um diesen später wiederherstellen zu können.

In Android™ gibt es primär drei unterschiedliche Möglichkeiten Daten persistent zu speichern (Lee, 2012, Kapitel 6, S. 251):

- Mittels Shared Preferences
- In einer SQLite-Datenbank
- Durch Speichern auf einen nicht flüchtigen Speicher

4.7.1. Shared Preferences

Android™ stellt einen Shared Preferences Framework zur Verfügung. Dieses erlaubt eine einfache persistente Speicherung von Key-Value Paaren. Das Android™-System übernimmt die physikalische Speicherung, indem dieser das Key-Value-Paar in eine XML schreibt, die dann wieder mittels Frameworks ausgelesen werden kann. Diese Art des Speicherns ist für kleinere Datenmengen gedacht, wie z.B. Einstellungen eines Programms (Textgröße, Font-Name, Hintergrundfarbe etc.) und bringt einen Performancevorteil gegenüber der Datenbanken mit sich und ist mit einem sehr geringem Aufwand zu realisieren.(Lee, 2012, Kapitel 6)

4.7.2. SQLite-Datenbank

> „Normalerweise dienen Datenbanken zur Speicherung und zum effizienten Zugriff auf große, strukturierte Datenmengen, die mehreren Anwendungen zeitgleich zur Verfügung gestellt werden. Diese Rahmenbedingungen sind auf einem Android™-Gerät nicht vorhanden. In der Regel werden dort aus Platzgründen kleinere Datenmengen verwaltet. Auf die Inhalte einer Datenbank greift meist nur eine Anwendung zu.“(Becker und Pant, 2009, Kapitel 11.2, S. 161)

Trotzdem stellt Android™ zum Speichern größerer Datenmengen, als die die mittels Shared Preferences gespeichert werden können, eine relationale Datenbank zur Verfügung. SQLite ist die Datenbank, die jeder Android™-Application zur Verfügung steht.

> „SQLite is a very popular embedded database, as it combines a clean SQL interface with a very small memory footprint and decent speed. Moreover, it is

> public domain, so everyone can use it. Many firms (e.g., Adobe, Apple, Google, Sun, and Symbian) and open source projects (e.g., Mozilla, PHP, and Python) ship products with SQLite."(Allen und Murphy, 2012, Kapitel 32, S. 367)

Mit SQLite hat Android™ eine sehr gute und effiziente Alternative zur persistenter Datenspeicherung geschaffen, die für größere Datenmengen, wie Bestenlisten, Messdaten oder Statistiken gut geeignet ist. Jedoch dürfen die Performanceeinbussen, die im Vergleich zur anderen Speichermöglichkeiten entstehen, sowie das Overhead, welches bei Erstellung einer Datenbank entsteht, nicht vernachlässigt werden.

4.7.3. Speichern auf einen nicht flüchtigen Speicher

Eine weitere Alternative, die Android™ zum persistenten Speichern von Daten bietet, ist die Speicherung von Daten auf einen nicht flüchtigen Speicher. „...Für diese Aufgabe können in Android™ Klassen des Pakets *java.io* genutzt werden ..."(Lee, 2012, Kapitel 6, S.263) Man unterscheidet in Android™ zwischen zwei Speicherbereichen:

1. Internal Data Storage
   Hierbei handelt es sich um einen Bereich, der nur einer einzigen Anwendung zur Verfügung steht. Keine andere Anwendung kann auf diesen Bereich zugreifen und die dort vorhandene Daten manipulieren. Diesen Datenbereich kann man in der DDMS-Perspektive der Entwicklungsumgebung Eclipse unter folgendem Pfad einsehen: `/data/data/app.name.inkl.package/files` (Lee, 2012, Kapitel 6, S.267)

2. External Data Storage
   Hierbei handelt es sich um einen Bereich, der für alle Anwendungen zur Verfügung steht. Damit das External Data Storage von der Anwendung genutzt werden kann, muss die Methode `getExternalStorageDirectory()` aufgerufen werden. Diese liefert den vollständigen Pfad zum externen Speicher. Im Emulator ist dieser: `/mnt/sdcard`. (Lee, 2012, Kapitel 6, S.269) Des Weiteren muss die `AndroidManifest.xml` der Anwendung um das Zugriffsrecht `WRITE_EXTERNAL_STORAGE` erweitert werden, um das Speichern auf dem externen Speicher zu erlauben. (Lee, 2012, Kapitel 6, S.270)

4.7.4. Fazit, Persistent der Daten

In Android™ sind drei unterschiedliche Möglichkeiten vorhanden, um Daten persistent Speichern zu können. Jedoch ist für die gestellte Aufgabe die herkömmliche Speicherung auf einen nicht flüchtigen Speicher am besten geeignet. Wie bereits im Kapitel 4.5.1 beschrieben, sind

Plug-ins serialisierte Objekte, die in JSON-Notation vorliegen. Um Zustände dieser Objekte persistent zu machen, bietet es sich an, die internen Objekte wieder als JSON zu serialisieren und in den nicht flüchtigen Speicher zu schreiben, um diese später mit den gleichen Werkzeugen, die zur Deserialisierung von Plug-ins genutzt werden, zu deserialisieren.

4.8. Konsistenz der Daten

Folgendes Szenario macht die Überlegungen über die Konsistenz der Daten bedeutend. Einsatzgebiet des Smart Home Controllers ist das Living Place in Hamburg. Bereits jetzt werden alle möglichen Services auf unterschiedliche Art und Weise gesteuert. Mal wird ein Kommando direkt über die Konsole versendet, mal testet einer der Kommilitonen seine eigene Entwicklung. Alles das führt dazu, dass die Information über die Zustände der einzelnen Komponenten inkonsistent sein kann. Ein weiteres Beispiel für Dateninkonsistenz wäre die parallele Benutzung des Controllers von zwei oder mehreren Geräten. Falls beispielsweise eine Lampe von einem Controller auf eine Intensität von 50% gestellt wird, bekommt ein anderer Controller dies gar nicht mit.

Leider ist dieses Problem schwieriger in den Griff zu bekommen, als es auf den ersten Blick erscheint. Viele bereits vorhandenen Services bieten eine Möglichkeit den Zustand einer Komponente abzufragen, erst gar nicht an. Es ist zum Beispiel nicht möglich den Service, welcher für die Steuerung des Lichts zuständig ist, nach dem Zustand einer Lampe zu fragen. Zur Zeit existiert leider für dieses Problem keine Lösung.

Mögliche Ansätze das Problem in den Griff zu bekommen wären:

- **Jeden Entwickler im Living Place dazu zu verpflichten einen entsprechenden Dienst für die eigene Entwicklung anzubieten bzw. nachzurüsten**
  Dieser Ansatz würde zwar die Problematik beheben, jedoch spricht jegliche Verpflichtung gegen die Philosophie des Living Place und würde gegebenenfalls die Entwickler bei den Kernaufgaben unnötig hindern.

- **In die Software eine Art Synchronisationsmechanismus integrieren**
  Dieser Ansatz würde die Problematik nur zum Teil beheben, da die Inkonsistenz nur zwischen den Controllern behoben wird, jedoch nicht im Zusammenhang mit anderen Systemen.

- **Einen zusätzlichen Service entwickeln, der den kompletten Zustand des Living Place kennt und Auskunft über einzelne Komponenten bereitstellt**
  Dieser Ansatz erscheint zur Zeit am sinnvollsten zu sein. Die Vision wäre eine Folgende:

einen Service zu haben, der auf dem gleichen Server, wie der ActiveMQ läuft, alle eingehenden Nachrichten analysiert und daraus den Zustand des Living Place generiert. Über eine Abfrageschnittstelle bietet dieser die Auskunft über den Zustand einzelner Komponenten an.

Solange der zuletzt beschriebene Abfrageservice lediglich eine Vision ist, wird dieser im Programm mit einem Mockup simuliert. Die Abfrage wird einmalig beim Programmstart ausgeführt, um einen gemeinsamen Initial-Zustand zu erreichen. Das Problem der Konsistenz ist nicht kritisch für den Controller und hat viel mehr einen ästhetischen Hintergrund. Bei den Bedienelementen, an denen der Zustand der Komponente nicht erkennbar ist, wie z.B. Buttons, entfällt das Problem völlig. Bei den Bedienelementen, die den Zustand der Komponente repräsentieren können, wie z.B. Slider oder Switch, ist die Inkonsistenz lediglich bis zur ersten Betätigung von Bedeutung. Trotz alledem macht der Einsatz eines Abfrageservices sehr viel Sinn und würde ebenfalls helfen andere Probleme zu lösen. Die Existenz eines solchen Services würde auch die Persistenz-Problematik entfallen lassen. Mit einem solchen Service ist der Zustand einer Komponente jederzeit abfragbar und aktuell, sodass bei der Wiederherstellung des UI's der Zustand einfach nur abgefragt wird. Somit entfällt die persistente Speicherung des Zustands völlig.

4.9. Softwaredesign

Um die Entwicklung eines Prototypen zu ermöglichen, wurde ein Softwaredesign entwickelt. Das gesamte Design wird in sechs Pakete unterteilt, die jeweils für bestimmte Aufgaben zuständig sind.

1. **plugin**
   Dieses Paket repräsentiert ein deserialisiertes Plug-in und beinhaltet alle Klassen, die die Erstellung und Nutzung im Programm ermöglichen. Die Abbildung 4.8 stellt das Paket „plugin“ als UML-Klassendiagramm dar.

   Die Klasse *TabContent* beinhaltet dabei die komplette Navigationskomponente, die durch Klasse *MyTree* realisiert wird und für die Navigation innerhalb des Plug-ins zuständig ist. *MyTree* besteht aus Knoten, die durch die generische Klasse *Nodes* repräsentiert sind und verkettet diese. Da jeder Knoten jeweils eine Liste von Kindesknoten erhält, ergibt dieses Konstrukt eine Baumstruktur. Jeder Knoten beinhaltet zudem Information über die Bedienelemente, die zum Knoten gehören. Dies implementiert die Klasse *NodeProperties*, indem diese eine Liste von sogenannten *ElementGroup'en* verwaltet. Eine *ElementGroup*

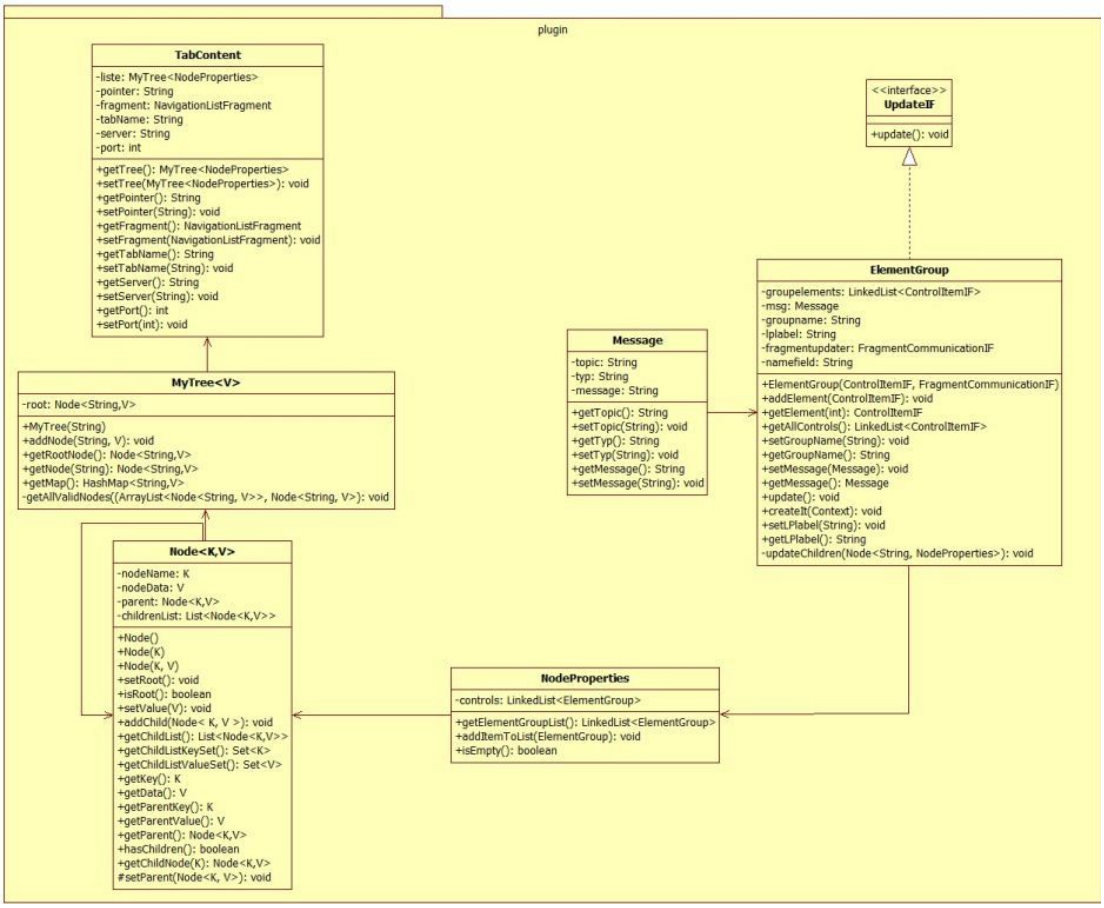


Abbildung 4.8.: Paket „plugin“

ist ein Verbund von *Message*, der Nachricht, die an das ActiveMQ gesendet wird, sowie einem oder mehreren „primitiven“ Bedienelementen. Die „primitiven“ Bedienelementen nutzen das *UpdateIF*, um die Zustandsänderungen der *ElementGroup* mitzuteilen.

2. **uielement**
   Dieses Paket beinhaltet die Implementierungen der „primitiven“ Bedienelementen. Die Abbildung 4.9 stellt das Paket „uielement“ als UML-Klassendiagramm dar.

   Die Klassen *MySwitch*, *MyButton* und *NewSlider* repräsentieren die Bedienelementen und implementieren das *ControlItemIF*, der diese Klassen generalisiert. Die Klasse *ControlFactory* übernimmt die Erstellung der Bedienelemente.

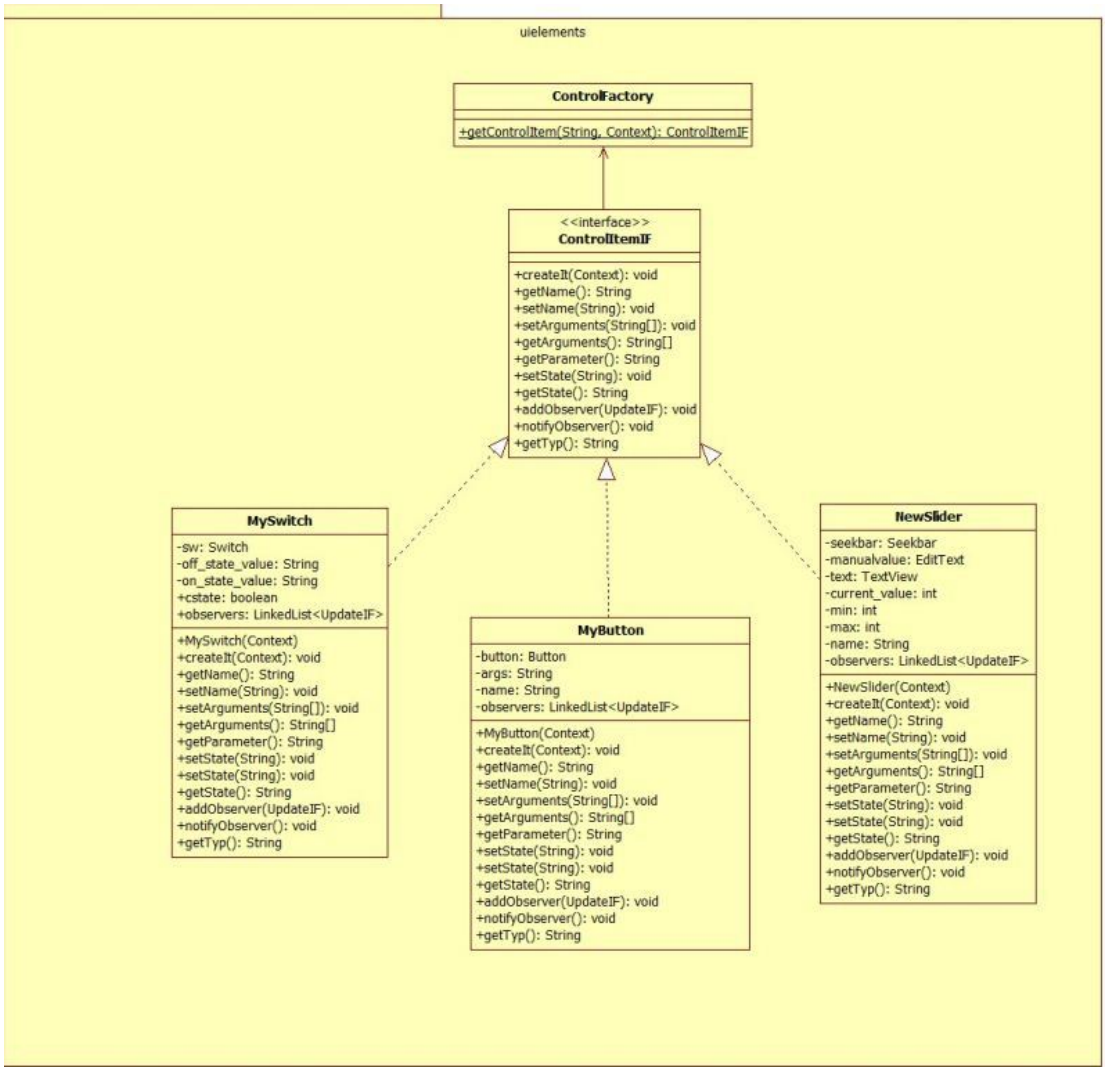


Abbildung 4.9.: Paket „uielement“

3. **helper**
   Dieses Paket enthält Klassen, deren Methoden „static“ sind und für Operationen mit dem Dateisystem Verantwortung tragen. Des Weiteren übernehmen diese die Serialisierung bzw. Deserialisierung von Plug-ins. Die Abbildung 4.10 stellt das Paket „helper“ als UML-Klassendiagramm dar.

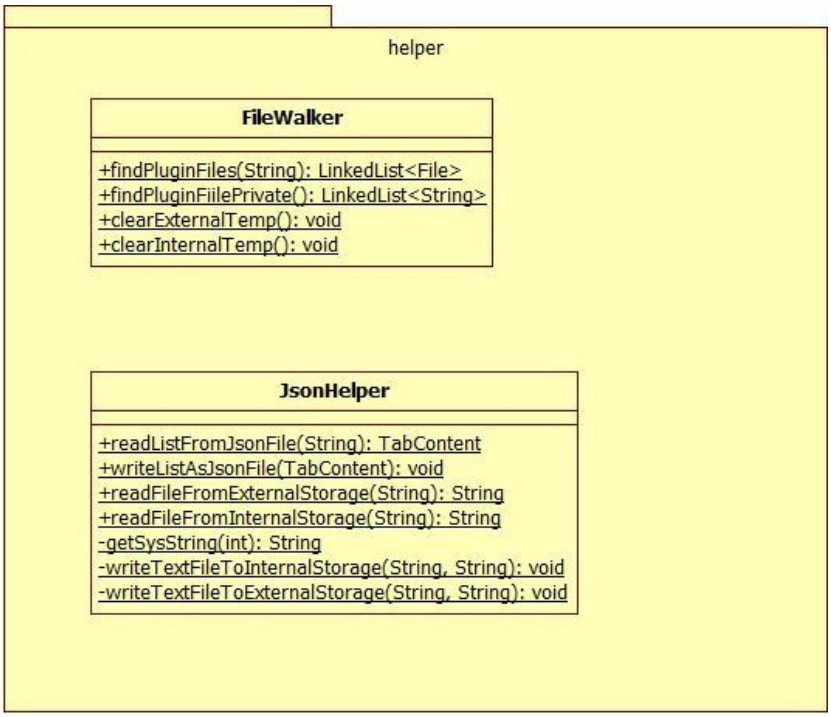


Abbildung 4.10.: Paket „helper“

   Die Klasse *Filewalker* enthält Methoden zum Durchsuchen des internen sowie externen Speichers nach serialisierten Plug-in-Dateien. Zudem enthält diese Klasse Methoden zum Entfernen der temporären Dateien. Die Klasse *JsonHelper* enthält außer Methoden zur Serialisierung, sowie Deserialisierung, Methoden, die das Schreiben und Lesen in den internen bzw. externen Speicher übernehmen.

4. **mockup**
   Dieses Paket enthält eine Klasse, die den fehlenden Abfrageservice im Living Place simuliert. Die Abbildung 4.11 stellt das Paket „mockup“ als UML-Klassendiagramm dar.

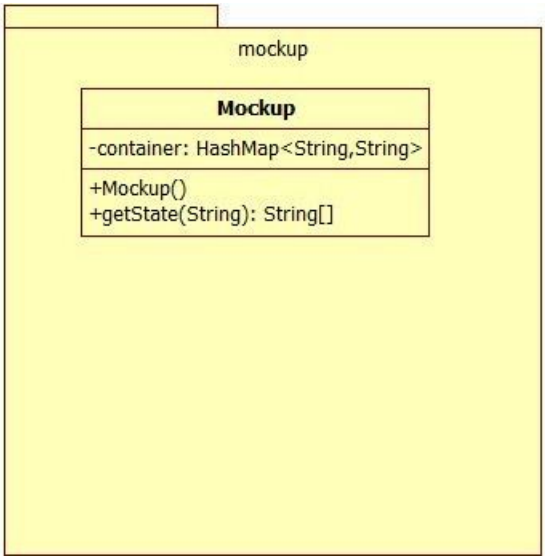


Abbildung 4.11.: Paket „mockup“

Die Klasse *Mockup* enthält intern eine HashMap mit Key-Value Paaren, wobei das Key dem Namen eines Objekts im Living Place entspricht und das Value ein String-Array mit Daten, die den Zustand repräsentieren, ist.

5. **AndroidPublisher**
   Dieses Paket stellt das Subprogramm AndroidPublisher von Sven Boris Bornemann (siehe (Bornemann, 2011, Kapitel 5.1.3)) dar und übernimmt die Kommunikation mit dem ActiveMQ, indem es eine Nachricht über die Socket-Schnittstelle an einen Proxy-Server sendet, der wiederum die Nachricht an das ActiveMQ weiterleitet.

6. **ui**
   Dieses Paket enthält die Main-Klasse, die die Koordination übernimmt und für die richtige Darstellung auf dem Bildschirm sorgt. Die Abbildung 4.12 stellt das Paket „ui“ als UML-Klassendiagramm dar.

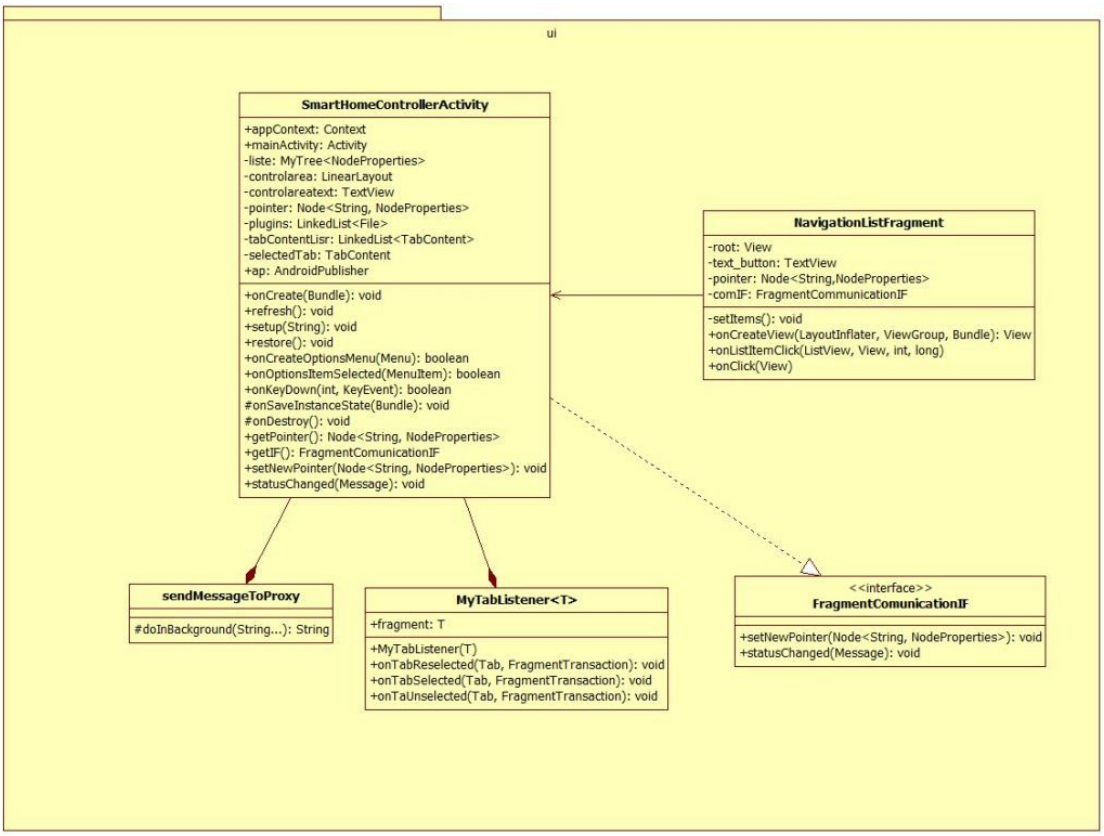


Abbildung 4.12.: Paket „ui“

Die Klasse *SmartHomeControllerActivity* ist die Haupt- und einzige Activity im gesamten Programm. Diese gewährleistet das richtige Durchlaufen des Android™-Lebenszyklus. Mithilfe der Klasse *NavigationListFragment* wird das Tab-Konzept in der Action Bar um-

gesetzt. Die interne Klasse *sendMessageToProxy* übernimmt das Erstellen eines Threads zum Versenden einer Nachricht an das ActiveMQ.

Wenn man den Programmablauf in Worte fassen soll, dann muss dies wie folgt aussehen: Nach dem Start des Programms durchsucht der *Filewalker* das Dateisystem nach vorhandenen Plug-ins. *JsonHelper* liest die Plug-ins ein und deserialisiert diese. Als Ergebnis hat man zum Schluss Objekte der Klasse *TabContent*, die jeweils ein Plug-in repräsentieren und von der *SmartHomeControllerActivity* als je ein Tab aufgenommen werden. Entsprechend dem gewählten Tab wird vom *SmartHomeControllerActivity* das richtige *NavigationListFragment* geladen und die Navigationsliste, sowie die zugehörigen Bedienelemente auf dem Bildschirm dargestellt. Sobald ein Bedienelement seinen Zustand ändert, wird diese Zustandsänderung dem *ElementGroup* mittels *UpdateIF* mitgeteilt. Dort wird die entsprechende Nachricht bearbeitet und mit Hilfe von *FragmentCommunicationIF* an die *SmartHomeControllerActivity* weitergereicht, die dann das Senden der Nachricht an das ActiveMQ in einem separaten Thread ausführt. Die Abbildung 4.13 stellt noch einmal das Softwaredesign im Überblick dar.

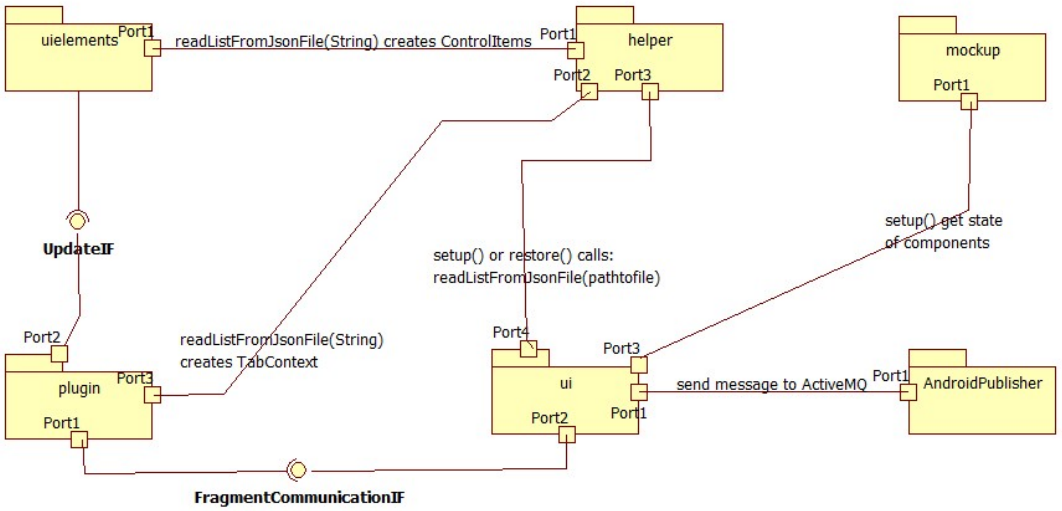


Abbildung 4.13.: Gesamtansicht Softwaredesign

4.10. Fazit

In diesem Kapitel wurde ein mögliches Design der Software vorgestellt. Dabei wurden Aspekte, wie die Gestaltung des UI's, Kommunikationsmöglichkeiten und das Nachrichtenformat,

sowie Aufbau und Format der Plug-ins betrachtet. Weiterhin wurde über die Möglichkeiten der persistenten Datenspeicherung und Konsistenzerhaltung der Daten diskutiert und ein Softwaredesign erarbeitet, der die obengenannten Aspekte und Überlegungen berücksichtigt und eine der Lösungen für das gesetzte Ziel darstellt. Realisierung des hier vorgestellten Designs findet im Folgekapitel 5 statt.

5. Realisierung

Um in dem Kapitel 3.3.2 formulierten Anforderungen zu genügen, wurden zwei Programme erstellt:

1. **Plug-in Builder**
   entspricht der Phase 1 des Plug-in Modells aus Kapitel 4.5
2. **SmartHomeController**
   entspricht der Phase 2 des Plug-in Modells aus Kapitel 4.5

Diese werden in diesem Kapitel vorgestellt.

5.1. Plug-in Builder

Das Programm „Plug-in Builder“ ist ein GUI-basiertes Tool, welches die Erstellung einer Plug-in-Datei erleichtern soll. Ein Plug-in kann auch mit einem einfachen Texteditor erstellt werden, der jedoch bestimmten Regeln entsprechen muss, auf die im Kapitel 4.5.2 näher eingegangen wurde. Die Gefahr einen Schreibfehler zu begehen, ist dadurch relativ groß. Der Plug-in Builder sorgt automatisch für die korrekte Schreibweise einer Plug-in Datei. Der Plug-in Builder ist ein Java-Programm und ist dementsprechend plattformunabhängig.

5.1.1. Evaluierung anhand des Licht-Plug-ins

Die Bilderserie 5.1-5.11 zeigt eine Reihe von Screenshots des Tools Plug-in Builder bei der Erstellung eines Plug-ins für die Lichtsteuerung auf. Die Abbildung 5.1 zeigt das Startbild des Programms. Nach dem Ausfüllen der benötigten Felder, kann das Anlegen mit dem Button „New Plugin“ gestartet werden. Wie in der Abbildung 5.2 dargestellt, wird die Location, Position des zu steuernden Objekts im Living Place, eingetragen. Wie aus der API-Dokumentation der Lichtsteuerung folgt, gibt es eine Möglichkeit das Licht in der Lounge des Living Place zu dimmen und Einzufärben (Menzel, 2012, Seite 5). Für die Steuerung der Lichtintensität ist ein einzelnes Bedienelement „Slider“ gut geeignet. Es wird also eine Gruppe von Bedienelementen

mit dem Namen „Intensity“ (Abbildung 5.3) und nur einem Bedienelement vom Typ „Slider“ (Abbildung 5.4) erzeugt. Wie in der Abbildung 5.4 zu sehen ist, gibt es jetzt die Möglichkeit entweder ein weiteres Bedienelement mit „Add Item“ oder mit „Add Message“ eine Nachricht hinzuzufügen. Ein weiteres Bedienelement wird hier nicht mehr benötigt, weshalb eine Nachricht hinzugefügt wird (Abbildung 5.5). Die Abbildung 5.5 zeigt weitere Möglichkeiten. Mit „next Group“ hat man eine Möglichkeit ein weiteres Steuerungselement an dieser Location hinzuzufügen. Mit „next Node“ hat man dagegen eine Möglichkeit eine weitere Location dem Plug-in hinzuzufügen und mit „done“ die Erstellung zu beenden und ein Dialog zum Speichern der Plug-in-Datei zu öffnen. Da außerdem die Farbsteuerung an der Location „Lounge“ erstellt werden soll, wird mit „next Group“ fortgefahren. Eine neue Gruppe von Bedienelementen „Color“ wird erstellt (Abbildung 5.6). Diese besteht diesmal aus drei Bedienelementen vom Typ „Slider“ und wird durch je eine Farbe der RGB-Palette repräsentiert (Abbildung 5.7 bis 5.9). Die Abbildung 5.10 zeigt das Hinzufügen einer Nachricht an die erstellte Bedienelementgruppe. An dieser Stelle sollte mit dem „next Node“ fortgefahren und die Bedienelemente für weitere Locations erstellt werden. Da diese Prozedur die bereits beschriebene einfach nur wiederholt, wird es hier nicht weiter erläutert. Stattdessen wird an dieser Stelle mit „done“ die Plug-in-Datei gespeichert (Abbildung 5.11).

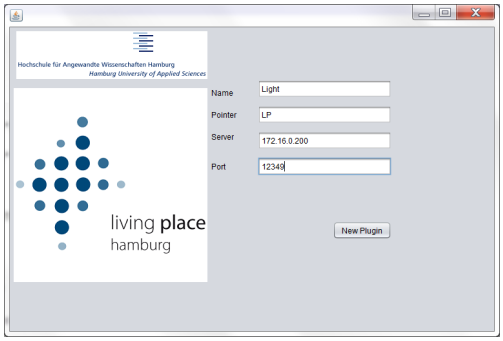


Abbildung 5.1.: Neues Plug-in erzeugen

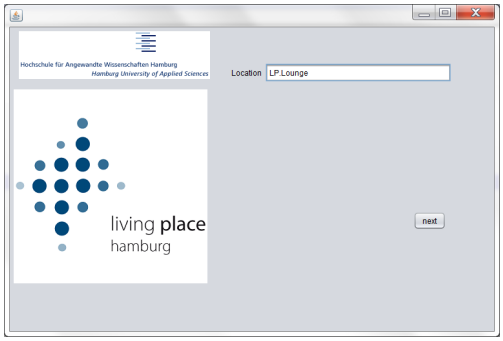


Abbildung 5.2.: Location hinzufügen

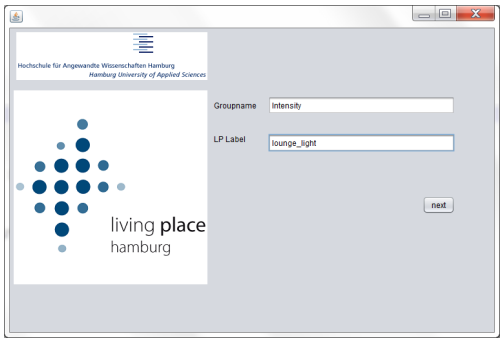


Abbildung 5.3.: Erzeugen einer neuen Bedienelementgruppe

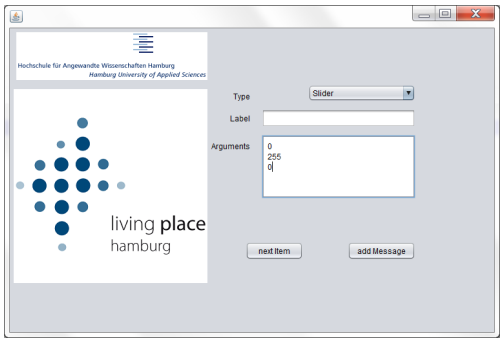


Abbildung 5.4.: Bedienelement „Intensity Slider“ erzeugen

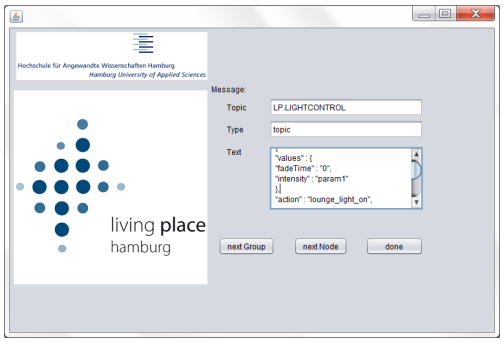


Abbildung 5.5.: Nachricht hinzufügen

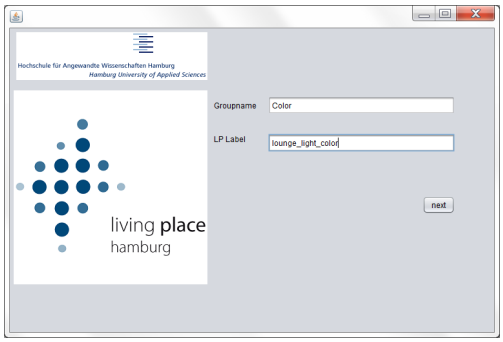


Abbildung 5.6.: Nächste Bedienelement-gruppe erzeugen

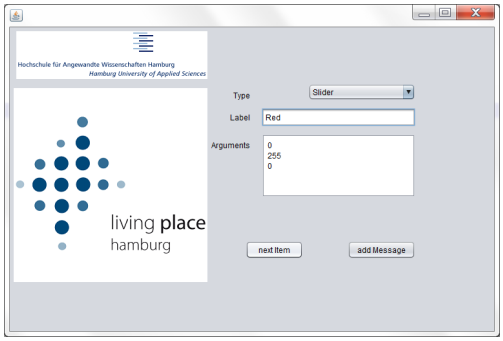


Abbildung 5.7.: Bedienelement „Red Slider“ erzeugen

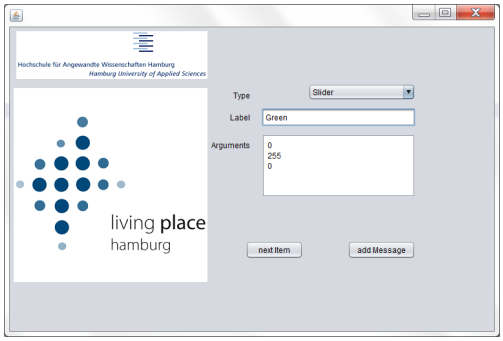


Abbildung 5.8.: Bedienelement „Green Slider“ erzeugen

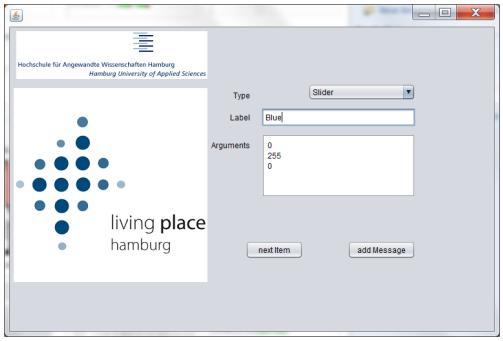


Abbildung 5.9.: Bedienelement „Blue Slider“ erzeugen

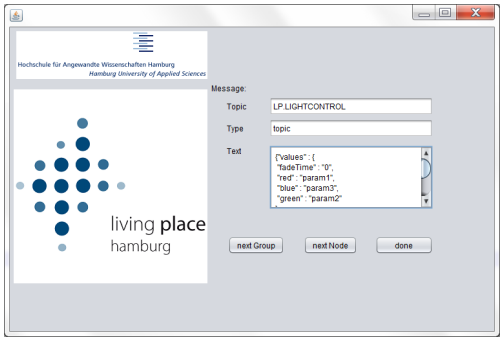


Abbildung 5.10.: Nachricht hinzufügen

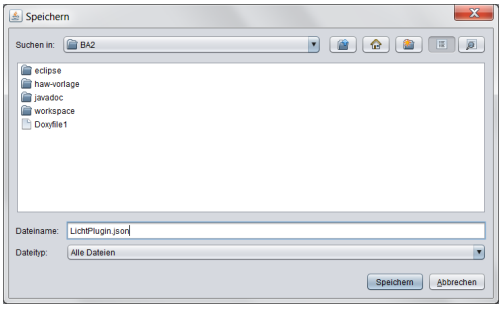


Abbildung 5.11.: Plug-in speichern

Das Listing 5.1 zeigt den Quellcode, welcher vom Plug-in Builder nach dem Speichern der „LichtPlugin.json“ erzeugt wurde und die Abbildung 5.12 das Ergebnis im SmartHomeController.

```
{
  "plugin_name": "Light",
  "plugin_pointer": "LP",
  "plugin_server": "172.16.0.200",
  "plugin_port": 12349,
  "main_array": [{
    "location_path": "LP.Lounge",
    "controls_array": [{
      "control_item": {
        "group_title": "Intensity",
        "lp_label": "lounge_light",
        "group_array": [{
          "group_item_typ": "Slider",
          "group_item_title": "",
          "group_item_argument_array": ["0","255","0"]
        }],
        "message": {
          "message_topic": "LP.LIGHTCONTROL",
          "message_typ": "topic",
          "message_value": {
            "values": {
              "fadeTime": "0",
              "intensity": "param1"
            },
            "action": "lounge_light_on",
            "Id": "controller_1",
            "Version": null
          }
        }
      }
    },
    {
      "control_item": {
        "group_title": "Color",
        "lp_label": "lounge_light_color",
        "group_array": [{
```

```
37          "group_item_typ": "Slider",
38          "group_item_title": "Red",
39          "group_item_argument_array": ["0","255","0"]
40        },
41        {
42          "group_item_typ": "Slider",
43          "group_item_title": "Green",
44          "group_item_argument_array": ["0","255","0"]
45        },
46        {
47          "group_item_typ": "Slider",
48          "group_item_title": "Blue",
49          "group_item_argument_array": ["0","255","0"]
50        }],
51        "message": {
52          "message_topic": "LP.LIGHTCONTROL",
53          "message_typ": "topic",
54          "message_value": {
55            "values": {
56              "fadeTime": "0",
57              "red": "param1",
58              "blue": "param3",
59              "green": "param2"
60            },
61            "action": "lounge_light_color",
62            "Id": "controller_1",
63            "Version": null
64          }
65        }
66      }
67    }]
68  }]
69 }
```

Listing 5.1: Quellcode von LichtPlugin.json

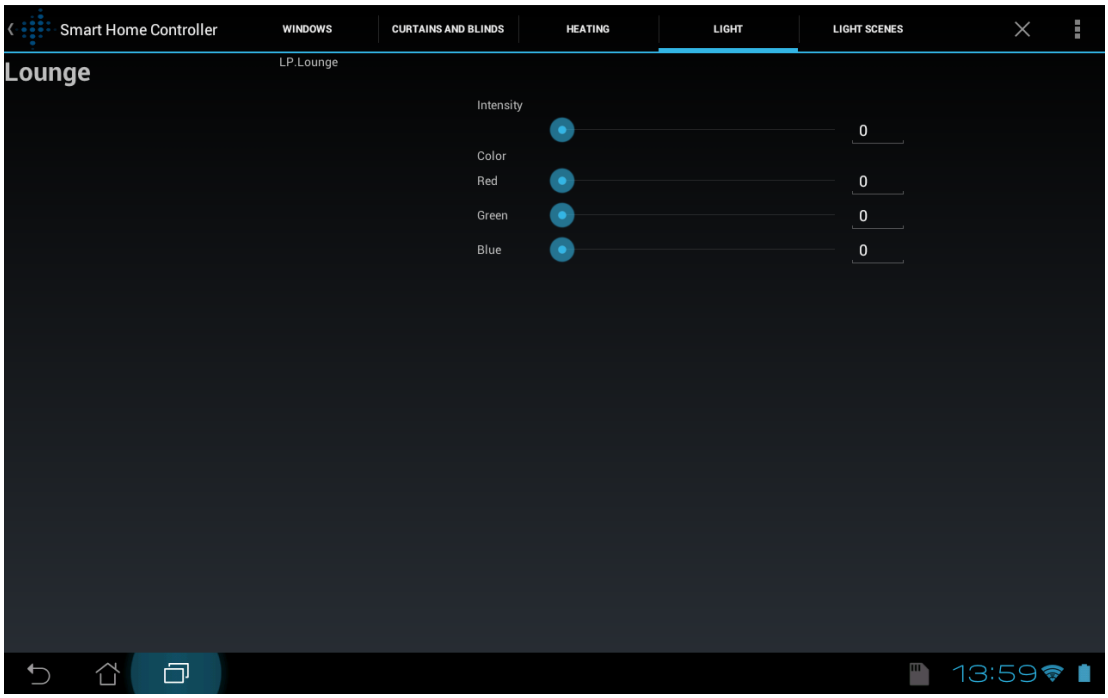


Abbildung 5.12.: Ergebnis der LichtPlugin.json im SmartHomeController

5.2. SmartHomeController

Das Programm „SmartHomeController“ übernimmt das Laden von Plug-ins, das Aufbauen der GUI, sowie die Steuerung der Komponenten. Die gesamte Application besteht aus nur einer Activity. Die Activity nimmt Fragmente auf, welche je einen Tab repräsentieren. Eine speziell für dieses Programm entwickelte Datenstruktur, basierend auf einer Baumstruktur, übernimmt die Abbildung der Plug-in-Dateien auf die interne Objekte und dessen Verwaltung. Helperklassen helfen bei der Arbeit mit dem Dateisystem, während das Unterprogramm Android™ Publisher die Kommunikation mit dem ActiveMQ übernimmt.

5.2.1. Funktionsweise

Bereits im Kapitel 4.1 und in der Abbildung 4.1 wurde ein Prototyp des User Interfaces vorgestellt. Die Abbildung 5.13 zeigt das User Interface der App SmartHomeController, welches im Design dem Prototypen sehr ähnelt.

Abbildung 5.13.: User Interface des SmartHomeControllers, aufgeteilt in neun Bereiche

Das User Interface ist in neun Bereiche aufgeteilt:

1. Navigationsbereich
2. Bedienelemente Bereich
3. Tab Bereich mit unterschiedlichen Plug-ins
4. Navigationsbereich: Taste „zurück, höhere Ebene“, aktueller Root-Node
5. Action Bar: Taste „zurück, höhere Ebene“, Funktion identisch mit Punkt Vier
6. Tatsächlicher Pfad zu den Bedienelementen, die unter Punkt zwei dargestellt sind
7. Action Bar: Taste „Programm Beenden“
8. Action Bar: Taste „Menü“
9. Haupttasten von Android™, haben für das Programm keine Funktion. Das Drücken der Zurück-Taste „beendet“ das Programm und darf nicht wie Tasten unter Punkt vier und fünf benutzt werden.

Das Programm SmartHomeController liest beim Starten die Plug-in-Dateien ein, die sich im Ordner „`/mnt/sdcard/SmartHomeController/Plugins`“ befinden, bildet aus den Plug-ins interne Objekte, organisiert diese und erstellt eine dementsprechende Benutzeroberfläche. Im Falle, dass das Programm die Oberfläche neu aufbauen muss (beim Verschieben in den Hintergrund oder beim Drehen des Bildschirms), werden die internen Objekte serialisiert und im Internal Data Storage (siehe 4.7.3) gespeichert. Beim Beenden des Programms über den „Programm Beenden“-Button, beschrieben unter Punkt sieben, werden die temporären Daten aus dem Internal Data Storage entfernt. Aus diesem Grund muss das Programm über „Programm Beenden“-Button beendet werden. Das Drücken der Android™-Taste „Zurück“ blendet einen Dialog ein, wie die Abbildung 5.14 zeigt, der eine Bestätigung erfordert, falls man das Programm wirklich beenden möchte. Im Falle einer positiven Bestätigung, wird das Programm beendet und die temporären Daten aus dem Internal Data Storage ebenfalls entfernt.

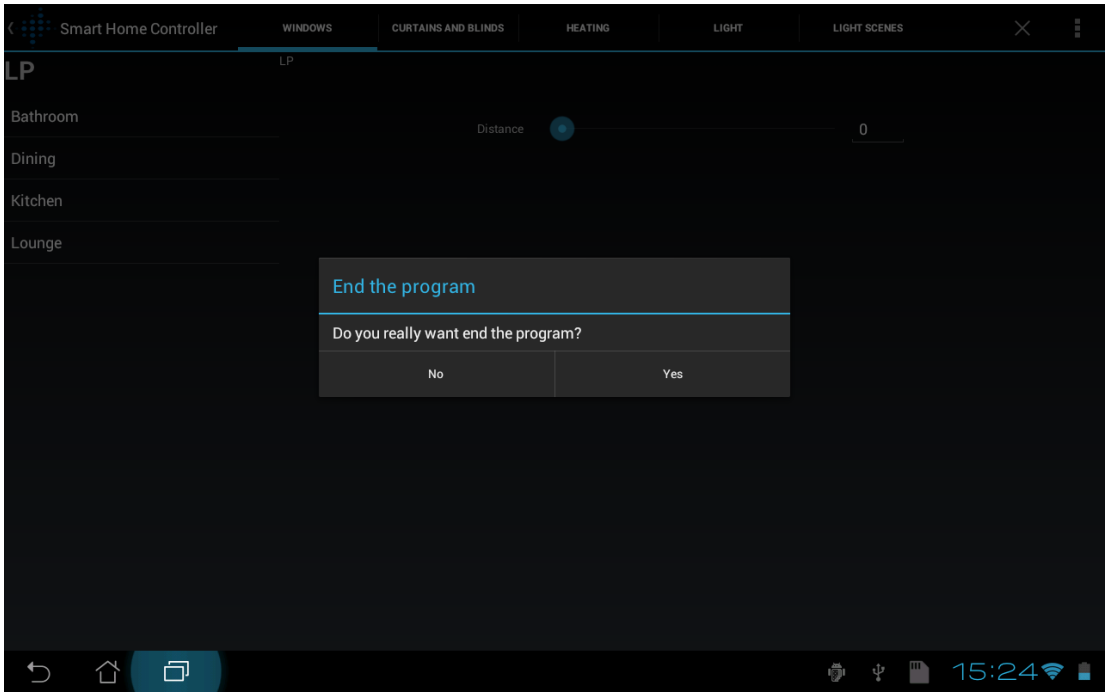


Abbildung 5.14.: Beenden-Dialog

5.2.2. Verwendete Designmuster

Im Programm wurden im Wesentlichen zwei Designmuster verwendet. Das erste Designmuster ist das „Observer Pattern“, welches dazu genutzt wird die Zustandsänderungen einzelner Bedienelemente dem Programm mitteilen zu können.

Die Abbildung 5.15 zeigt die UML-Darstellung der Klasse ElementGroup. Die Klasse ElementGroup implementiert das `UpdateIF`-Interface und somit die Methode `update()`. Außerdem besitzt ElementGroup eine Liste `groupelements` mit Elementen vom Datentyp `ControlItemIF`. An jedem Element der Liste registriert sich ElementGroup als Observer und wird über die Änderungen mit Hilfe von `update()` informiert.

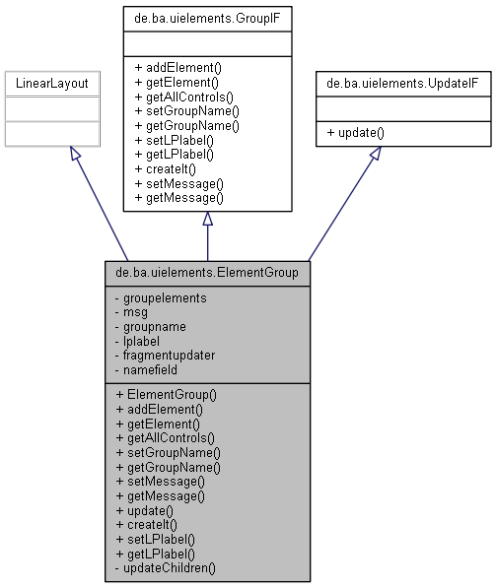


Abbildung 5.15.: UML-Darstellung der Klasse ElementGroup

Die Abbildung 5.16 zeigt die UML-Darstellung der Klasse MyButton. Die Klasse MyButton implementiert das `ControlItemIF`-Interface und somit die Methoden `addObserver()` und `notifyObserver`. Die Methode `addObserver()` bietet für die Klasse ElementGroup und andere Klassen, die das `UpdateIF`-Interface implementieren, die Möglichkeit sich bei myButton als Observer registrieren zu können. Eine der Membervariablen der Klasse MyButton ist „button“ vom Typ `android.widget.Button`. Beim Betätigen des „button“ informiert das Android™-System die MyButton-Klasse mit Hilfe eines `OnClickListeners` über die Aktion und ruft die `notifyObserver()`-Methode auf, die intern jeden Observer über seine `update()`-Methode über die Zustandsänderung informiert.

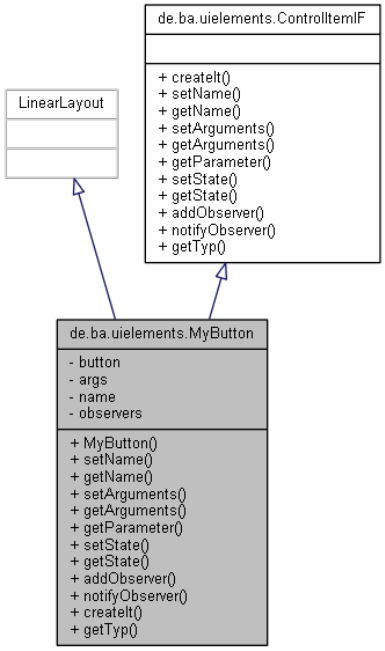


Abbildung 5.16.: UML-Darstellung der Klasse MyButton

Das zweite Designmuster ist das „Factory Pattern“, welches das Erweitern des Programms mit zusätzlichen Bedienelementen ermöglicht. Die genauere Beschreibung folgt im Kapitel 5.2.3.

5.2.3. Erweiterung der Bedienelemente

Die App SmartHomeController bietet die Möglichkeit das Programm mit weiteren Bedienelementen zu erweitern. Zur Zeit sind bereits folgende Bedienelemente implementiert:

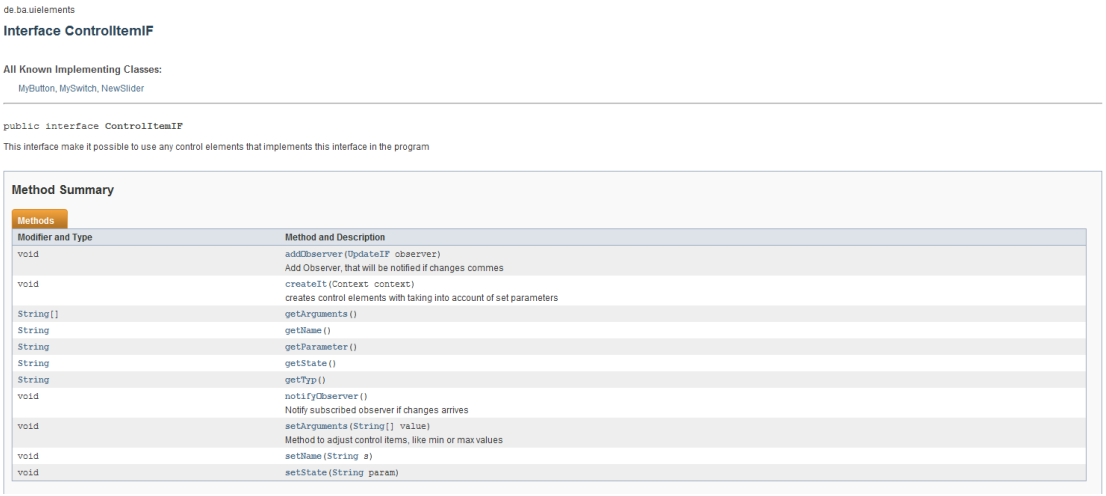
- Button
  ist ein einfacher Knopf, der beim Betätigen eine Nachricht an das ActiveMQ versendet
- Switch
  ist eine Art Toggle-Button mit zwei Zuständen, beim Betätigen wird der Wert in der Nachricht je nach Zustand manipuliert und an das ActiveMQ versendet
- Slider
  ist ein Schieberegler, der die Nachricht entsprechend dem Wert der Skala manipuliert und an das ActiveMQ versendet. Wertebereich und Startwert können frei eingestellt werden.

Auf den ersten Blick scheint es so, dass die obengenannte Bedienelemente Standard-Bedienelemente eines Android™ Systems sind. Beim genaueren analysieren der Klassen aber, stellt man fest, dass diese etwas komplexer sind. Button und Switch, sind zwei Klassen, die nicht von `android.widget.Button` oder `android.widget.Switch` abgeleitet sind, sondern von `android.widget.LinearLayout` und intern ein Textfeld für die Bezeichnung, sowie ein Button bzw. Switch verwalten. Diese Vorgehensweise ermöglicht die Implementierung von zusammengesetzten oder völlig neuen, eigenen Bedienelementen. Das Bedienelement Slider besteht beispielsweise intern aus einer `android.widget.Seekbar`, einem `android.widget.TextView` und einem `android.widget.EditText` und verwaltet diese.

Grundsätzlich reichen Bedienelemente Button und Slider zum Steuern der meisten Komponenten aus. Manchmal jedoch ist ein anderes Bedienelement bequemer, passt besser ins Gesamtkonzept oder erleichtert die Steuerung. Um das nachträgliche Hinzufügen der Bedienelemente zu ermöglichen, werden alle Bedienelemente mit Hilfe eines Interfaces, die dieses implementieren, verallgemeinert, so dass diese aus der Sicht des Programms gleich sind. Um ein Bedienelement nachträglich implementieren zu können muss folgendermaßen vorgegangen werden:

1. Bedienelement muss das „ControlItemIF“-Interface implementieren
2. „ControlFactory“-Klasse um entsprechendes Statement zur Erzeugung des konkreten Bedienelements erweitert

Die Abbildung 5.17 zeigt einen Auszug der Projekt-Dokumentation mit der Beschreibung der Methoden des ControlItemIF.



de.ba.uielements

Interface ControlItemIF

All Known Implementing Classes:

MyButton, MySwitch, NewSlider

```
public interface ControlItemIF
```

This interface make it possible to use any control elements that implements this interface in the program

Method Summary

Methods

Modifier and Type	Method and Description
void	addObserver(UpdateIF observer) Add Observer, that will be notified if changes commes
void	createIt(Context context) creates control elements with taking into account of set parameters
String[]	getArguments()
String	getName()
String	getParameter()
String	getState()
String	getTyp()
void	notifyObserver() Notify subscribed observer if changes arrives
void	setArguments(String[] value) Method to adjust control items, like min or max values
void	setName(String s)
void	setState(String param)

Abbildung 5.17.: Auszug der Projekt-Dokumentation mit der Beschreibung der Methoden des ControlItemIF

Außer der sogenannten „getter“- und „setter“-Methoden, die zur Serialisierung, sowie Deserialisierung oder zum Abfragen des aktuellen Zustands genutzt werden, enthält das Interface folgende wichtige Methoden, die hier weiter erläutert werden:

- **addObserver** und **notifyObserver**
  Diese Methoden sind ein Teil des Observer Pattern und werden zur Benachrichtigung des Programms über die Zustandsänderungen des Bedienelements verwendet.
- **createIt**
  Diese Methode erstellt das Bedienelement unter Berücksichtigung aller Parameter und Argumente und zeigt diesen an.

Folgende Parameter werden per „getter“- und „setter“-Methoden gesteuert und für folgende Aufgaben genutzt:

- Name
  Bezeichnung des Bedienelements z.B. „Licht aus“
- Type
  Typ des Bedienelements, welches in der „ControlFactory“-Klasse verwendet wird (zur Zeit: Button, Switch, Slider)

- State
  Der aktuelle Zustand des Bedienelements
- Arguments
  Ein String-Array der jegliche Einstellungen für das Bedienelement enthält, wie min-Wert oder max-Wert
- Parameter
  Der aktuelle Zustand des Bedienelements als fertiges Parameter für das ActiveMQ

5.3. Fazit

Mit den beiden Programmen Plug-in Builder und SmartHomeController sind alle funktionale und nicht funktionale Anforderungen, die im Kapitel 3.3.2 definiert wurden, erfüllt. Die Funktionalität des SmartHomeControllers kann über die Plug-ins erweitert werden. Im Rahmen des Systemtests wurden bereits Plug-ins für die Lichtsteuerung, Fenstersteuerung, für die Gardinen und Rollos, Heizung, sowie für die Steuerung der Lichtszenen, erstellt und während des System-, sowie des Abnahmetests erfolgreich getestet. Zudem wurde eine Möglichkeit geschaffen eigene Bedienelemente nachträglich zu implementieren, was die Flexibilität der Software nochmals erhöht hat. Obwohl die Plug-ins in der für den Menschen sehr gut lesbaren Form, der JSON-Notation, formuliert werden, wurde Plug-in Builder implementiert, ein zusätzliches Tool, welches die Erstellung von Plug-ins erleichtert.

5.3.1. Vorschläge

Auch wenn mit den beiden Programmen eine gute Basis geschaffen wurde, kann man diese erweitern und noch weiter optimieren. Denkbare Szenarien wären zum Beispiel:

- Erweiterung des Plug-in Builders bezüglich der Änderung bzw. Editierung der bestehenden Plug-ins
- Implementierung von weiteren Bedienelementen, z.B. einer Farbpalette zur Auswahl der Farbe des Lichts
- Erweiterung des SmartHomeControllers um die Funktionalität des Plug-in Builders
- Implementierung des Lokalisationservices, der Bedienelemente des Plug-ins abhängig von der aktuellen Position im Living Place aufbaut. Bastian Karstaedt befasste sich

während seiner Masterarbeit:„Kontextinterpretation in Smart Homes auf Basis 3D semantischer Gebäudemodelle“ mit der Realisierung des *Indoor Spatial Information Service*, welches zur Lokalisierung im Living Place verwendet werden kann. (Karstaedt, 2012)

- Optimierung des SmartHomeControllers für den Einsatz auf dem Smartphone

6. Schluss

6.1. Zusammenfassung

Im Rahmen dieser Arbeit wurde untersucht, ob es möglich ist eine Software zu entwickeln, die in der Lage ist die Steuerung der Installationen im Living Place an der HAW-Hamburg zu übernehmen. Dabei wurden unterschiedliche Steuerungskonzepte der Komponenten in einer Software vereint, sowie eine Möglichkeit geschaffen die Software mittels Plug-in-Verfahren erweitern zu können.

Es wurden Szenarien und Rahmenbedienungen analysiert und Anforderungen formuliert. Die Erweiterung der Software per Plug-in, sowie eine einfache Erstellung dieser, sind dabei mit die wichtigsten Anforderungen. Eine für die Realisierung passende Hardware-, sowie Softwareplattform wurde festgelegt und die Machbarkeit geprüft. Dabei wurde festgestellt, dass ein Tablet PC mit Android™ OS Ice Cream Sandwich eine sehr gute Basis für die Umsetzung des Projektes bietet.

Ein GUI-Design wurde formuliert und mögliche Aktion-Reaktion-Szenarien überlegt. Dabei wurde auf Erfahrung anderer Programme, wie z.B. Windows Explorer gesetzt und die „Exploreransicht“ als Grundlage verwendet. Die Kommunikationsmöglichkeiten zwischen den Komponenten des Living Place wurden diskutiert und eine Entscheidung für das Einsetzen eines Proxy getroffen. Die Möglichkeit der Verwendung eines Plug-in Frameworks wurde analysiert. Dabei wurde entschieden ein eigenes Plug-in Modell einzusetzen. Es wurden unterschiedliche Plug-in Formate miteinander verglichen, wobei sich die Verwendung von JSON durchgesetzt hat. Der Inhalt eines Plug-ins wurde definiert und die „Punktnotation“ als Lösung für die Darstellung der Hierarchie in den Plug-ins festgelegt. Unterschiedliche Möglichkeiten der persistenten Datenspeicherung in einem Android™ System wurden untersucht und die Serialisierung mit Speicherung in den nicht flüchtigen Speicher als eine bessere Alternative befunden. Die Problematik der Konsistenz der Daten im Living Place wurde erläutert, einige mögliche Lösungen vorgestellt und die „Notlösung“ mittels Mockups festgelegt.

Auf Grundlage der getroffenen Entscheidungen wurde das Programm SmartHomeController entwickelt, welches ein Plug-in Host ist. Die Funktionalität des Programms wird über die

Plug-ins definiert, die ohne gesonderte Programmierkenntnisse erstellt werden können. Um die Erstellung der Plug-ins unterstützen zu können, wurde das Tool Plug-in Builder implementiert. Zwecks Evaluierung wurden Plug-ins für alle bestehende Komponente des Living Place erstellt und die Funktionalität des Programms im Funktionstest erfolgreich getestet.

6.2. Gesammelte Erfahrungen

Die Arbeit hat gezeigt, dass es durchaus möglich ist ein generisches Programm zu entwickeln, welches unterschiedlichste Kommunikationsformate vereint und dadurch ermöglicht viele Komponenten mit einem einzigen Programm steuern zu können.

Jedoch wurde auch festgestellt, dass das Fehlen von Regeln die Universalfähigkeit sehr erschwert. Bei der Entwicklung einer plug-in-fähigen Anwendung ist es signifikant mit Hilfe verschiedener Regeln einen Rahmen zu schaffen, der alle Plug-ins verallgemeinert und damit die Interaktion mit dem Plug-in Host für jedes Plug-in gleich macht. Die Installationen im Living Place wurden von vielen unterschiedlichen Entwickler zu unterschiedlicher Zeit erstellt, ohne das Aspekt einer universeller Fernbedienung in Betracht gezogen zu haben. Damit wurden lediglich zwei Regeln beachtet, die für die Kommunikation im Living Place wichtig sind. Zu einem die Kommunikation über ActiveMQ, zum Anderen JSON als Nachrichtenformat. Diese Umstände haben das Design der Plug-ins erschwert.

Ein weiteres Problem stellt die fehlende Funktionalität mancher Installationen im Living Place dar. Das Fehlen der Zustandsabfragefunktion mancher Komponenten, macht das Programm SmartHomeController leider unvollständig und nur bedingt produktiv einsetzbar. Dieser Umstand macht die Entwicklung eines Zustandsabfrageservices im Living Place unentbehrlich. Die Entwicklung eines solchen im Rahmen dieser Arbeit, stellt sich wegen der Komplexität als nicht möglich heraus.

Die während der Arbeit gesammelte Erfahrung, hat gezeigt, dass Android™ ein sehr leistungsfähiges Betriebssystem mit einer hohen Gerätevielfalt ist. Durch den Open Source Aspekt und sehr gute Dokumentation, gestaltet sich die Entwicklung für dieses Betriebssystem als äußerst attraktiv.

6.3. Ausblick

Die „Smartisierung“ hat schon längst begonnen. Noch vor ein paar Jahren war der Begriff „Smart“ für viele lediglich eine Automarke. Heutzutage hört man viele Begriffe, in denen Smart vorkommt. Smartphone, SmartTV, Smart Home sind einige davon. Smart ist ein Synonym

für clever. Der Markt für clevere Technik boomt. Mehr und mehr clevere Geräte werden in den Wohnungen und Häusern miteinander vernetzt, was letztendlich ein Smart Home ausmacht. Die Smart Home Technik wird in den nächsten Jahren höchstwahrscheinlich aus den Informatiklaboren den Einzug in das Eigenheim finden. Durch Projekte, wie Android™@Home, werden auch mobile Betriebssysteme, wie Android™, eine bedeutende Rolle auf dem Smart Home Markt spielen und somit auch dem SmartHomeController ähnliche Programme an Bedeutung gewinnen.

Die Verwendung des SmartHomeControllers außerhalb des Living Places ist eher unwahrscheinlich, da diese Lösung zu sehr auf die Infrastruktur des Living Place optimiert ist. Das Konzept der Software hat jedoch gezeigt, dass es durchaus möglich ist generische Programme zu entwickeln, die auch von Menschen ohne Programmierkenntnisse erweitert werden können. Im Zusammenhang mit Accessory Development Kit von Google und dem Arduino kann dieses Konzept eine echte Alternative zu den fertigen, aber meist nur Insellösungen, aus der Industrie darstellen.

A. Technische Mittel

Folgende technische Mittel wurden für die Erstellung dieser Arbeit verwendet:

- **Textverarbeitungsprogramm**
  - TEXnicCenter 2.0 Alpha 4
  - MikTEX2.9
- **Entwicklungsumgebung „SmartHomeController“**
  - Eclipse Indigo Service Release 1
  - Android™ SDK Tools Revision 20.0.1
  - Android™ SDK Plattform API-Level 16 Revision 2
  - JavaJDK 1.7.0
- **Hardwareplattform „SmartHomeController“**
  - Asus Eee Pad Transformer TF-101
  - Android™ 4.0.3 Betriebssystem
- **Entwicklungsumgebung „Plug-in Builder“**
  - NetBeans IDE 7.1.1
  - JavaJDK 1.7.0
- **Betriebssystem**
  - Windows 7 64 Bit Service Pack 1

B. Inhalt der CD

Die beiliegende CD enthält folgendes Material:

- Diese Arbeit als PDF-Datei
- Eclipse-Projekt „SmartHomeController“ als *.zip
- Eclipse-Workspace
- DoxyGen-Dokumentation des Projekts „SmartHomeController“
- JavaDoc-Dokumentation des Projekts „SmartHomeController“
- SmartHomeController.apk
- Plug-in Dateien
- NetBeans-Projekt „Plug-in Builder“

Literaturverzeichnis

[Allen und Murphy 2012] Allen, Grant ; Murphy, Mark: *Beginning Android 4*. Apress, 2012. – ISBN 1-430-23984-0

[Becker und Pant 2009] Becker, Arno ; Pant, Marcus: *Android, Grundlagen und Programmierung*. Auflage 1. dpunkt.verlag GmbH, 2009. – ISBN 978-3-89864-574-4

[Bernin 2011] Bernin, Arne: *Einsatz von 3D-Kameras zur Interpretation von räumlichen Gesten im Smart Home Kontext*, Hamburg University of Applied Sciences, Masterarbeit, 2011

[Bornemann 2011] Bornemann, Sven B.: *Android-basierte Smart Home Interaktion am Beispiel einer Gegensprechanlage*, Hamburg University of Applied Sciences, Bachelorarbeit, 2011

[Haiges 2011] Haiges, Sven: *Android Schnelleinstieg*. entwickler.press, 2011. – ISBN 978-3-86802-067-0

[Hewett u. a. 2007] Hewett ; Baecker ; Card ; Carey ; Gasen ; Mantei ; Perlman ; Strong ; Verplank: ACM SIGCHI Curricula for Human-Computer Interaction / ACM SIGCHI. ACM SIGCHI, Juli 2007. – Forschungsbericht. – URL http://old.sigchi.org/cdg/cdg2.html. ACM: Association for Computing Machinery ; SIGCHI: Special Interest Group on COMPUTER-HUMAN INTERACTION

[Hußmann 2006] Hußmann, Prof. Dr. H.: *Designing Interactive Systems I*. Vorlesungsskript. Dezember 2006. – URL http://videoonline.edu.lmu.de/node/1027/367272. – Vorlesung Mensch-Maschine-Interaktion an der Ludwig-Maximilians-Universität München

[Karstaedt 2012] Karstaedt, Bastian: *Kontextinterpretation in Smart Homes auf Basis 3D semantischer Gebäudemodelle*, Hamburg University of Applied Sciences, Masterarbeit, 2012

[Komatineni und MacLean 2012] Komatineni, Satya ; MacLean, Dave: *Pro Android 4*. Apress, 2012. – ISBN 1-430-23930-1

[Lee 2012] Lee, Wei-Meng: *Beginning Android 4 Application Development*. Wrox, 2012. – ISBN 1-118-19954-5

[of Living Place] Living Place, Members of: *Living Place Wiki*. – URL http://livingplace.informatik.haw-hamburg.de/wiki/index.php/Hauptseite

[von Luck u. a. 2010] Luck, Prof. Dr. K. von ; Klemke, Prof. Dr. G. ; Gregor, Sebastian ; Rahimi, Mohammad A. ; Vogt, Matthias: Living Place Hamburg – A place for concepts of IT based modern living / Hamburg University of Applied Sciences. HAW Hamburg, Mai 2010. – Forschungsbericht. – URL http://livingplace.informatik.haw-hamburg.de/content/LivingPlaceHamburg_en.pdf. no notes

[Menzel 2012] Menzel, Jan: *Lichtsteuerung API Dokumentation*. 06.01.2012. HAW Hamburg: , Januar 2012. – URL http://livingplace.informatik.haw-hamburg.de/svn/LP-LightControl/doc/API%20Dokumentation.pdf

[Mosemann und Kose 2009] Mosemann, Heiko ; Kose, Matthias: *Android, Anwendungen für das Handybetriebssystem erfolgreich programmieren*. Carl Hanser Verlag München Wien, 2009. – ISBN 978-3-446-41728-1

[Ostrander 2012] Ostrander, Jason: *Android UI Fundamentals*. Develop and Design. Peachpit Press, 2012. – ISBN 0-321-81458-4

[Strese u. a. 2010] Strese, Hartmut ; Seidel, Uwe ; Knape, Thorsten ; Botthof, Alfons: Smart Home in Deutschland
Untersuchung im Rahmen der wissenschaftlichen Begleitung zum Programm Next Generation Media (NGM) des Bundesministeriums für Wirtschaft und Technologie / Institut für Innovation und Technik (iit) in der VDI/VDE-IT. Institut für Innovation und Technik, Mai 2010. – Forschungsbericht. – URL www.iit-berlin.de. ISBN: 978-3-89750-165-2

[W3C 2008] W3C (Veranst.): *Extensible Markup Language (XML) 1.0 W3C Recommendation*. Fifth Edition. 2008. – URL http://www.w3.org/TR/2008/REC-xml-20081126/

[Weiser 1991] Weiser, Mark: The Computer for the 21st Century. In: *Scientific American* (1991), September

[YAML™ 2009] YAML™ (Veranst.): *YAML Ain't Markup Language (YAML™) Version 1.2*. Third Edition. 2009. – URL http://yaml.org/spec/1.2/spec.html

[Zollondz 2011] Zollondz, Alexander: *Android@Home: Google entwickelt Heimsteuerung*. Mai 2011. – URL http://www.netzwelt.de/news/86625-android-home-google-entwickelt-heimsteuerung.html. – abgerufen am 15.07.2012

Glossar

ADK Accessory Development Kit, Seite 27

API Application Programming Interface, Seite 3

HAW Hochschule für Angewandte Wissenschaften, Seite 13

JDK Java Development Kit, Seite 24

JMS Java Message Service, Seite 16

JPF Java Plug-in Framework, Seite 33

JSON Java Script Object Notation, Seite 17

MOM Message Oriented Middleware, Seite 16

OSGi Open Services Gateway initiative, Seite 34

SDK Software Development Kit, Seite 24

SGML Standard Generalized Markup Language, Seite 35

UI User Interface, Seite 20

W3C World Wide Web Consortium, Seite 36

XML Extensible Markup Language, Seite 35

YAML Ain't Markup Language, Seite 35

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

Hamburg, 22.08.2012 Viktor Kolbaja