Wahlpflichtkurs, Wintersemester 2012/2013, HAW Hamburg

Projektplan

Smart Home Control

Nils Feyerabend, Tobias Meurer
29.01.2013

Projektplan

Smart Home Control

Inhalt

Dokument-Historie

Bearbeiter	Anpassungen	Datum	Version
TM	Erstellen des Dokuments	19.11.2012	0.1
TM	Anpassungen und Ergänzungen auf Basis der Aufgabenstellung fürs Praktikum	22.11.2012	0.2
TM, NF	Ergänzung im gesamten Dokument	26.01.2013	0.3
TM, NF	Ergänzung im gesamten Dokument	28.01.2013	0.4
TM, NF	Ergänzung im gesamten Dokument	29.01.2013	0.5

Team

Das Team besteht aus folgenden zwei Personen:

- Nils Feyerabend (nils.feyerabend@haw-hamburg.de)
- Tobias Meurer (tobias.meurer@haw-hamburg.de)

Motivation

Autor: NF

Unsere größte Motivation dieses Modul zu wählen war neben dem Livingplace das Androidframework.

Beide von uns haben Interesse für Androidgeräte zu entwickeln, hatten aber leider noch nicht die Zeit und die Hardware um uns damit zu beschäftigen. Also war dieses Modul ein perfekter Einstieg in die das doch sehr umfangreiche Framework.

Ein weiterer Pluspunkt war, dass Endgeräte gestellt wurden, so konnten wir gleichzeitig das Samsung Galaxy Tab 2 testen.

Der letzte aber nicht der unwichtigste Punkt war außerdem, dass wir für den Livingplace entwickeln durften, dass schönste und auch spannendste Labor der HAW.

Verantwortungsbereiche

Autor: TM, NF

- Projektplan (NF, TM)
- GUI-Mockup (NF)
- MessageAdapter (TM)
- ButtonFactory (NF)
- Context und Controller Enums (TM)
- ColorPicker (TM, NF)

Zielsetzung und Aufgabenstellung

Projektbeschreibung

Autor: TM

Im Laufe des Wahlpflicht Moduls „Smart Home Control“ soll eine Fernbedienung für den Living Place für ein Android Tablet erstellt werden.

Die Fernbedienung soll das Licht, die Fenster, die Gardinen und die Rollos der Wohnung steuern können. Der Benutzer hat die Möglichkeit über die Navigation auf der linken Seite der App einen Raum auszuwählen, für den er etwas steuern möchte. Daraufhin werden in der Actionbar die in diesem Raum verfügbaren Arten von steuerbaren Elementen (Licht, Rollos, ...) angezeigt.

Darüber hinaus soll anhand eines Sensors die aktuelle Position des Benutzers ermittelt werden, sodass sich die Oberfläche der Android-App dem Raum anpasst, in dem sich der Benutzer aktuell befindet. Die Ermittlung der aktuellen Position erfolgt kontinuierlich. Das Umschalten der Oberfläche läuft jedoch mit einer Verzögerung von mehreren Sekunden nach der letzten Eingabe des Benutzers und nach dem letzten Wechsel der Position im Raum. Diese Verzögerung soll verhindern, dass die Oberfläche der App sich verändert, während der Benutzer mit der App interagiert.

Funktionale Anforderungen

Autor: TM

- Bedienbarkeit folgender Element-Arten möglich:
  - Licht
  - Rollos
  - Gardienen
  - Fenster
  - Heizung
- Gliederung der steuerbaren Elemente nach Raum und Art
- Optional: Speichern der fünf zuletzt ausgewählten und favorisierten Licht-Farben
- Optional: Automatische Anpassung der Oberfläche auf Basis der Position des Benutzers in der Wohnung

Technische, Nicht-Funktionale Anforderungen

Autor: TM, NF

- Lauffähig auf Tablets ab Android 4.0.3
- nicht Abwärtskompatibel
- Intuitive Bedienung
- Einfache Erweiterbarkeit

Zeitplanung

Praktikumsziele

Autor: TM, NF

KW 45 – 09.11.2012

- Prototyp erstellen, um Nachrichten an die Message Queue des Living Place zu senden und Ereignisse (Licht, Gardinen, Fenster, Rollos steuern) auszulösen.

KW 47 – 23.11.2012

- Zeitplan und Projektplan aufstellen.
- GUI-Konzept erstellen

KW 49 – 07.12.2012

- Nachrichten Komponente mit JASON-Wrapper für Licht-Messages erstellen
- GUI-Konzept umsetzen

KW 51 – 21.12.2012

- Layouts zu Tabs zuordnen
- Button-Factory erstellen
- Für Raum-Context Bezeichnungen aus Strings.xml auslesen

KW 55 – 18.01.2013

- Nachrichten Komponente für andere Nachrichten fertigstellen
- Finalisieren der App
- Testen der App

KW 55+

- Fehlerkorrektur
- Finalisieren der App
- Optionale Implementierungen

Entwurf

Systemarchitektur (beide)

Teilsysteme des LivingPlace? Welche Elemente wir ansteuern? Was genau soll hier aufgeführt werden?

Softwarearchitektur

Allgemein

Autor: TM

Benutzt wurde:

- LPControlTabSample:
  - Dient als Vorlage für unsere App
  - Quelle: https://pub.informatik.haw-hamburg.de/home/pub/prof/wendholt/wpsmarthome/
- AndroidPublisher:
  - Versenden von Nachrichten
  - Quelle: http://livingplace.informatik.haw-hamburg.de/content/DoorBell/AndroidPublisher.jar
- ColorPicker:
  - Auswählen von Farben für Licht
  - Quelle: Android API Demos
    - Android SDK-Manager starten,
    - Android 4.0.3 (API 15) Samples for SDK installieren
    - %AndroidInstallDirectory%\android-sdk\samples\android-15\ApiDemos\src\com\example\android\apis\graphics\ColorPickerDialog.java

Alle weiteren Bestandteile der Software wurden in Eigenarbeit erstellt.

Button Factory

Autor: NF

Die Button Listener Factory kapselt das Erzeugen der Listener für die einzelnen Buttons. Das Ziel der Factory ist es bei Änderungen oder Erweiterungen an der Funktionalität der Fernbedienung einen single point of control für die Listener zu erschaffen.

Dazu werden folgende Informationen vom ControlFragment benötigt.

- Control
- View
- Room

Im ersten Schritt wird über die Enumeration Control identifiziert für welches Layout Listener erstellt werden sollen.

Anschließend wird über die View für ein bestimmtes Layout alle Buttons gesucht und mit einem Listener versehen.
Welcher Button für welchen Raum zuständig ist geschieht über die Enumeration Room. Im Listener abhängig von dieser Variable eine Message erzeugt die beim auslösen des Events über den Message Adapeter an die ActiveMQ des Living Places geschickt wird.

MessageAdapter

Autor: TM

Der *MessageAdapter* dient dem Erstellen und Versenden von Nachrichten zur Steuerung der Geräte im LivingPlace.

Der *MessageAdapter* beinhaltet das Interface *Message*, welches von Nachrichten-Klassen für die steuerbaren Geräte-Klassen des LivingPlace implementiert wird:

- Fenster: *WindowMessage*
- Gardienen: *CurtainMessage*
- Heizung: *HeatingMessage*
- Licht: *LightMessage*
- Rollos: *BlindsMessage*

Ein weiterer Bestandteil des MessageAdapters ist die Utility-Klasse *Messages*. Diese Klasse dient dem Erstellen von Message-Objekten. Da die Konstruktoren der o.g. Message-Klassen mit keinem Access Modifier versehen sind und daher außerhalb des Packages nicht sichtbar sind, ist die Utility-Klasse *Messages* die einzige Möglichkeit, um neue Nachrichten von außerhalb des Packages zu erzeugen. *Messages* stellte eine Reihe von Methoden zur Verfügung, um diese Message-Objekte zu erzeugen.

Eine weitere Utility-Klasse des *MessageAdapter*s ist die Klasse *MessageSender*, welche dem Versenden von Nachrichten dient. Diese Klasse beinhaltet die private Klasse *MessageSender.Sender*, welche *android.os.AsyncTask* implementiert. MessageSender verwendet MessageSender.Sender, um die Nachrichten asynchron versenden zu können und damit den Ablauf des Main-Threads des Programms nicht zu blockieren, währende die Nachrichten gesendet werden.

MessageSender.Sender greift zum Senden der Nachrichten auf die Libary AndroidPublisher zu, welche den Verbindungsaufbau und das eigentliche Übertragen der Nachricht an das ActiveMQ-System des LivingPlace vornimmt.

Der letzte Bestandteil des MessageAdapters ist die Klasse *Values*. Hier sind in einigen Konstanten Konfigurations-Daten für den MessageAdapter gespeichert. Dazu zählen unter anderem die IP-Adresse und der Port des ActiveMQ-Servers.

Enum-Klassen Room und Control

Autor: TM

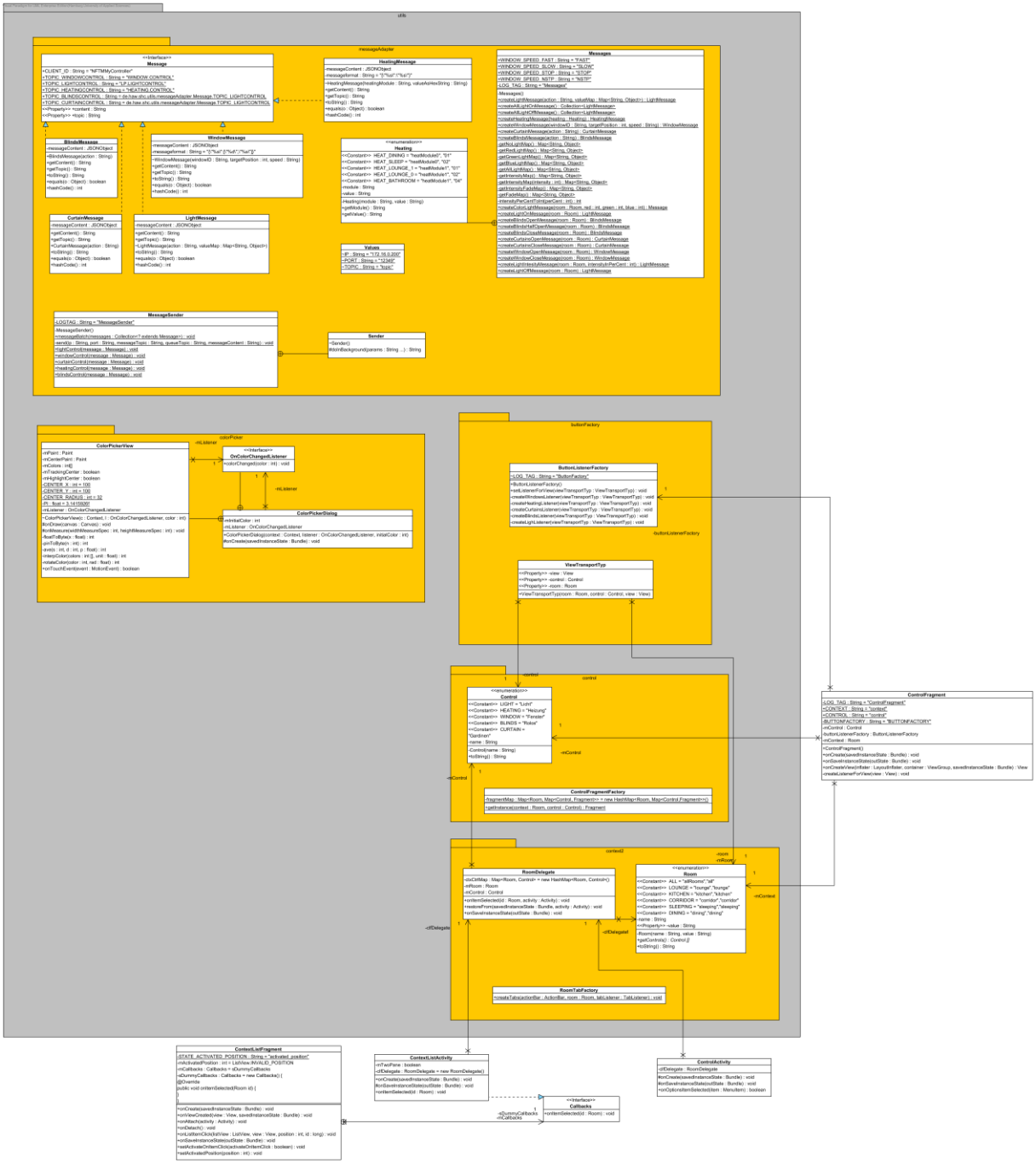
Die Klassen Room und Control wurden weitestgehend aus der Vorlage LPControlTabSample übernommen, jedoch erweitert.

Die Klasse Room ist eine Aufzählung aller Räume der Wohnung und befindet sich im Package context, während die Klasse Control, welche die ansteuerbaren Geräte-Klassen des LivingPlace (Fenster, Gardinen, Heizung, Licht, Rollos) auflistet, im Package Control zu finden ist.

Diese beiden Klassen werden zum einen verwendet, um im GUI die Räume und Geräte-Klassen aufzulisten. Dazu werden die Bezeichnungen der Räume und Geräte aus der Ressource-Datei Strings.xml ausgelesen. Zum anderen werden die beiden Enums verwendet, um die korrekten Kombination aus Raum und dem zu steuernden Geräteklasse zu ermitteln und anschließend die richtigen Nachrichten zu erstellen.

Klassenmodell

Autor: NF



GUI-Konzept

Autor: TM

Navigation

Für die Navigation haben wir uns dazu entschlossen, auf der linken Seite ein ListView zu verwenden, über das der entsprechende Raum ausgewählt werden kann. In der ActionBar werden Tabs für die verschiedenen Räume angezeigt. Beim Auswählen einer Kombination aus Raum und Geräte-Klasse wird ein entsprechendes Layout mit den steuerbaren Elementen ausgewählt.

Layout und Designaufwände

Bei den verwendeten Layouts handelt es sich zum Größten Teil um Linear-Layouts, die verwendet wurden, um die Position der Bedienelemente möglichst genau anzugeben.

Die Layouts wurden mit dem GUI-Editor von IntelliJ IDEA entworfen. Die Layouts für die jeweiligen Geräte-Klassen sin für alle Räume identisch.

Komplexität der Bedienelemente

Wie bereits erwähnt, sind die Layouts für die Geräte-Klassen für alle Räume gleich, sodass möglichst wenig unterschiedliche Layouts erzeugt werden müssen und somit die Komplexität der Layouts gering gehalten werden kann. Dadurch müssen Änderungen am Layout nur einmal pro Geräteklasse durchgeführt werden.

Um die Funktionalität der Buttons an die korrekte Kombination aus Room und Control anzupassen, kommt die ButtonFactory zum Einsatz.

Bedienelemente

In den meisten Fällen werden Buttons verwendet, um Lichter ein- oder auszuschalten, Fenster zu öffnen und zu schließen oder weitere Aktionen durchzuführen.

Um das Licht zu dimmen, kommt ein Schieberegler zum Einsatz.

Die Farbe des Lichts lässt sich wählen, indem der *ColorPicker* über einen Button aufgerufen wird. Dieser öffnet sich in einem eigenen Dialog Fenster. Indem man mit dem Finger über den Ring fährt wählt man die entsprechende Farbe aus, wobei bei schwarz das Licht ausschaltet. Die gewählte Farbe wird in der Mitte des Rings angezeigt. Berührt man diese, wird die Farbe aktiviert und das Licht wechselt die entsprechende Farbe.

Realisierung

Entwicklungsumgebung

Autor: NF

- Intellij IDEA 12
    - Zuvor wurde Eclipse verwendet, wir sind jedoch aus folgenden Gründen auf IDEA umgestiegen:

    - Besserer GUI-Editor in Intellij IDEA
    - Probleme beim Kooperativen Arbeiten mit Eclipse
- Android SDK
- Samsung Kies
- Visual Paradigm

Laufzeitumgebung

Autor: NF

- Android 4.0.3 (minimum)
- Nicht Abwärtskompatibel
- Unterstützung für Tablets (getestet ab 10‘‘)

Zusätzliche Bibliotheken

Autor: NF

- JDK 1.6
- Android SDK 4.0.3
- AndroidPublisher

Screendumps (?)

Einfache Screenshots? Von allen möglichen Layouts, nur Besonderheiten hervorheben oder verwendete Elemente jeweils einmal zeigen?

Installationsanleitung

Was genau soll hier geschrieben werden? Installation ohne IDE? Ansonsten wäre das einfach nur aus IDE heraus starten.

Erweiterbarkeit (beide, jeder für seine Teile…)

Button Factory

Autor: NF

Kapselt das Erzeugen von Listener der Buttons für die einzelnen Views. Dazu wird zuerst geprüft um welches Layout es sich handelt. Anschließend wird für das entsprechende Layout für jeden Button ein Listener mit einem bestimmten Kontext erzeugt.

Fall ein neues Layout erstellt wird muss nur die Button Factory und die Enumeration Control angepasst werden.

Falls sich die Funktionalität eines Layouts ändert muss nur die Button Factory angepasst werden.

MessageAdapter

Autor: TM

Der *MessageAdapter* ist sehr entscheidend, wenn man die App erweitern möchte, da dies die zentrale Komponente ist, über die Nachrichten erstellt und versendet werden.

Möchte man neue Geräte-Klassen ansteuern (zum Beispiel ein Fernsehgerät, falls dies zukünftig möglich seien sollte) so muss zunächst ein neuer *Message*-Typ angelegt und die Utility-Klasse *Messages* um entsprechende Methoden zur Nachrichtenerzeugung erweitert werden.

Solange man die App lediglich um das Steuern von zusätzlichen Geräten bereits vorhandener Geräte-Klasse (z.B. Licht, Rollos, …) erweitern möchte, ist eine Anpassung des *MessageAdapters* nicht dringend erforderlich. *Messages* stellt Methoden zur Verfügung, um Nachrichten mit allen erforderlichen Details zu erstellen, um jedes Gerät einzeln anzusprechen. Es kann unter Umständen jedoch sinnvoll sein, neue Hilfs-Methoden zu erstellen, die das Erzeugen von häufig verwendeten Nachrichten vereinfachen.

Enum-Klassen Room und Control

Autor: TM

Die beiden Enum-Klassen *Room* und *Control* müssen auf jeden Fall erweitert werden, sobald neue Räume oder Geräte-Klassen zum Raum hinzugefügt werden, da diese beiden Enums entscheiden, Aktionen den Bedienelementen zugewiesen werden. Dazu muss im Quelltext der Klassen lediglich ein neuer Raum, bzw. eine Geräte-Klasse eingefügt weren.

Schlussbetrachtung und Ausblick (beide)

Ubisense-Intergration nicht erreicht aus Zeitgründen

Herausragend: Gute Erweiterbarkeit, schöne Architektur, …

Ubisense integrieren,