Wahlpflichtmodul, Wintersemester 2012/2013, HAW Hamburg

Projektplan

Smart Home Control

Nils Feyerabend, Tobias Meurer
03.02.2013

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
TM, NF	Finalisieren des Dokuments	03.02.2013	1.0

Team

Autor: TM, NF

Das Team besteht aus folgenden zwei Personen:

- Nils Feyerabend (nils.feyerabend@haw-hamburg.de, Kürzel: NF)
- Tobias Meurer (tobias.meurer@haw-hamburg.de, Kürzel TM)

Im Nachfolgenden werden alle Abschnitte mit dem oben genannten Kürzel der Autoren gekennzeichnet.

Motivation

Autor: NF

Unsere größte Motivation dieses Modul zu wählen war, neben dem Living Place, das Android-Framework.

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 Living Place entwickeln durften, eines der spannendsten Labore der HAW Hamburg.

Verantwortungsbereiche

Autor: TM, NF

- Projektplan (NF, TM)
- GUI-Layout (NF)
- MessageAdapter (TM)
- ButtonListenerFactory (NF)
- Context und Controller Enums (TM)
- ColorPicker (TM, NF)
- Speichern von favorisierten Licht-Farben
    - “Drag & Drop”-Funktionalität (TM)
    - Lesen und Schreiben von Farben in Preference-Dateien (NF)

Zielsetzung und Aufgabenstellung

Projektbeschreibung

Autor: NF

Im Laufe des Wahlpflicht Moduls „Smart Home Control“ soll eine Fernbedienungs-App für den Living Place erstellt werden, welche auf einem Android Tablet läuft.

Die Fernbedienung soll es ermöglichen das Licht, die Fenster, die Gardinen und die Rollos der Wohnung steuern zu 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 Tabs mit den in diesem Raum verfügbaren Geräte-Klassen (Licht, Rollos, ...) angezeigt.

Darüber hinaus sind zwei optionale Funktionalitäten für die App vorgesehen.

Zum einen soll die aktuelle Position des Benutzers ermittelt werden, sodass sich die Oberfläche der Android-App dem Raum anpasst, in dem sich der Benutzer aktuell befindet.

Als zweite Option ist es geplant, dem Benutzer die Möglichkeit zu bieten seine Lieblings-Farben für das Licht zu speichern. Diese Farben sollen auch nach einem Neustart der App zur Verfügung stehen.

Bei der Entwicklung der Fernbedienung soll vor allem auf eine gute Erweiterbarkeit geachtet werden.

Funktionale Anforderungen

Autor: NF

- Bedienbarkeit folgender Element-Arten:
  - Licht
  - Rollos
  - Gardienen
  - Fenster
  - Heizung
- Beim Drehen des Tabletts Änderung von Portrait in Landscape und umgekehrt möglich
  - Ausgewählte Taps sollen erhalten bleiben
- Gliederung der steuerbaren Elemente nach Raum und Art
- Optional: Speichern von vier favorisierten Licht-Farben
  - Speichern der Farben auch beim Beenden der App
- Optional: Automatische Anpassung der Oberfläche auf Basis der Position des Benutzers in der Wohnung
  - Anpassung der Oberfläche mit einer Verzögerung von mehreren Sekunden nach letzter Eingabe, um „Sprünge“ und einen Wechsel während eines Bedienvorgangs zu vermeiden.

Technische, Nicht-Funktionale Anforderungen

Autor: TM, NF

- Lauffähig auf Tablets ab Android 4.0.3
- 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-Kontext Bezeichnungen aus Strings.xml auslesen

KW 55 – 18.01.2013

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

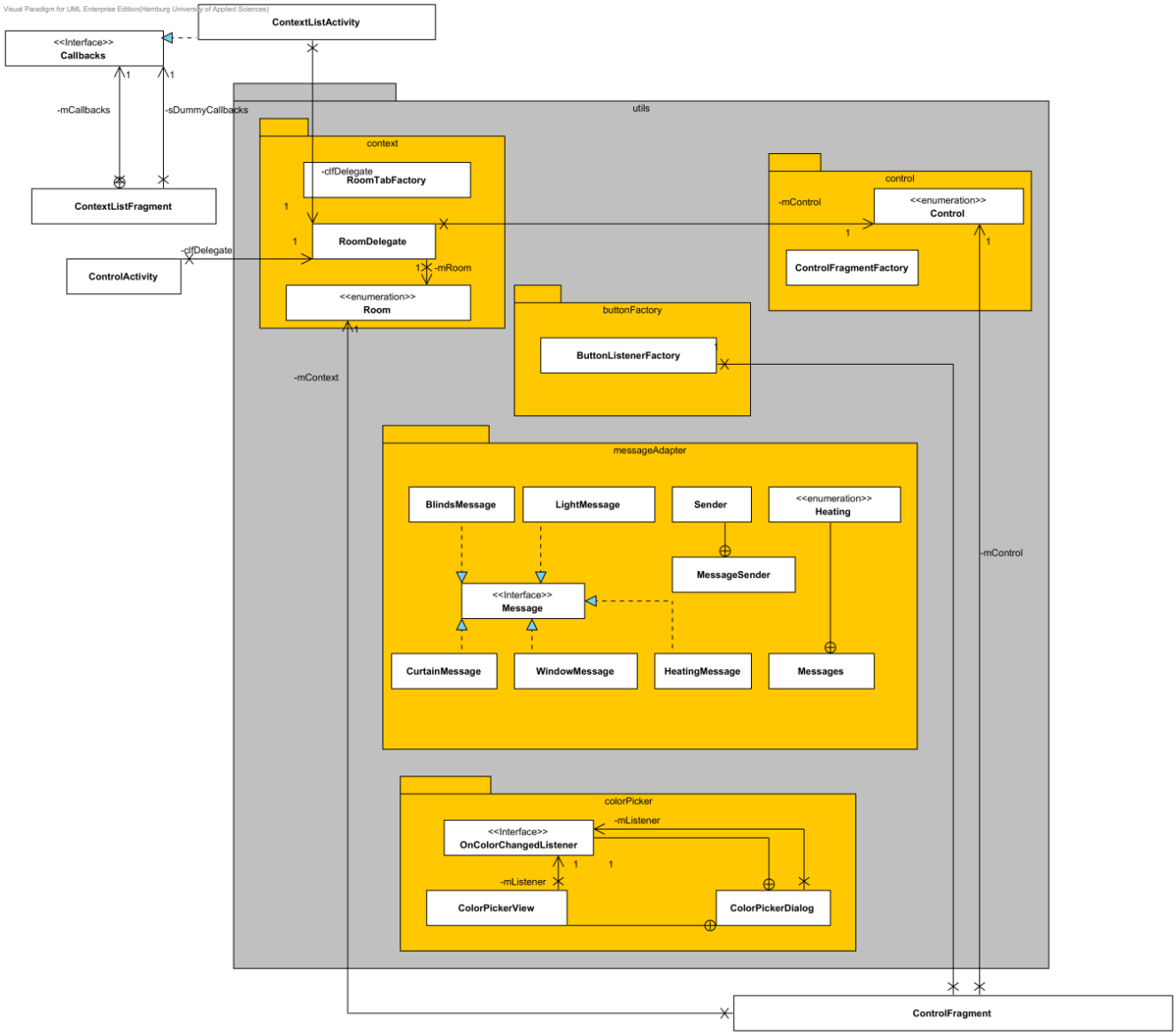
KW 55+

- Fehlerkorrektur
- Finalisieren der App
- Optionale Implementierungen

Entwurf

Klassenmodell

Autor: NF



Klassendiagram: komplett

Systemarchitektur

Autor: TM, NF

Um die Steuerung der Gräte im Living Place zu ermöglichen müssen Nachrichten von der Fernbedienung an die entsprechenden Geräte gesendet werden. Der *AndroidPublischer* wird lokal auf der Fernbedienung ausgeführt. Dieser leitet Nachrichten über das WLAN des Living Place an den Proxy des Message-Queue-Systems weiter. Der Proxy nimmt die Nachrichten in Empfang und übergibt diese an die ActiveMQ. Dort werden die Nachrichten unter einen Nachrichtenspezifischen Topic veröffentlich und von den Subscriber abgeholt. Die Subscriber können in diesem Fall die Licht-, Fenster-, Rollos- oder Gardienen-Steuerungssysteme sein.

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.

Gemeinsamkeiten

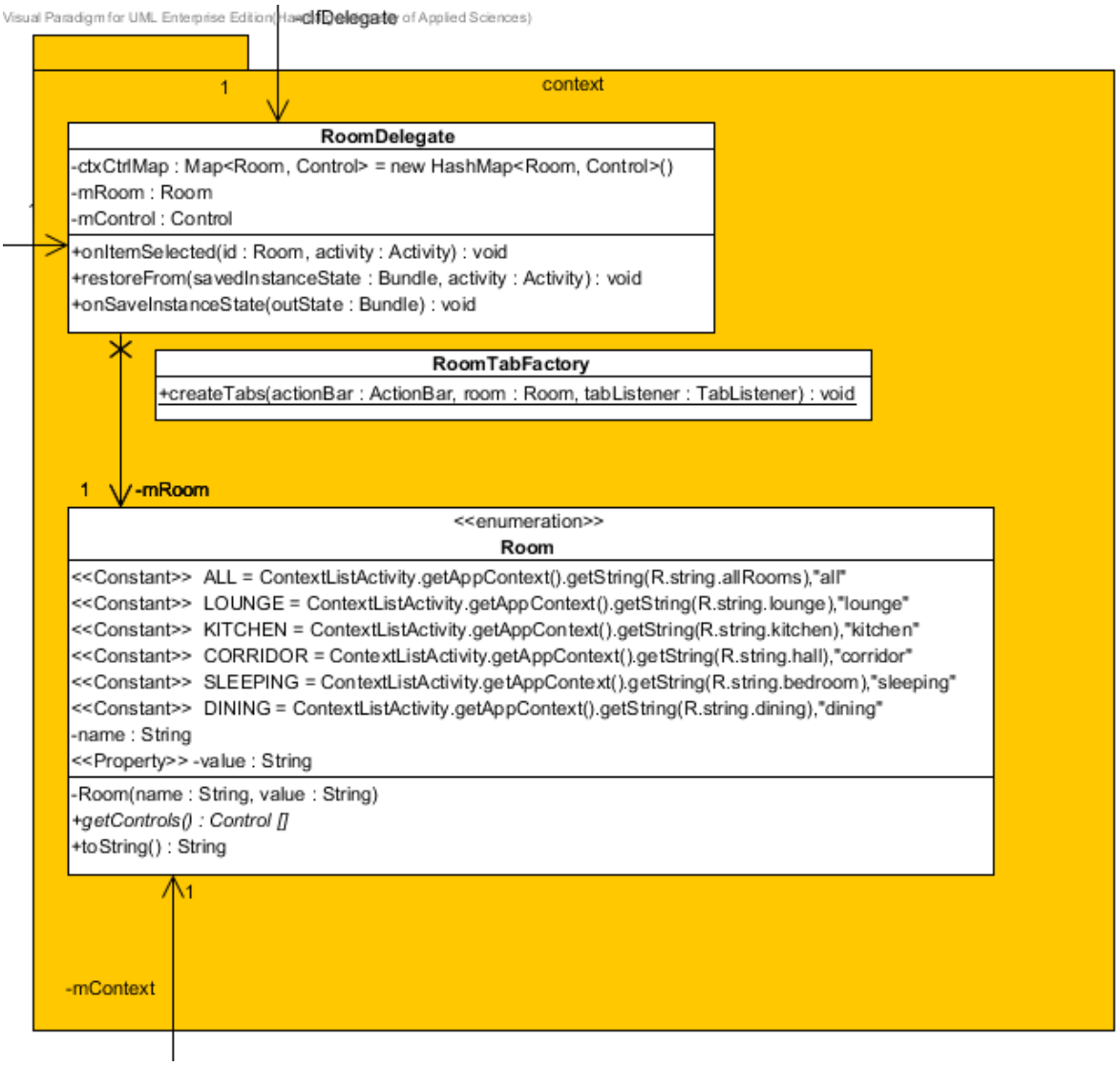
Autor: NF

Es gibt viele Gemeinsamkeiten in der Steuerung der Räume des Living Places. Der Dialog zum manipulieren des Lichtes ist zum Beispiel für die meisten Räume identisch. Um Codeduplizierung zu vermeiden und die Fernbedienung leicht erweiterbar zu machen, haben wir uns dafür entschieden diese Gemeinsamkeiten zusammenzufassen. Um diese Ziel umzusetzen wurden die *ButtonListenerFactory* und der *MessageAdpater* entworfen.

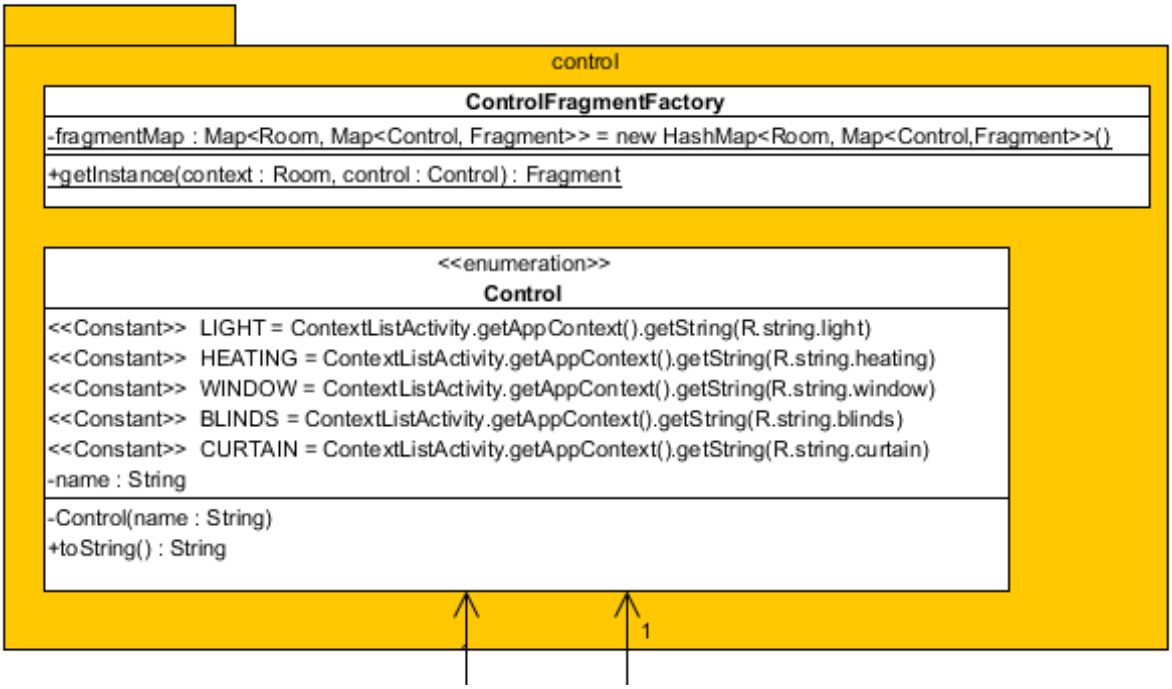
Enum-Klassen Room und Control

Autor: TM

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



Klassendiagram: Auswahl context



Klassendiagram: Auswahl control

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 Living Place (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.

ControlFragment

Autor: NF

Im *ControlFragment* findet die Zuordnung zwischen Layout und Tabs statt. Dazu wird beim Erstellen der View geprüft, welcher Tab im Moment aktiv ist. Dementsprechend wird das passende Layout geladen.

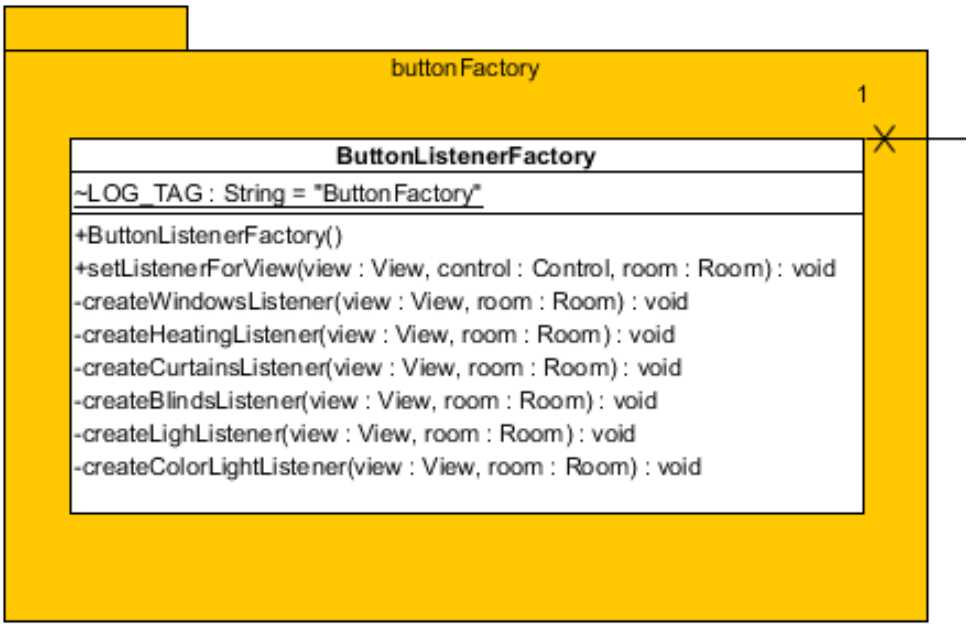
Da das Layout nur das Design bereit stellt und alle Schaltflächen noch keine Funktionalität haben wird anschließend die *ButtonListenerFactory* angerufen.

Außerdem ist das *ControlFragment* für das Speichern und Laden der Lieblingsfarben verantwortlich. Dies geschieht sobald eine View mit Farbauswahl geschlossen bzw. geöffnet wird. Damit die Einstellung der favorisierten Farben nicht nach dem beenden verloren gehen werden diese in den SharedPreferences gespeichert.

ButtonListenerFactory

Autor: NF

Die *ButtonListenerFactory* 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.



Klassendiagram: Auswahl buttonFactory

Dazu werden folgende Informationen vom *ControlFragment* benötigt.

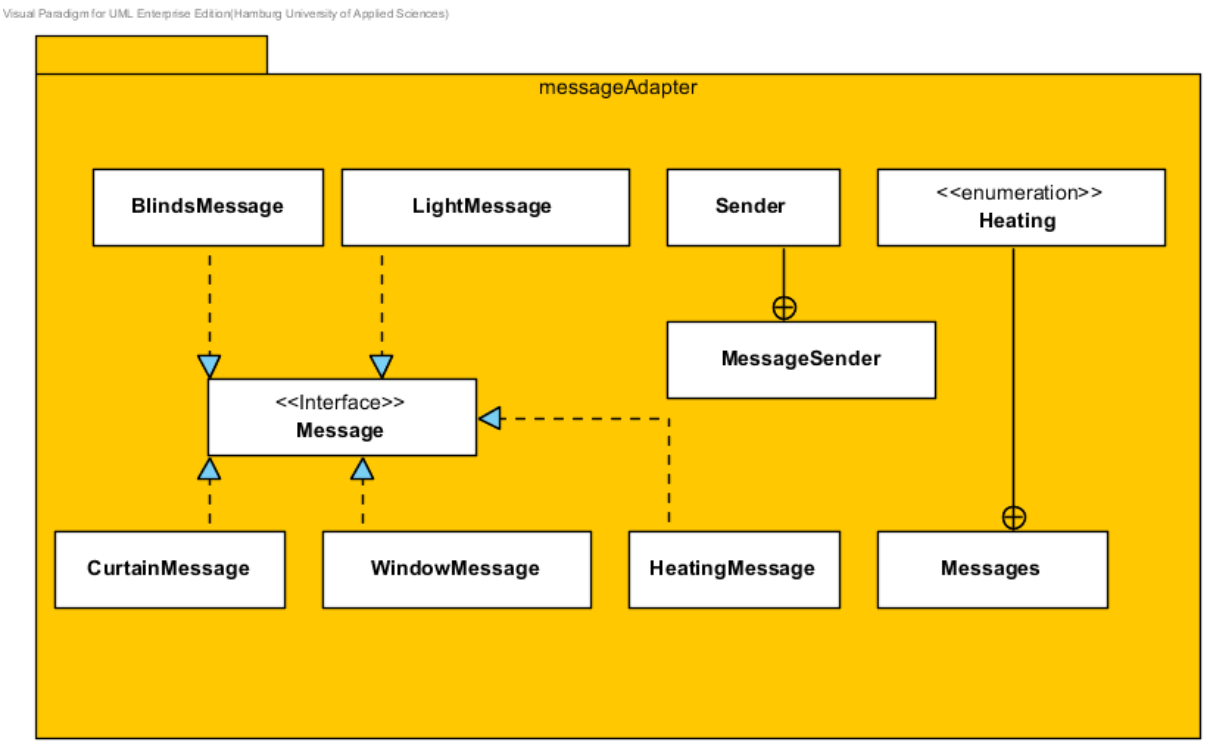
- *Control* Aufzählung der verschiedenen Geräteklassen
- *View* Basisklasse von anzeigbaren GUI-Elemente. In diesem Fall ist View mit dem aktuellen Layout(z.B. Lichtsteuerung) gleichzusetzen.
- *Room* Aufzählung der verschiedenen Räume

Im ersten Schritt wird über die Enumeration *Control* identifiziert für welches Layout Listener erstellt werden sollen. Dies ist wichtig da jedes Layout verschieden Funktionen hat die nur schwer zusammengefasst werden können.
Anschließend werden die zu diesem Layout gehörenden Buttons gesucht und mit einem Listener versehen.
Welcher Button für welchen Raum zuständig ist wird über die Enumeration *Room* entschieden. Im Listener wird abhängig vom entsprechenden Raum eine Message über d en *MessageAdpater* generiert und versendet.

MessageAdapter

Autor: TM

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



Klassendiagram: Ausschnitt MessageAdapter

Der *MessageAdapter* beinhaltet das Interface *Message*, welches von Nachrichten-Klassen für die steuerbaren Geräte-Klassen des Living Place 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 *Sender*, welche dem Versenden von Nachrichten in einem eigenen Thread dient.

Sender greift zum Senden der Nachrichten auf den *AndroidPublisher* zu, welcher den Verbindungsaufbau und das eigentliche Übertragen der Nachricht an den Proxy des ActiveMQ-System des Living Place 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.

GUI-Konzept

Autor: TM

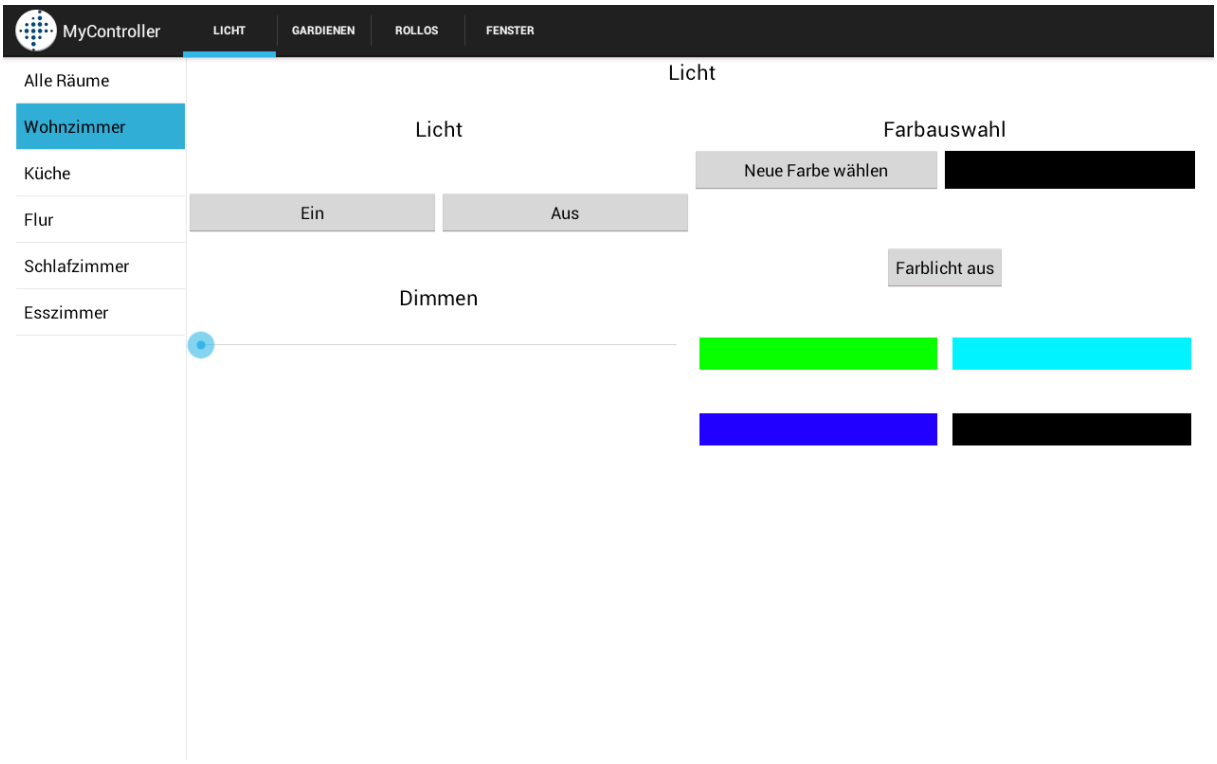
Navigation

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

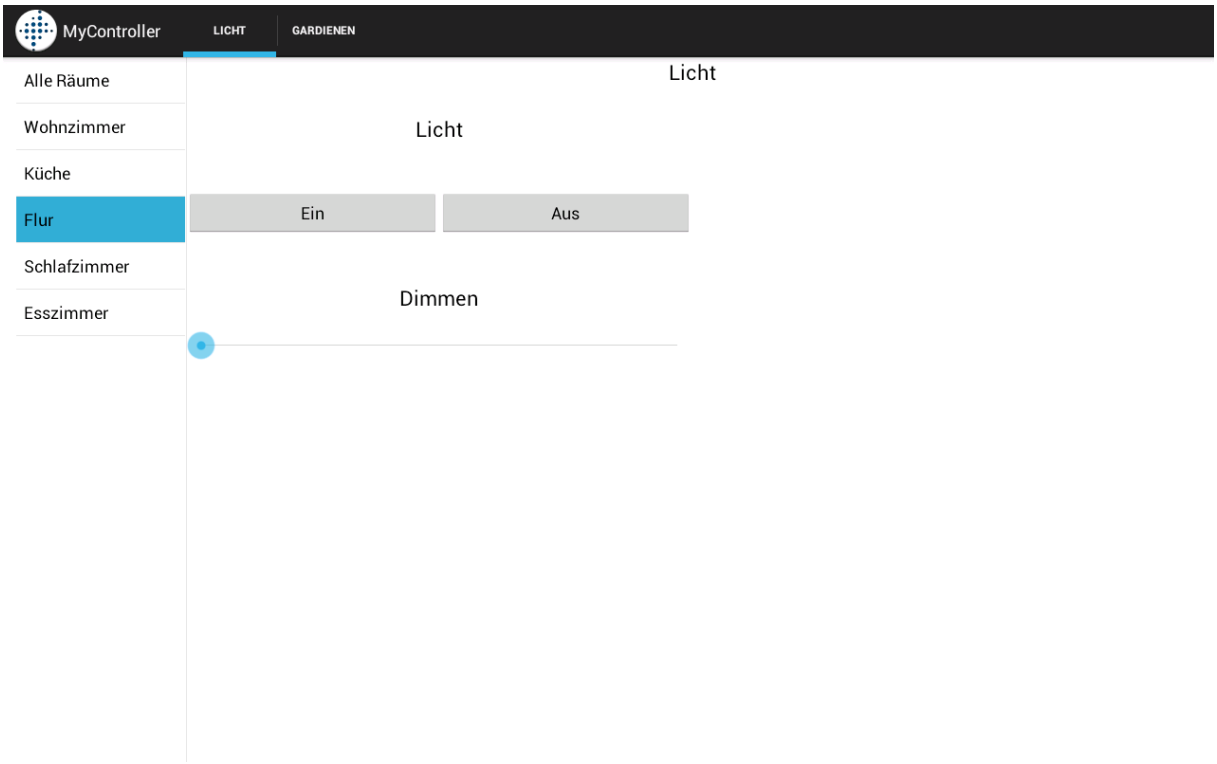
Layout und Designaufwände

Bei den verwendeten Layouts handelt es sich zum Größten Teil um LinearLayouts, 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. Eine Ausnahme bildet das Layout für die Lichtsteuerung im Flur. Dieses ist ein eigenes Layout ohne Farbauswahl, da es im Flur kein farbiges Licht gibt. Nachfolgend sind Screenshots der Lichtsteuerung im Wohnzimmer und des Flurs im Vergleich zu sehen:



Screenshot: Wohnzimmer Lichtsteuerung



Screenshot: Flur Lichtsteuerung

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 *ButtonListenerFactory* 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.

Die gewählte Farbe wird in einem Feld neben dem Button für den die Farbauswahl angezeigt. Von dort aus lässt sie sich per Drag & Drop auf vier verschiedene Favoriten-Felder ziehen. Die Favoriten-Felder sind ebenso, wie alle Buttons, anklickbar.

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 10.1‘‘)

Zusätzliche Bibliotheken

Autor: NF

- JDK 1.6
- Android SDK 4.0.3
- AndroidPublisher

Installationsanleitung

Autor: NF

Bei der Installation sind keine Besonderheiten zu beachten. Beim Starten der App au der IDE heraus werden aller erforderlichen Dateien automatisch auf dem Tablet installiert. Manuelle Tätigkeiten sind bei der Installation nicht erforderlich.

Erweiterbarkeit

Allgemein

Autor: NF

Durch die Verwendung des Master Detail Flows ist eine einfache Anpassung der App für Android-Geräte anderer Größen möglich, z.B. Smartphones.

Da die Bezeichnung der GUI-Elemente nicht fest im Quellcode verankert ist, sondern aus der Datei Strings.xml ausgelesen wird ist einfach eine multilinguale Unterstützung zu implementieren.

Falls neue Geräteklassen(z.B. eine Kaffeemaschine) eingeführt werden soll, muss nur ein neues Layout angelegt, die Aufzählung *Control* muss um den Gerätetypen erweitert werden und es müssen die, weiter unten beschriebenen, Änderungen in der *ButtonListenerFactory*, sowie im *MessageAdpater* vorgenommen werden.

Button Listener Factory

Autor: NF

Die Factory kapselt das Erzeugen von Listener der Buttons für die einzelnen Views. Deswegen ist sie ein entscheidender Punkt, wenn es um Änderungen oder Hinzufügen von Funktionalität geht.

Falls ein neues Layout erstellt wird, muss in der Factory eine neue Methode zum Erzeugen der Listener für die Steuerelemente des Layouts erstellt werden.

Falls sich die Funktionalität eines Layouts ändert muss in der Methode für dieses Layout die Logik der Listener für die entsprechenden Steuerelemente 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. Die Klasse *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 zur Fernbedienung hinzugefügt werden, da diese beiden Enums entscheiden, welche Aktionen den Bedienelementen zugewiesen werden. Dazu muss im Quelltext der Klassen lediglich ein neuer Raum, bzw. eine Geräte-Klasse eingefügt werden.

Schlussbetrachtung und Ausblick

Autor: TM, NF

Rückblick

Aus unserer Sicht haben wir eine gute Applikation entwickelt. Wir haben bei der Entwicklung auf eine mögliche Erweiterung der Fernbedienung, eine Einfache Software-Architektur und die Kapselung der einzelnen Komponenten geachtet.

Das GUI wurde schlicht und übersichtlich gehalten, um dem Benutzer eine intuitive Bedienoberfläche zu bieten.

Alle geplanten obligatorischen Anforderungen wurden im Laufe des Semesters in dieser Android-App umgesetzt. Darüber hinaus ist es uns gelungen, eines unserer optionalen Ziele zu erreichen.

Dabei handelt es sich um die Möglichkeit, favorisierte Farben für die Lichtsteuerung so zu speichern, dass diese auch nach einem Neustart der App zur Verfügung stehen. Das Speichern funktioniert automatisiert. In diesem Zusammenhang haben wir auch eine „Drag & Drop“-Funktionalität implementiert, die dem Benutzer eine besonders einfache Bedienung ermöglicht.

Leider konnte das zweite optionale Ziel, die Ubisense-Integration zur Positionserkennung des Benutzers, aus zeitlichen Gründen nicht umgesetzt werden. Des Weiteren wurde die Steuerung der Heizungen in der App deaktiviert. Grundsätzlich steht diese Funktionalität zwar zur Verfügung, jedoch war aus der vorhandenen Dokumentation das Nachrichten-Format für die Heizungssteuerung nicht ersichtlich, sodass die Implementierung nicht vollständig durchgeführt werden konnte.

Ausblick

Wenn wir die Möglichkeit hätten, unsere App noch weiter zu entwickeln, so würden wir zunächst die Ubisense-Integration umsetzten, da dies ein komfortables Features zur Manipulation der Oberfläche auf Basis des aktuellen Standortes des Benutzers ist. Darüber finden die Ubisense-Technologie zur Positionsbestimmung interessant und würden uns gerne näher damit befassen.

Außerdem fänden wir es gut ein individuelles GUI-Design zu erstellen, sodass dieses sich vom Standard-Design für Apps abhebt und individuell konfigurierbar (zum Beispiel Farbschema) ist.