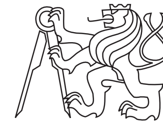
Insert here your thesis' task.

Czech Technical University in Prague
Faculty of Information Technology
Department of Software Engineering



Master's thesis

Intelligent home control system

Bc. Jiří Adamec

Supervisor: Ing. Vratislav Zima

4th May 2015

Acknowledgements

I would like to thank my supervisor who gave me great advices and helped me a lot with my thesis. I would also like to thank my fiancee for her support and my family for their support and funding my education.

Declaration

I hereby declare that the presented thesis is my own work and that I have cited all sources of information in accordance with the Guideline for adhering to ethical principles when elaborating an academic final thesis.

I acknowledge that my thesis is subject to the rights and obligations stipulated by the Act No. 121/2000 Coll., the Copyright Act, as amended, in particular that the Czech Technical University in Prague has the right to conclude a license agreement on the utilization of this thesis as school work under the provisions of Article 60(1) of the Act.

In Prague on 4th May 2015 ....

Czech Technical University in Prague
Faculty of Information Technology
© 2015 Jiří Adamec. All rights reserved.
This thesis is school work as defined by Copyright Act of the Czech Republic. It has been submitted at Czech Technical University in Prague, Faculty of Information Technology. The thesis is protected by the Copyright Act and its usage without author's permission is prohibited (with exceptions defined by the Copyright Act).

Citation of this thesis

Adamec, Jiří. *Intelligent home control system.* Master's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2015.

Abstrakt

Tato práce se zabývá tvorbou prototypu aplikace pro platformu Android a Android Wear. Tato aplikace je určena k ovládání domovního systému, takzvaného chytrého domu. Aplikace je spustitelná na mobilním telefonu a také na chytrých hodinkách. V této diplomové práci je popsáno jaké technologie jsou použity, dále jak probíhala analýza a návrh aplikace, realizace a také testování.

Klíčová slova Android, Wear, chytrý dům, chytré hodinky

Abstract

This work deals with the creation of prototype application for the Android and Android Wear platforms. The application is designed to control a house system, the so-called smart house. The application is running on the mobile phone and also on the smart watches. In this master's thesis is described which technologies are used, how the analysis and application design were done and also implementation and testing.

Keywords Android, Wear, smart house, smart watch

Contents

Introduction **1**
Definition of the task 2
Requirements 2
Review of similar services 3

1 State-of-the-art **5**
1.1 Android 5
1.2 Android Wear 16
1.3 Android Studio 19
1.4 WebSocket 20
1.5 JSON 22

2 Analysis and design **25**
2.1 Analysis ot the server API 25
2.2 Design of data classes 28
2.3 Design of user interface 30

3 Realisation **35**
3.1 Used libraries 35
3.2 Implementation of the mobile application 39
3.3 Implementation of the wearable application 42
3.4 Communication between wearable and mobile applications . . . 43

4 Testing **47**
4.1 Testing on Android 47
4.2 Usability testing 47

Conclusion **53**

Bibliography **55**

A Acronyms **57**

B Contents of enclosed CD **59**

List of Figures

1.1 Android versions 6
1.2 Android architecture 10
1.3 Activity lifecycle 12
1.4 Fragment lifecycle 14
1.5 Service lifecycle 15
1.6 Android Studio project 21

2.1 ELAN RF 25
2.2 Data classes 29
2.3 Screen relation 31
2.4 Loading screen 32
2.5 Settings screen 32
2.6 Room list screen 33
2.7 Room screen 33
2.8 Bulb off 34
2.9 Bulb on 34
2.10 Thermostat 34
2.11 Thermostat opened 34

3.1 Color picker 38
3.2 State change from wearable 44
3.3 State change from WebSocket 45

4.1 The Android testing framework. 48
4.2 Number of problems found depending on number of users. 49

List of Tables

1.1 Gartner research 5
1.2 Versions share 6

Introduction

Nowadays we are experiencing large expansion of smart electronics to all parts of our everyday life. From our smartphones to smart fridges that send us an SMS about running low on food. A lot of people is interested in controling their house devices like bulbs, sockets and thermostats from computer or smartphone. These systems that are in control of the whole house and provide control from computer or smartphone are called smart house systems.

This thesis is about designing and implementing a prototype client application for smart house system made by ELKO EP s.r.o. company. The application is developed for Android and Android Wear platforms, in order to provide functionality to smart phones and smart watches.

The thesis contains six chapters. The first chapter is this Introduction, that describes the task of this thesis, all requirements and also contains review of similar smart house systems.

In the chapter named State-of-the-art are described current technologies that are used to implement the application. The first section 1.1 describes the Android platform, its history, architecture and its core components. Second section 1.2 contains description of the Android Wear platform with its API and functionality that it offers. In section 1.3, named Android Studio, is information about the new IDE designed for developing applications for Android platform. The section 1.4 describes the WebSocket protocol, that is used to receive notifications from server. Last section 1.5 is about JSON format.

Second chapter Analysis and design describes how the analaysis and design of the application were done. In first section 2.1 is described how the server part is implemented and what API it provides. Next section 2.2 contains definition of data classes designed according to the server API and server resources. Last section 2.3 describes design of the user interface, the design is created by the Android standards.

The third chapter Realisation contains all information about implementation of the application. Description of all used libraries is in first section 3.1. Second section 3.2 describes the implementation of the mobile application

and the next section 3.3 is about implementation of the wearable application. Last section 3.4 contains description of communication between the mobile and wearable applications.

In chapter Testing is described all testing of the application that was done. First section 4.1 offers information about possibilities of testing on Android platform and second section 4.2 describes usability testing. The usability testing is performed by usability heuristic and also user testing.

Last chapter Conclusion contains a summary of the whole thesis. In Conclusion, there is also described how the clients were satisfied with the application.

Definition of the task

The objective of this master thesis is to desing and implement a prototype of android client application for existing smart house system made by ELKO EP s.r.o. company. The prototype will include application for Android Wear platform in order to provide possibility of controling the smart house by using smart watches.

Requirements

Functional requirements

1. Allow users to control following devices on phone:
    - simple light
    - dimmable light
    - RGB light
    - blinds
    - thermostat
2. Allow users to control these devices on smart watches.
3. Allow users to open current room from phone on smart watch.

Non-functional requirements

1. Application will be compatible with Android 4.0.3 and newer.
2. Application will include Czech and English localization.

Review of similar services

Loxone

Loxone[1] is a company that develops smart home system based on their Loxone Miniserver. The Miniserver is designed for a single family home, but it keeps plenty of capacity in reserve. Up to 30 extensions can be connected to a single Miniserver which translates to an extra 492 inputs and 372 outputs. The Miniserver has a powerful 400 MHz processor and 64MB of memory (RAM). The operating system uses approximately 10 MB of memory, so the remaining free memory is available to the program and all the communication tasks.

Loxone provides control of all important appliances in home, like lights, blinds, heating, alarm etc.. The system takes care about energy consumption by automatic temperaturte setting or blinds control. This helps keep your house heated in winter and cool in summer. With the help of a central function, Loxone can stop the electricity being drained away by devices left on standby.

One of the great functions that Loxone has is mobile app developed to control the system. Their app is made for Android and iOS platforms. The mobile application contains light control, automatic blinds control, intelligent temperature control, alarm system etc.. It is also possible to control the whole system over web interface or wall-mounted tablet.

NEST Thermostat

The NEST Thermostat is designed to lower heating and cooling bills up 20%. The termostat learns your day schedule, programs itself and can be controled from mobile phone. The Nest Thermostat learns what temperatures user like and builds a personalized schedule[2]. It learns efficient temperatures for a few days and, within a week, it will start setting them on its own.

With Auto-Away function, the Nest Thermostat automatically turns to an energy-efficient Away temperature when the user is gone. Auto-Away works in 90% of homes, even if your Nest is in a spot user do not pass on your way out the door.

The mobile and web application is used to control the thermostat remotely. Users can change actual temperature and their schedule or put NEST into Away mode. The application also provides detailed Energy History, so users can see how the termostat really saves their money.

NuBryte

The NuBryte is a smart home console designed to automate your lighting and security capabilities, and provide you with updated energy and weather reports, plus an array of other household management features[3]. The NuBryte

system controls the lights in each room where a console is installed. However, some features, such as the intercom, require at least two units in order to function properly. Unfortunately, the NuBryte console does not provide ability to control lights through any mobile app at this moment.

Chapter 1

State-of-the-art

1.1 Android

1.1.1 Introduction to Android

Android is the most widely used operating system for smartphones. It gained absolute majority on a smartphone market, as you can see in table 1.1 from IDC research[4] published in February 2015. Android holds this majority for about three years and is very likely to continue in his domination.

1.1.2 History of Android

Android, Inc. was founded by Andy Rubin, Rich Miner, Nick Sears and Chris White in Palo Alto, California in October 2003. At first, the wanted to create system for smart cameras. However after the Apple inc. released their first iPhone, Andy Rubin saw big opportunity in smartphone market and started to think about targeting the Android to smartphones. They gave themselves a task to create *smarter mobile devices that are more aware of its owner's location and preferences*[5], but after a year of creating a new mobile operating system, they ran out of money. Fortunately, Steve Perlman, a close friend of Rubin, brought him $10,000 in an envelope and refused a stake in the company. After nearley two years of existence, on August 17, 2005, Android

Period	Android	iOS	Windows Phone	BlackBerry OS	Others
Q4 2014	76.6 %	19.7 %	2.8 %	0.4 %	0.5 %
Q4 2013	78.2%	17.5%	3.0%	0.6%	0.8%
Q4 2012	70.4%	20.9%	2.6%	3.2%	2.9%
Q4 2011	52.8%	23.0%	1.5%	8.1%	14.6%

Table 1.1: Smartphone operating systems market shares

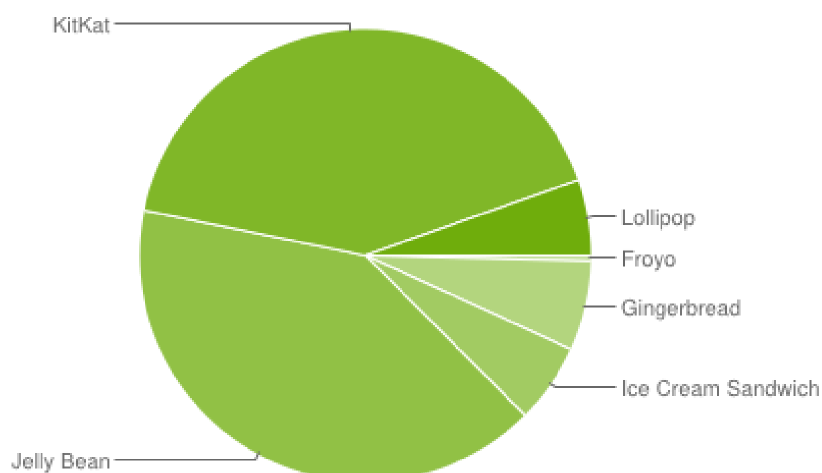


Figure 1.1: Share of devices running a given version of the Android from April 6, 2015[7].

Version	Codename	Distribution
2.2	Froyo	0.4%
2.3.3 - 2.3.7	Gingerbread	6.4%
4.0.3 - 4.0.4	Ice Cream Sandwich	5.7%
4.1.x	Jelly Bean	16.5%
4.2.x	Jelly Bean	18.6%
4.3	Jelly Bean	5.6%
4.4	KitKat	41.4%
5.0	Lollipop	5.0%
5.1	Lollipop	0.4%

Table 1.2: Share of Android platform versions

with all employees was bought by Google, Inc. which was planning to enter the mobile phone market.

1.1.3 Versions of Android

Since the first public Android release, 22 more updates has been released. The newest version of Android, 5.1 Lollipop was released in February 2015. The following list of versions contains main features which they included[6]. The actual share of Android platform versions is show inf figure 1.1 and table 1.2.

- **Beta version** - The first Android beta version was released on 5 November 2007 and after one week, on 12 November 2007 was released SDK[**?**].

- **Android 1.0 (API Level 1)** - On 23 September 2008 Google released first commercial version of Android, after one month HTC announced a first smartphone running Android, it was the HTC Dream also known as T-Mobile G1. HTC Dream was equiped with a hardware QWERTY keyboard, because the first version of Android did not contain any software keyboard. This version included several Google services such as Android market, the official store with applications, Gmail, Google calendar, Google Maps, YouTube.

- **Android 1.1 (API level 2)** - Released on 9 February 2009, initialy for HTC Dream only, resolved a couple of issues and improved system stability. This version also brought new features e.g. support for saving attachments from MMS.

- **Android 1.5 Cupcake (API level 3)** - On 30 April 2009, the Android 1.5 update was released, it was the first release to use a codename based on a dessert item. The new version brought the ability of creating application widgets on home screen, the media framework was improved providing raw audio recording and playback, video recording API and media search. It was also added a couple of localizations including czech language.

- **Android 1.6 Donut (API level 4)** - Released on 15 September 2009. In Donut were introduced new feature such as quick search box providing search across multiple sources e.g. browser history, contacts, apps, directly on home screen, VPN support, battery usage indicator or better and faster interface for camera and camcorder.

- **Android 2.0 – 2.1.x Eclair (API level 5-7)** - The Android 2.0 was released on 26 October 2009 bringing ability to add multiple accounts for email and contacts synchronization. The camera application was improved and included support for built-in flash, digital zoom, scene modes, white balance and color effects. Browser now supported HTML5 features such as application cache, database api, geolocation api and `<video>` tag in fullscreen mode.

- **Android 2.2.x Froyo (API level 8)** - Released on 20 May 2010 bringing better Exchange support, Exchange administrators can remotely reset the device to factory defaults, Exchange calendars can be synced in the Calendar application. The application Camera was upgraded, videos can be shot with LED flash on. This version also brought a great new feature, which is the possibility to turn the device into a portable Wi-Fi hotspot.

- **Android 2.3.x Gingerbread (API level 9-10)** - On 6 December 2010, the Android 2.3 was released with new functions such as one-touch

word selection and copy/paste, improved power management, NFC support and also support for multiple cameras. The Android soft keyboard was redesigned and optimized for faster text input and editing.

- **Android 3.x Honeycomb (API level 11-13)** - The first tablet-only Android version was released on 22 February 2011. The UI was optimized for tablets. The system bar was placed on the bottom of screen including software button. This version introduced Action Bar which provides a better navigation in applications. For the first time, developers could use Fragments in their applications. Fragments provide better user experience. The Android keyboard was redesigned to make writting faster. Multiple tabs replaced browser windows and a new "incognito" mode allows anonymous browsing. The first device running Honeycomb was Motorola Xoom.

- **Android 4.0.x Ice Cream Sandwich (API level 14-15)** - The Android 4.0.1 was released on 19 October 2011 was designed for unification of Android for phones and tablets running Honeycomb. The UI was redesigned for ideal performance on phones and tablets a new typeface brought better readability on larger screens. Widgets were now interactive, users can flip trough their calendars, check emails and more. Multiple system apps were improved for example voice search, spell checker, data usage controls and camera which received panorama mode. The new Android supports Wi-Fi Direct technology which provides peer to peer connection between devices.

- **Android 4.1.x – 4.2.x Jelly Bean (API level 16-17)** - First Jelly Bean was released on 9 July 2012 with the primary aim of improving the functionality and performance of the user interface. The performance improvement involved "Project Butter", which uses touch anticipation, triple buffering, extended vsync timing and a fixed frame rate of 60 fps to create a fluid and "buttery-smooth" UI. New version brought a great feature for users sharing one device with others. These users can now switch between multiple Google accounts on one device and use a separate environment, including their own homescreens, widgets, accounts, settings, files, and apps, and the system keeps these separate.

- **Android 4.3 Jelly Bean (API level 18)** - The last Jelly Bean version was released on 24 July 2013. Google aimed on better battery performance. This version brought several small fixes and improvements. For example right-to-left languages support. With this version, Google brought Google Now function, which informs users about importatnt information that he may use. Such as weather, filght times, sport results, many of the information Google gets from users Gmail.

- **Android 4.4 KitKat (API level 19-20)** - KitKat version is the first version named after commercial product. Google made deal with Nestle company about rights to use this name. This version brought memory optimisations in project called "Project Svelte". The required mimimum of memory is 340MB. The biggest inovation is ART (Android Runtime) which replaces Dalvik. ART uses ahead of time compilation, which results in better application permormance and speed. Unfortunately ART was not enabled by default, but could be only enabled in developer options. The API level 20 brought support for Wearable devices.

- **Android 5.0 - 5.1 Lollipop (API level 21 - 22)** - The Lollipop was released on November 12, 2014. In this version, ART definitely replaces Dalvik. Google introduced Material design, responsive design pattern used in all Google applications. Lollipop brought a lot of improvements, unforutnately also few bugs, the biggest bug was a memory leak in system applications causing the device to almost run out of memory. The new Android improved notifications, they are now shown on lock screen and important notifications show up as pop up on the top of the screen. Google Play Services now contains Smart Lock feature which holds device unlocked if user sets safe locations or safe device to which is phone paired.

1.1.4 Android Architecture

The architecture is divided into several layers, as you can see in figure 1.2, the bottom layer is a Linux kernel with device drivers. First Android systems were using Linux kernel version 2.6.x and since Ice Cream Sandwich the 3.x kernel version has been used. Above the kernel lays a HAL[1]. HAL provides standard interface to access device drivers without need to know low-level implementation.

Application framework prodvides a rich collection of components simplifying applications development such as set of Views that can be used to build an application. The views are including lists, buttons, text boxes, layouts, etc.. Content Providers are enabling applications to access data from other applications. The access to non-code resources such as localized strings, graphics, and layout files is providing Resource Manager. Developers have access to multiple System Services like Location Service, Bluetooth Service, Notifications Service etc.. Usage of these Services is permited by requiring permissions in application manifest.

Every Android application runs in its own process, with its own instance of the virtual machine. Until the version KitKat, Android used Dalvik as virtual machine. Dalvik has been written so that a device can run multiple VMs

[1]Hardware Abstraction Layer - interface to call into device driver layer

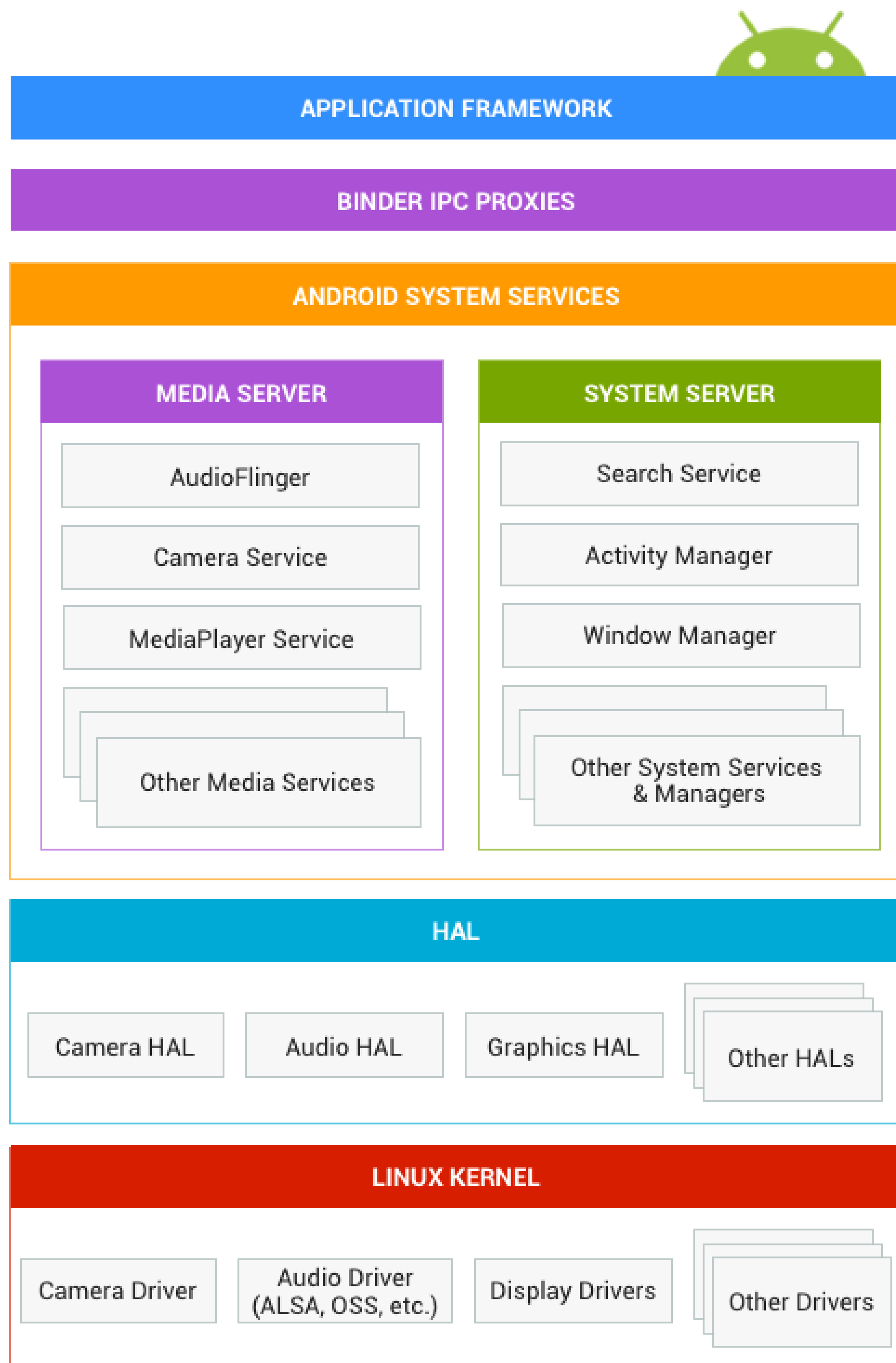


Figure 1.2: Android architecture

efficiently. The Dalvik VM is register-based, and runs classes compiled by a Java language compiler that have been transformed into the .dex format. Since the Android 2.2 Dalvik uses JIT[2]. From version KitKat, Dalvik was replaced by ART (Android Runtime). ART intorduced AOT[3] compilation which uses little bit more storage space on device, but nowadays devices storage space has grown so it is not a big problem. ART brings faster execution of applications, improved memory allocation and garbage collection mechanisms.

1.1.5 Core components of Android

1.1.5.1 Activity

An Activity[8] is an application component that provides a screen with which users can interact in order to do something. Each activity is given a window in which to draw its user interface. The window typically fills the screen, but may be smaller than the screen and float on top of other windows. Every Activity has to be declared in application manifest in order to be accessible to the system:

```
<manifest ... >
    <application ... >
        <activity android:name=".MainActivity" />
        ...
    </application ... >
    ...
</manifest >
```

Typically, one activity in an application is specified as the "main" activity, which is presented to the user when launching the application for the first time. Every activity can start another activity in order to perform different actions and show different interface. Each time a new activity starts, the previous activity is stopped, but the system preserves the activity in a stack (the "back stack"). When a new activity starts, it is pushed onto the back stack and takes user focus.

When an activity is stopped because a new activity starts, it is notified of this change in state through the activity's lifecycle callback methods. There are several callback methods that an activity might receive, due to a change in its state—whether the system is creating it, stopping it, resuming it, or destroying it—and each callback provides you the opportunity to perform specific work that's appropriate to that state change. For instance, when stopped, your activity should release any large objects, such as network or database connections. When the activity resumes, you can reacquire the necessary resources

[2] Just In Time compilation - compilation to bytecode only when it is necessary
[3] Ahead of Time compilation - compilation to bytecode during application instalation

and resume actions that were interrupted. These state transitions are all part of the activity lifecycle shown in figure 1.3.

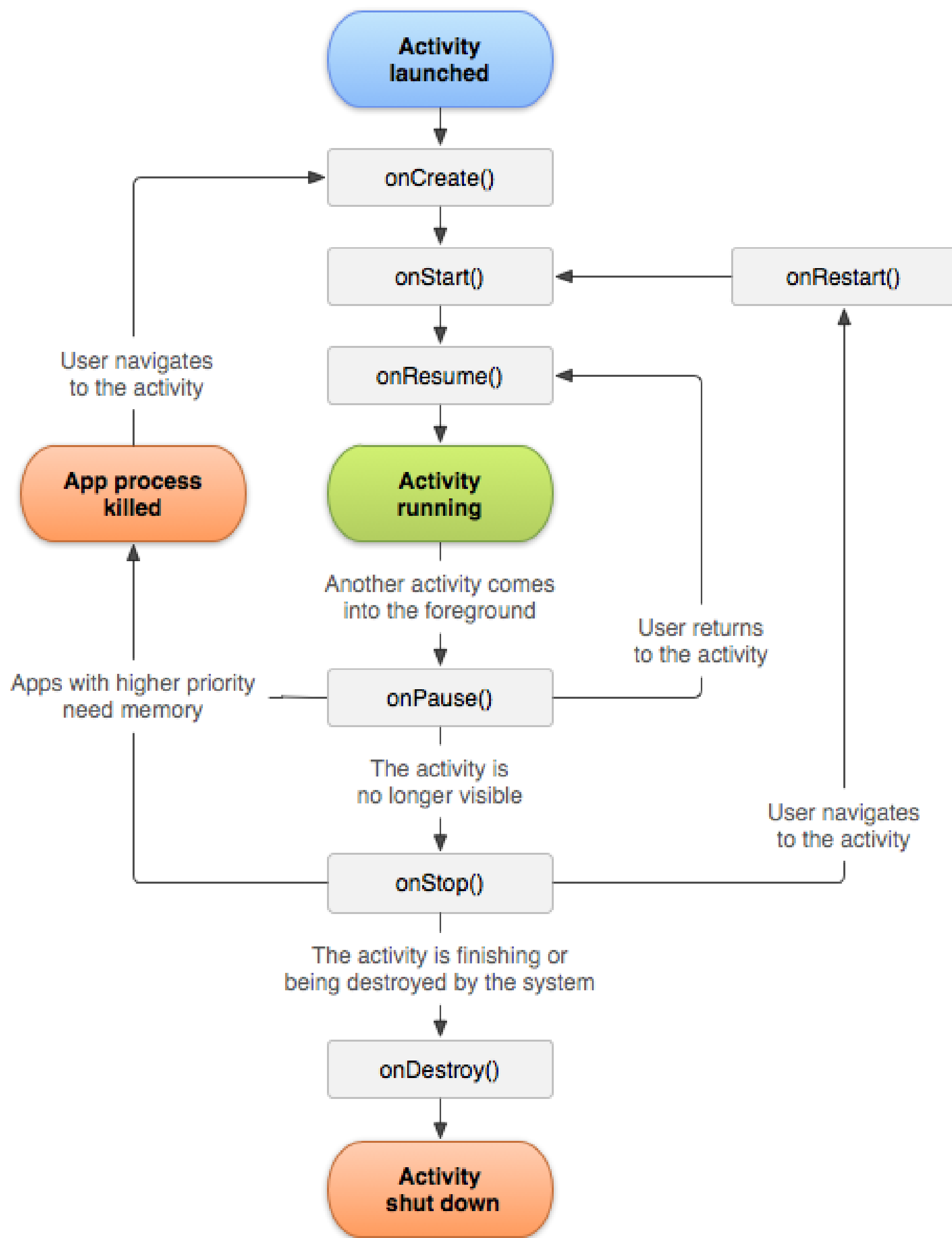


Figure 1.3: Activity lifecycle

1.1.5.2 Fragment

A Fragment[9] represents a behavior or a portion of user interface in an Activity. You can combine multiple fragments in a single activity to build a multi-pane UI and reuse a fragment in multiple activities. A fragment must always be embedded in an activity and the fragment's lifecycle (shown in figure 1.4) is directly affected by the host activity's lifecycle.

When you add a fragment as a part of your activity layout, it lives in a ViewGroup inside the activity's view hierarchy and the fragment defines its own view layout. You can insert a fragment into your activity layout by declaring the fragment in the activity's layout file, as a `<fragment>` element, or from your application code by adding it to an existing ViewGroup. However, a fragment is not required to be a part of the activity layout; you may also use a fragment without its own UI as an invisible worker for the activity.

Android introduced fragments in Android 3.0, primarily to support more flexible UI designs on large screens, such as tablets. Because a tablet's screen is much larger than that of a handset, there is more room to combine and interchange UI components. Fragments allow such designs without the need for you to manage complex changes to the view hierarchy. By dividing the layout of an activity into fragments, you become able to modify the activity's appearance at runtime and preserve those changes in a back stack that is managed by the activity.

1.1.5.3 Service

A Service[10] is an application component that can perform long-running operations in the background and does not provide a user interface. Another application component can start a service and it will continue to run in the background even if the user switches to another application. There are essentially two types of services:

- **Started** - A service is "started" when an application component (such as an activity) starts it by calling `startService()`. Once started, a service can run in the background indefinitely, even if the component that started it is destroyed. Usually, a started service performs a single operation and does not return a result to the caller. For example, it might download or upload a file over the network. When the operation is done, the service should stop itself.

- **Bound** - A service is "bound" when an application component binds to it by calling `bindService()`. A bound service offers a client-server interface that allows components to interact with the service, send requests, get results, and even do so across processes with interprocess communication (IPC). A bound service runs only as long as another application

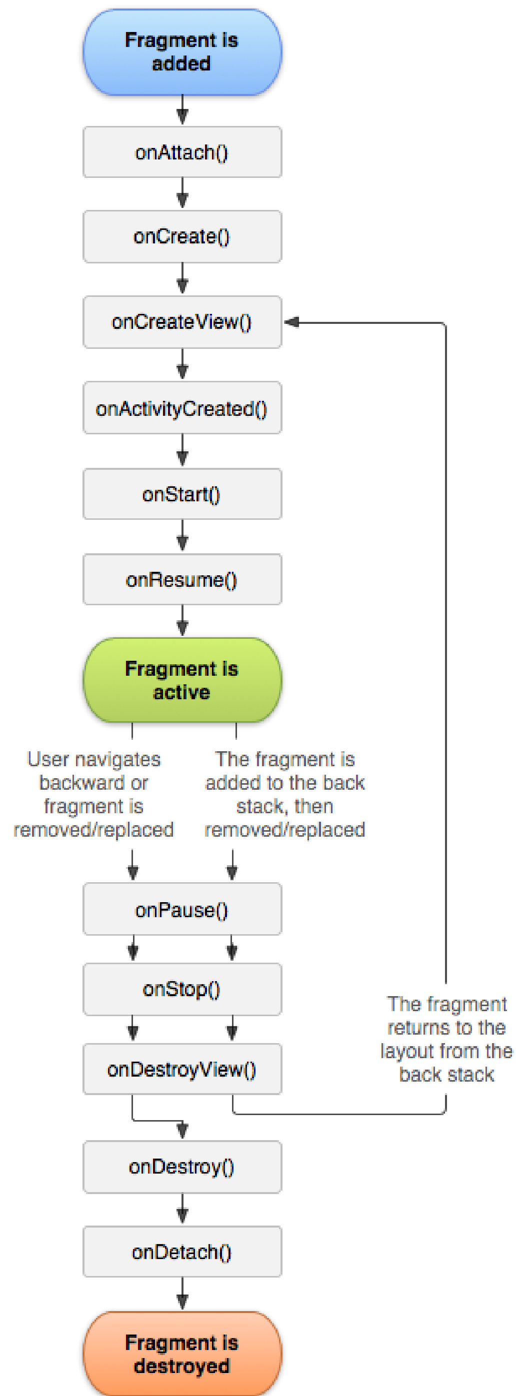


Figure 1.4: Fragment lifecycle

component is bound to it. Multiple components can bind to the service at once, but when all of them unbind, the service is destroyed.

Like activities, all services must be declared in the application's manifest file:

```
<manifest ... >
    <application ... >
        <service android:name=".DataService" />
        ...
    </application ... >
    ...
</manifest >
```

The lifecycle of a service is much simpler than that of an activity. However, it is even more important that developer should pay close attention to how the service is created and destroyed, because a service can run in the background without the user being aware. Figure 1.5 describes the lifecycle of a service.

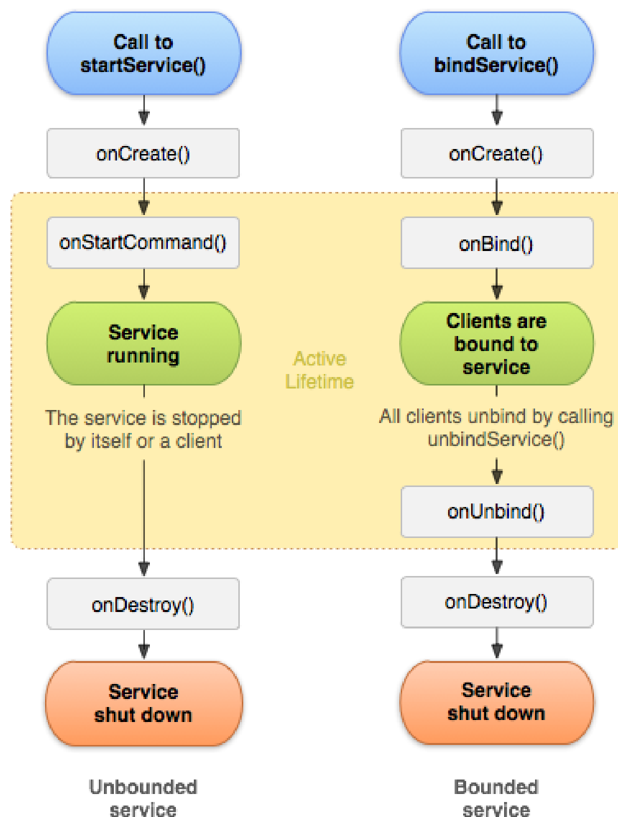


Figure 1.5: Service lifecycle

1.2 Android Wear

Android Wear is a subversion of Google's Android platform designed for smartwatches and other wearables. Wear devices can be paired with devices running Android 4.3 and newer. Android Wear integrates Google Now technology and mobile notifications into a smartwatch form. Users can perform voice searches using "OK Google" voice command. New applications can be installed to watch using phone and Google Play Store.

In many ways, Android Wear seems to be based on the experience from Google Glass project. The operating system looks similar to the interface used by Glass. It is probable that Google has transformed their work from controversial Glasses to something more usable.

The platform was presented on March 18, 2014 along with release of developer preview and developer kit. At the same time, manufacturers such as Motorola, LG, HTC, etc. were announced as partners[11]. At Google I/O on 25 June 2014 the Samsung Gear Live and LG G Watch were launched.

The first version was based on Android 4.4 KitKat and contained API to handle round and square watch screaens. On December 10, 2014 Google released new version based on Android 5.0 Lollipop introducing new Watch Face API, which provides the ability to change watch home screen with custom watch face.

1.2.1 Android Wear API

Android Wear provides API subset of Android platform but do not support the following list of API:

- `android.webkit`
- `android.print`
- `android.app.backup`
- `android.appwidget`
- `android.hardware.usb`

Wearable Apps should be small in size and functionality compared to handheld apps. The should contain only what makes sense on the wearable which is usualy small subset of handheld app. In general, you should make all operations on handheld and to the wearable send only result. The handheld should be doing all heavy processing. Also all the network actions can be accessed only through paired handheld. Wearables provide interface to low-level hardware such as heartbeat sensor, pedometer, etc..

The system enforces a timeout period. If your activity is displayed and user do not interact with it, the device goes to sleep. After wake up the

Wear home screen is displayed instead of your activity. When you need to show something persistent, you should create a notification and send it to the wearable.

Users do not install apps directly onto their wearables, but install handheld app which is bundled with wearable app. The system automatically installs this app to paired wearable. However for developing purposes, you can install the Wear app directly to the wearable using developer tools.

1.2.2 Android Wear Layouts

The Android Wear platform provides basic layouts and widgets form standard Android, such as `Button`, `Switch`, `LinearLayou` etc.. The Wear API also provides layouts for handling the difference between square and round watch screen which is necessary to display appropriate GUI. The first one is `WatchViewStub` which inflates specific layout depending on the shape of the device's screen. The second one is `BoxInsetLayout` that is basically a `FrameLayout` that is aware of screen shape and can box its children in the center square of a round screen.

Other Wear specific layouts are:

- `CardFragment` - A fragment that presents content within an expandable, vertically scrollable card.
- `CircledImageView` - An image view surrounded by a circle.
- `ConfirmationActivity` - An activity that displays confirmation animations after the user completes an action.
- `DelayedConfirmationView` - A view that provides a circular countdown timer, typically used to automatically confirm an operation after a short delay has elapsed.
- `DismissOverlayView` - A view for implementing long-press-to-dismiss.
- `DotsPageIndicator` - A page indicator for `GridViewPager` that identifies the current page in relation to all available pages on the current row.
- `GridViewPager` - A layout manager that allows the user to both vertically and horizontally through pages of data. You supply an implementation of a GridPagerAdapter to generate the pages that the view shows.
- `GridPagerAdapter` - An adapter that supplies pages to a GridViewPager.

- `FragmentGridPagerAdapter` - An implementation of GridPagerAdapter that represents each page as a fragment.

- `WearableListView` - An alternative version of `ListView` that is optimized for ease of use on small screen wearable devices. It displays a vertically scrollable list of items, and automatically snaps to the nearest item when the user stops scrolling.

1.2.3 Data synchronization

Communication between handheld and wearable is provided by Wearable Data Layer API, which is part of Google Play Services. Without Google Play Services is wearable unable to pair with phone. The API consists of a set of data objects that the system can send and synchronize between wearable and handheld. These objects are:

- **Data Items** - A DataItem API provides data storage and automatic syncing between the handheld and wearable.

- **Messages** - The Messages API is good for remote procedure calls (RPC), such as controling media player or starting activity from wearable on handheld. Messages are also great for a request/response communication.

- **Asset** - Asset objects are designed for sending binary data, mainly images. The image is attached to data item and system takes care of the transfer, conserving Bluetooth bandwidth by caching large assets to avoid re-transmission.

- **WearableListenerService** - Extending WearableListenerService lets you listen for important data layer events in a service. The system manages the lifecycle of the WearableListenerService, binding to the service when it needs to send data items or messages and unbinding the service when no work is needed.

- **DataListener** - Implementing DataListener in an activity lets you listen for important data layer events when an activity is in the foreground. Using this instead of the WearableListenerService lets you listen for changes only when the user is actively using your app.

Google strictly discourages opening low-level bluetooth communication between handheld and wearable in their developer manual. The correct approach is to use the API listed above.

1.3 Android Studio

Android Studio is the official IDE for Android application development, based on IntelliJ IDEA. On top of the capabilities you expect from IntelliJ, Android Studio offers[12]:

- Flexible Gradle-based build system
- Build variants and multiple apk file generation
- Code templates to help with building common app features
- Rich layout editor with support for drag and drop theme editing
- Lint tools to catch performance, usability, version compatibility, and other problems
- ProGuard and app-signing capabilities

1.3.1 Android Build System

The biggest change in comparison to Eclipse is the Gradle build system. The Gradle system provides rich support for customized build scripts. You can create multiple APKs for you app with different features using the same project. The flexibility of the Android build system enables you to achieve all of this without modifying your app's core source files.

With the Android build system (based on Gradle), the applicationId attribute is used to uniquely identify application packages for publishing. The application ID is set in the android section of the `build.gradle` file in comparison to Eclipse where the application ID is set in application manifest, also the `versionCode` and `versionName` should be set in `build.gradle` file. Here is an example build file (from my project):

```
apply plugin: 'com.android.application'

android {
    compileSdkVersion 21
    buildToolsVersion "21.1.2"

    defaultConfig {
        applicationId "cz.cvut.adameji4.smarthouse"
        minSdkVersion 15
        targetSdkVersion 21
        versionCode 1
        versionName "1.0"
    }
```

```
    buildTypes {
        release {
            minifyEnabled false
            proguardFiles
                getDefaultProguardFile('proguard-android.txt'),
                'proguard-rules.pro'
        }
    }
}
```

1.3.2 Android Studio Project

The project structure appearance is changed in comparison to Eclipse, that was previous official IDE for developing Android applications. Each instance of Android Studio contains a project with one or more application modules. Each application module folder contains the complete source sets for that module, including `src/main` directories, resources, build file and the Android manifest. For the most part, you will need to modify the files under each module's `src/main` directory for source code updates, the `gradle.build` file for build specification. In figure 1.6 you can see my Android project.

1.4 WebSocket

WebSocket is a protocol providing full-duplex communication channel over a TCP connection. The protocol was standardized as RFC 6455[13] in 2011, and the WebSocket API in Web IDL is being standardized by the W3C. The goal of this technology is to provide a mechanism for browser-based applications that need two-way communication with servers that does not rely on opening multiple HTTP connections (e.g., using XMLHttpRequest or `<iframe>` and long polling).

The WebSocket is desinged to be used in web browsers and web servers, but can be used in any client-server application. The protocol is an independent TCP-based protocol. The only relationship to HTTP is that its handshake is recognized by HTTP servers as an Upgrade request. The WebSocket protocol brings better possibility of communication between browser and web server, facilitating live content and real-time gaming. This is psossible by providing a standardized way for the server to send content to browser and allowing for messages to be pushed both ways, from server to client and from client to server.

Standard WebSocket communication is done over TCP port 80, which is benefit for environments which block non-web connections by firewall.

To establish a WebSocket connection, the client initiates a WebSocket handshake request, for which the server returns a WebSocket handshake re-

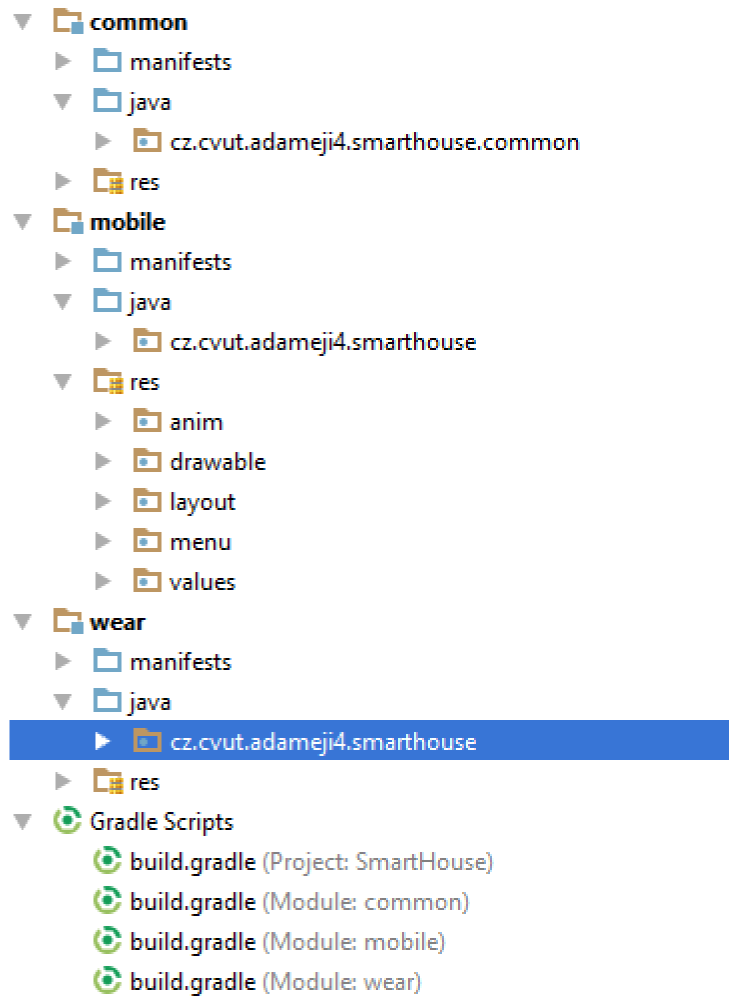


Figure 1.6: Example of Android Studio project structure

sponse, as you can see in following example, the example is taken from the RFC web-page:

Client request:

```
GET /chat HTTP/1.1
Host: server.example.com
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Key: x3JJHMbDL1EzLkh9GBhXDw==
Sec-WebSocket-Protocol: chat, superchat
Sec-WebSocket-Version: 13
Origin: http://example.com
```

Server response:

```
HTTP/1.1 101 Switching Protocols
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Accept: HSmrc0sMlYUkAGmm5OPpG2HaGWk=
Sec-WebSocket-Protocol: chat
```

The client sends a `Sec-WebSocket-Key` which is a random value that has been base64 encoded. To form a response, the GUID `258EAFA5-E914-47DA -95CA-C5AB0DC85B11` is appended to this base64 encoded key. The base64 encoded key will not be decoded first. The resulting string is then hashed with SHA-1, then base64 encoded. Finally, the resulting reply occurs in the header `Sec-WebSocket-Accept`.

When the connection is established, the client and server can send WebSocket data or text frames back and forth in full-duplex mode. The data is only minimally framed, with a small header followed by payload. WebSocket transmissions are described as "messages", where a single message can optionally be split across several data frames. This can allow for sending of messages where initial data is available but the complete length of the message is unknown.

1.5 JSON

JSON is a shorthand for JavaScript Object Notation. It is an open standard format for human-readable text used to transmit data objects consisting of key-value pairs. JSON is primarily used to transmit data between a server and web application. The JSON format was originally designed by Douglas Crockford as an alternative to XML. JSON is currently described in RFC 7159[14] standard. The official internet media type is `application/json`.

1.5.1 Data types and format

JSON's basic types are following:

- Number — a signed decimal number that may contain a fractional part and may use exponential E notation.
- String — a sequence of zero or more Unicode characters. Strings are delimited with double-quotation marks and support a backslash escaping syntax.
- Boolean — either of the values `true` or `false`

- Array — an ordered list of zero or more values, each of which may be of any type. Arrays use square bracket notation with elements being comma-separated.
- Object — an unordered collection of key-value pairs where the keys are strings. Since objects are intended to represent associative arrays, it is recommended, that each key is unique within an object. Objects are delimited with curly brackets and use commas to separate each pair, while within each pair the colon ':' character separates the key from its value.
- `null` — An empty value, using the word `null`

JSON generally ignores any whitespace around or between syntactic elements but not within a string value. JSON does not provide or allow any sort of comment syntax. The following example shows usage of JSON representing JSON object. The example is taken from the RFC page.

```
{
    "Image": {
        "Width":  800,
        "Height": 600,
        "Title":  "View from 15th Floor",
        "Thumbnail": {
            "Url":    "http://www.example.com/image/481989943",
            "Height": 125,
            "Width":  100
        },
    "Animated" : false,
    "IDs": [116, 943, 234, 38793]
    }
}
```

Chapter 2

Analysis and design

2.1 Analysis ot the server API

The hardware and its server API is provided by ELKO EP s.r.o company. Their product is named ELAN RF and acts as local server which directs all requests from clients to a particular devices and controls their states. The ELAN RF is small embedded system with ARM processor running custom version of linux. API for ELAN RF is running on HTTP server, which handles all request in JSON format.

The server API is based on REST architecture. Root resource of the server is `/api` which provides info about current API used on local ELAN RF and links to other resources available to client. In my application prototype, the mainly used resources are `/api/rooms` and `/api/devices`. The API also provides address to WebSocket service running on ELAN RF, that provides

Figure 2.1: ELAN RF device (picture provided by ELKO EP s.r.o company)

notifications about state changes.

2.1.1 WebSocket notifications

When the ELAN RF performs any change in its state, such as configuration change or device state change, the WebSocket service sends to all connected clients a message containing URL of the resource which state has changed. In most cases, the changed resource is device's state. The client should immediately fetch the new state and make appropriate changes in his user interface.

2.1.2 Rooms resource

The rooms resource represents all rooms available in a local smart house controled by ELAN RF. The room resource itself contains information about its type, its label and set of devices placed inside the room. Specific room resource is on path `/api/rooms/<room_id>`. The room JSON representation:


```
{
    "floorplan": {},
    "room info": {
        "type": "bathroom",
        "label": "Bathroom"
    },
    "id": "room00396",
    "devices": {
        "HeatCoolArea": {
            "coordinates": [
                0.49686846137046814,
                0.4689781069755554
            ]
        },
        "RGBBulb": {
            "coordinates": [
                0.5167014598846436,
                0.46715328097343445
            ]
        },
        "Blinds": {
            "coordinates": [
                0.7223381996154785,
                0.4197080135345459
            ]
        }
    }
}
```

2.1.3 Devices resource

In devices resource on path /api/devices are listed all devices available in a local smart house. Specific device info is located on path /api/devices/<device_id>. Every device is described by its type, product type and label. The main information about every device is what primary and secondary action it can perform and information about that action. For example simple bulb has primary action on and secondary actions delayed on and delayed off. Unfortunately the API does not guarantee that the type assigned to a device is correct, so the decision about correct device type is based on what actions can the device perform. The following JSON represents dimmable light:

```
{
    "device info": {..},
    "actions info": {
        "brightness": {
            "type": "int",
            "min": 0,
            "max": 100,
            "step": 10
        }
    },
    "primary actions": ["brightness"],
    "secondary actions": ["brightness"],
    "settings": {},
    "id": "RFDA11B"
}
```

As you can see, the device provides action brigthness which type is integer. Minimal value is 0, maximal 100 and step is 10. The action is executed by sending PUT request to the device resource containing following JSON:

```
{
    "brightness": 20
}
```

If the request was successful, the server returns response with code 204 No Content. If an internal error occurred, the response has code 500.

Every device provides a resource with information about its state on path /api/devices/<device_id>/state. For example a dimmable light has following state:

```
{
```

```
    "brightness": 20,
    "locked": false
}
```

This state means that the bulb is on 20% of its brightness and is not locked so the client can change its state.

2.2 Design of data classes

After analyzing the server part, I started to design data classes according to room and device resources from server. The figure 2.2 describes all data classes and their relations.

2.2.1 Device class

The class Device is an abstract class which contains parameters that are common for all devices in ELAN RF system. The field `deviceId` is a unique ID of a device among local ELAN RF server. Second field `type` is type of the device, unfortunately it is not guaranteed that the type is correct so the application can not rely on it. Field `title` contains name of the device and last field `locked` defines if the device is in locked state. Locked state menas that no one can change its state.

2.2.1.1 Light class

Light class describes a simple light which has only one field `on`. This field maintains state if the light is switched on or off.

2.2.1.2 DimmableLight class

Dimmable light provides ability to change brightness of a light. It holds only one field and that is the mentioned `brightness`.

2.2.1.3 RGBLight class

The RGBLight class is extending a DimmableLight class, so it can change brightness. The RGBLight also provides functionality to change color of the light. It has three fields `red`, `green` and `blue`. These three fields define the color of the light in additive RGB model.

2.2.1.4 Blinds class

Blinds have only one field named `roll_up` that defines if the blinds are in rolled up state. Unfortunately the ELAN RF does not provide better status of blinds. It would be better if blinds had also information about whether

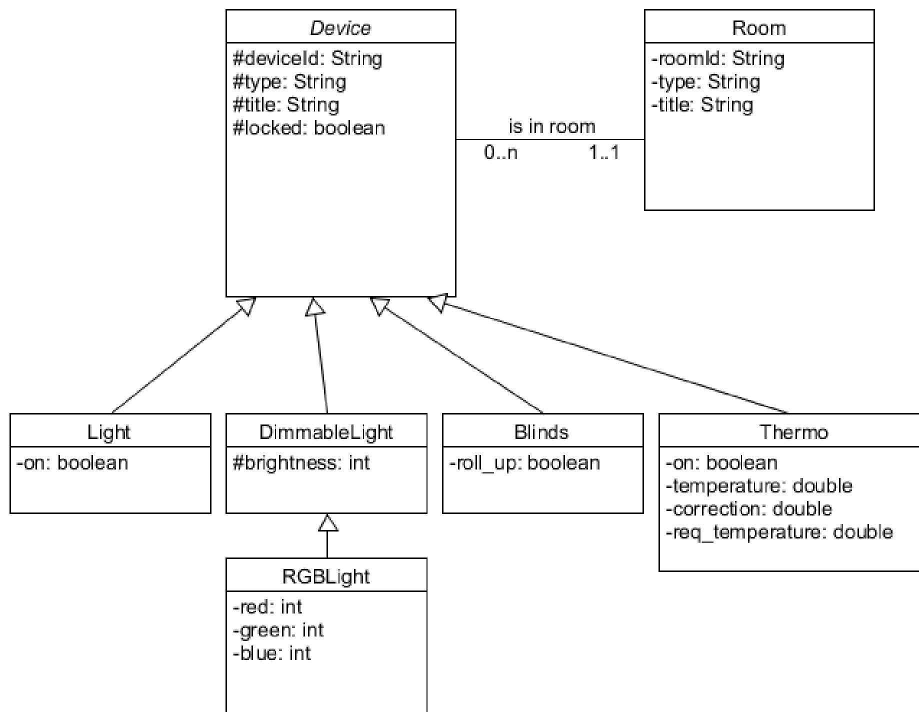


Figure 2.2: Design of the data classes

they are actually in motion or not and whether they are fully rolled up or down or in a middle position.

2.2.1.5 Thermo class

The Thermo class describes a Thermostat. Every thermostat has following fileds: `on`, `temperature`, `req_temperature` and `correction`. The `on` field indicates whether a thermostat is in operation. `Temperature` stores information about actual temperature in a room. Third field `req_temperature` contains what temeperature should be in the room according to temperature plan. Last field `correction` contains a temperature correction against the requested temperature.

2.2.2 Room class

Every room in the smart house is represented by Room class. A room is identified by `roomId` which is unique among the local ELAN RF server. In the filed `type` is stored a room type. Available room type values are: living

room, bedroom, kitchen, dining room, child room, guestroom, hall, study, workroom, bar, garage, garden, bathroom, toilet, laundry, misc, pool, cellar. The last field `title` contains a name of the room that has been assigned by an administrator.

2.3 Design of user interface

After discussion with my supervisor and after reviewing of the similar services, we agreed to design the user interface as simple as it goes and according to the standards of Android. I started to design the application interface by sketching it on the paper and we discussed the sketches with my supervisor. After several discussions we agrred on final design, that I redrawed in a computer in software called Wireframe Sketcher.

2.3.1 Mobile application design

The mobile application is designed to reflect Android Material design, the new design standard that Google introduced with Android Lollipop. Material design introduced replacement for ActionBar, the Toolbar that can be placed anywhere in the UI not only on the top of the screen. The Material design also defines new standard in usage of colors within an application to provide the best user experience. I followed all these standards.

In the mobile app, there are four screens with which the user interacts, the relations between all screens are displayed in figure 2.3.

2.3.1.1 Loading screen

The loading screen is the first screen shown to user. This screen shows a progress during loading of all data from server. When an error occurs, the user is informed with an alert. User has acces to server settings through settings button on Toolbar. The design sketch of loading screen is in figure 2.4.

2.3.1.2 Settings screen

The settings screen provides ability to add and remove server addresses. There are two possibilities of adding new server. The first one is to add manually new IP address. The second one is to run a search service, that searches the local LAN for presence of a local server. In this screen the user should pick which server he wants to control. This screen is shown in fugure 2.5.

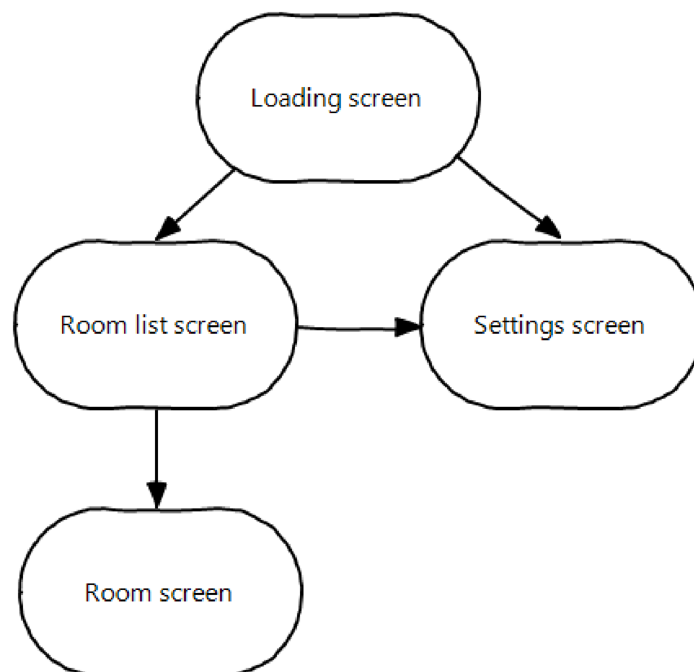


Figure 2.3: The relations between all screens

2.3.1.3 Room list

After all data is loaded, the room list is shown and a user can pick which room he wants to control. The Toolbar provides two buttons, the first button opens the wearable application and the second leads to settings. Figure 2.6 shows the design sketch.

2.3.1.4 Room screen

When a user picks a room he wants to control a Room screen is displayed. The room is represented by list of all devices that a user can control located inside the room. Every device has specific view according to actions that the device can perform.

2.3.2 Wearable application design

The wearable application is just like the mobile part designed according to Material design. Google recommends to all developers to create wearable applications with a minimum of control elements. User of a wearable app should be able to perform all actions within 5 to 10 seconds. The controls should be big enough to be operated during walk, run etc..

With my supervisor and the client (ELKO EP s.r.o company) we discussed the requirements for the wearable application. The result of the discussion

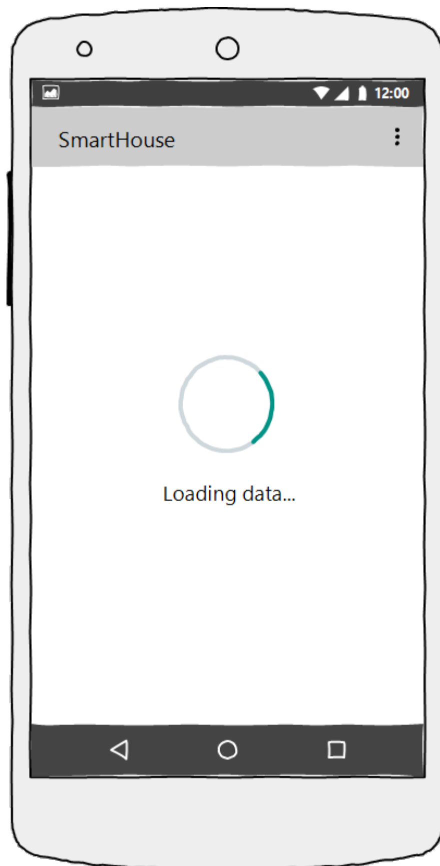


Figure 2.4: Loading screen

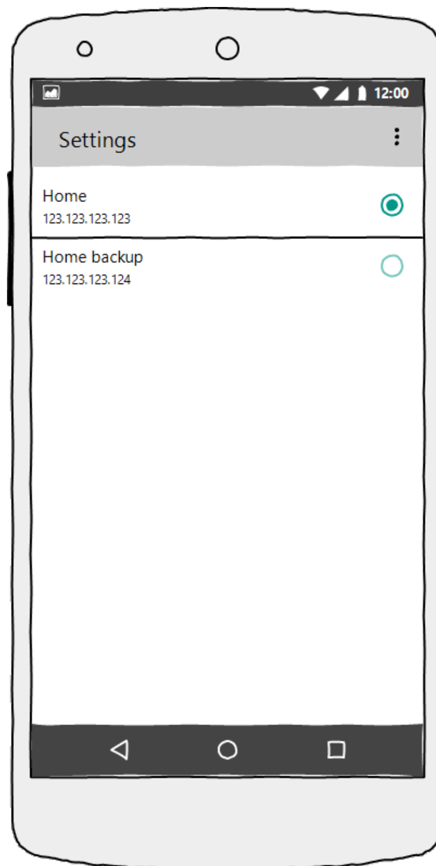


Figure 2.5: Settings screen

was, that the main use case of the wearable app is turning on/off the lights. Therefore the control element for light is a quite large bulb that works as a toggle. The design is shown in figures 2.8 and 2.9. Other devices with more difficult controls (such as thermostat) have expandable card with settings. Figures 2.10 and 2.11 describe closed and opened states.

All devices are in the wearable application displayed as a 2D array. A row represents a room and its devices. By swiping left or right user selects device that he wants to control. Swiping up and down changes room. When a device's layout is in expanded state, the swiping is disabled in order to facilitate control of the elements.

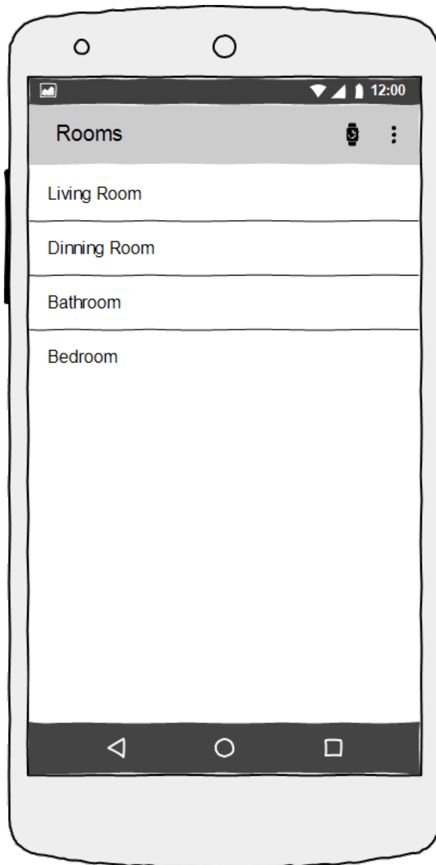


Figure 2.6: Room list screen

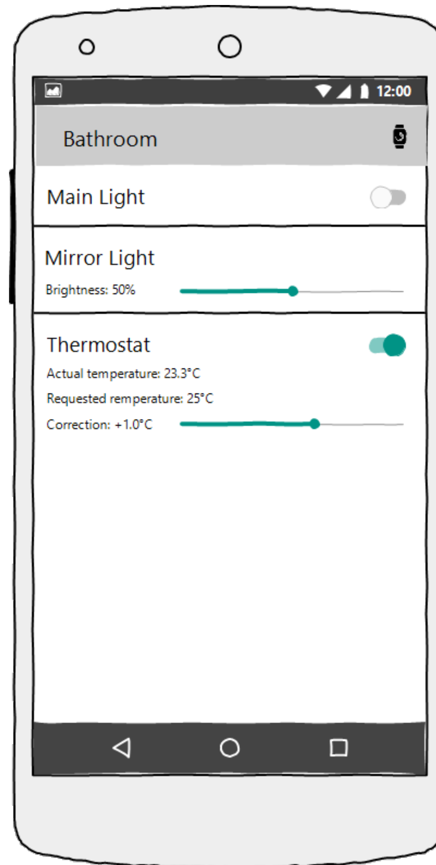


Figure 2.7: Room screen

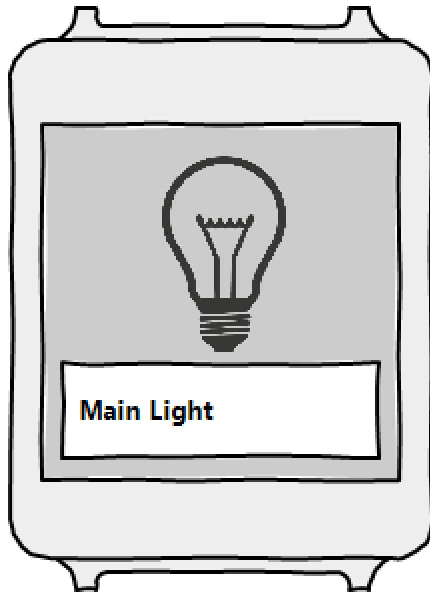


Figure 2.8: OFF state

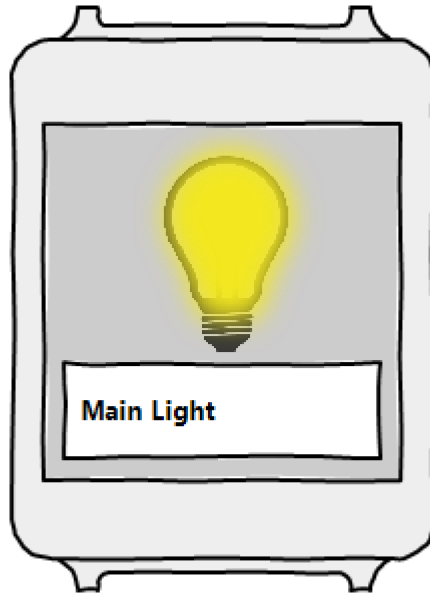


Figure 2.9: ON state

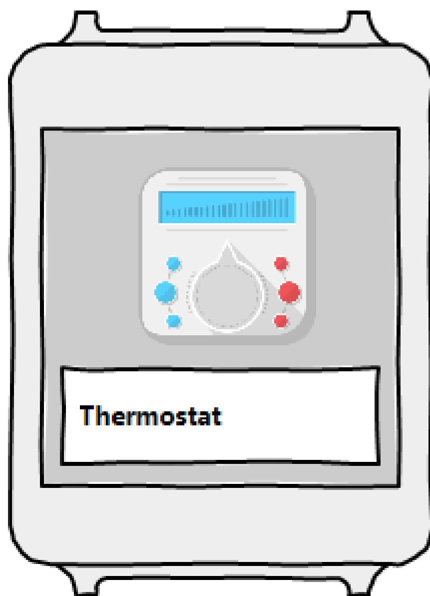


Figure 2.10: Default state

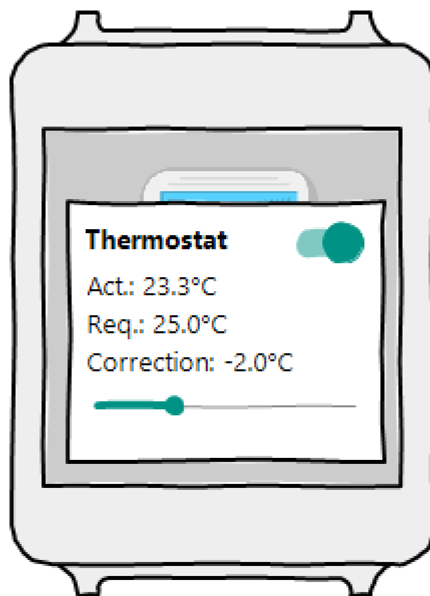


Figure 2.11: Opened state

Chapter 3

Realisation

3.1 Used libraries

Before implementation of the application, my supervisor and I have discussed which libraries will be good to use to avoid writing again things that already exist and are debugged. The libraries and how are they used are listed in following subsections.

3.1.1 Gson

Gson is a Java library developed by Google[15], that can be used to convert Java Objects into their JSON representation. It can also be used to convert a JSON string to an equivalent Java object. Gson provides:

- Simple toJson() and fromJson() methods to convert Java objects to JSON and vice-versa.
- Allow pre-existing unmodifiable objects to be converted to and from JSON.
- Extensive support of Java Generics.
- Custom representations for objects.
- Support arbitrarily complex objects.

At first I was thinking about using Gson library for all serialization and deserialization tasks in the application. However, during the analysis of the server I dicovered several issues that made it impossible.

The first issue was, that if you want to deserialize a JSON string into an Object of abstract class (specifically the Device class) you have to implement `JsonDeserializer` interface defined by Gson library and register it to Gson.

The `JsonDeserializer` will then create correct subclass. It would be possible if the device type was always correct, but the server does not guarantee it.

Second issue was in JSON representation of a device. The fields `primary action` and `secondary action` are sometimes arrays of strings and sometimes arrays of array of string. Here are the examples:

Actions for RGB light

```
"secondary actions": [
    [
        "red",
        "green",
        "blue",
        "brightness"
    ],
    "demo"
]
```

Actions for Thermostat

```
"secondary actions": [
    "correction",
    "mode",
    "on"
]
```

Last issue was that the device's JSON representation sent by server contains field indetifiers with whitespace in it. This prevents direct deserialization into an Object, because no standard Java Object can contain a whitespace character inside a field name. It is not a fatal issue, because the mapping between the JSON field and an Object field can be done also by annotating the Object field with `SerializedName` anotation.

3.1.1.1 Usage of Gson library

Despite all the issues, the Gson library is used to serialize data transfers between the mobile and wearable applications, because the serialization works without issues. However, for the deserialization is used standard `JSONObject` class provided by Android system.

3.1.2 Butterknife

The Butterknife library provides powerful View injection support by annotations. Essentially Butterknife allows you to annotate the inflation of your views and listeners so that you can get back to writing the code that really

matters[16]. Unlike the Roboguice library, Butterknife annotations are processed at compile time so there is no performance impact.

The injection is invoked by static method `Butterknife.inject(this)`. For example in Activity, you should annotate the views to be injected with `@InjectView(R.id.view)` annotation and then in Activity's `onCreate` metod call `Butterknife.inject(this)`. The Butterknife takes care about finding the appropriate view and casting to correct type.

The standard view finding approach

```
public class MyActivity extends Activity {
    private TextView myText;
    private Button myButton;

    public void onCreate(Bundle state) {
        ...
        myText = (TextView) findViewById(R.id.text);
        myButton = (Button) findViewById(R.id.button);
    }
}
```

Approach with Butterknife

```
public class MyActivity extends Activity {
    @InjeectView(R.id.text)
    TextView myText;
    @InjeectView(R.id.button)
    Button myButton;

    public void onCreate(Bundle state) {
        ...
        ButterKnife.inject(this);
    }
}
```

3.1.3 Crashlytics

Crashlytics is a complex crash reporting tool for Android and iOS platforms. It offers detailed report about every crash that happens in your application. Often, the cause of crashes goes beyond the device model or operating system. With Crashlytics, you gain even deeper insights – for example, if an app only crashes in landscape mode, or if the proximity sensor is always on. You can resolve crashes caused by more complex factors, like rooted or jailbroken devices, memory, version-specific issues, etc. – factors you would not otherwise have visibility into. By having this insight, you would be able to ignore an issue

that only occurs on jailbroken devices or focus on how things like orientation and memory can be indicative of what is causing the crash.

The crashlytics reporting system is used in over one milion applications[17]. It is really fast evolving and expanding report system. Crashlytics also provides a tool for beta testing called Beta. It allows you to distribute your application to selected beta testers. If a beta tester encounters an error, the error is immediately delivered to your dashboard with crash reports.

3.1.4 Java-WebSocket library

This library offers an impelemtation of an abstract WebSocket client and server. The client has to implement only four methods to be able to connect to the server and receive messages. These methods are: `onOpen`, `onClose`, `onError` and `onMessage`. The `onMessage(String message)` is the most important, becase it is triggered every time a new message is received.

3.1.5 Android Holo ColorPicker

Holo ColorPicker provides implemented color picker in a Holo design created by Lars Werkman. The library is published under Apache 2.0 license. This library is used to allow users to pick and change color of RGB light bulbs. The usage is shown in figure 3.1.

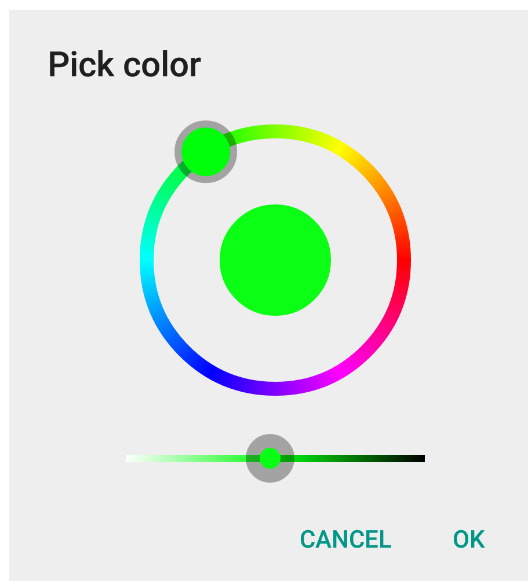


Figure 3.1: Implemetation of the Color picker

3.2 Implementation of the mobile application

The mobile application is based on one main activity and four fragments. The fragments represent screens described in the design subsection 2.3. Important parts of the application are also two services, which handle data processing in background and API client that communicates with the server. All the important parts are described in following subsections.

3.2.1 MainActivity

The `MainActivity` is the only activity in the mobile application. It is the main communication link between the frontend application and background services. The activity takes care about appropriate changing of fragments and displaying menu.

When the activity is started, in the method `onStart` it bounds background service (`DataUpdateService`) and makes a `ServiceConnection` to it. The `ServiceConnection` is used to communicate with the service and invoke its methods.

The activity also implements a local broadcast receiver, used to receive broadcasts from the background service. After a broadcast is received, the activity handles its content and according to its type performs action. For example if the Wifi connection is lost, the activity closes all fragments and shows notification that the user has to be connected to Wifi in order to control devices.

3.2.2 Fragments

In the mobile application are only four Fragments. The first one called `SplashFragment`, is shown at the start of the application and shows a progress circle and a text that the application is loading data from server. If an error occurs during the loading, this fragment shows appripriate error message and gives a user possibility to initiate the loading again.

From this fragment user can navigate to the `SettingsFragment`. In the `SettingsFragment` the user may change the server to which he want to connect. It is possible to start a search through a local network, that will find all servers connected to it.

The third fragment is `RoomListFragment` that contains a list of all rooms in a smart house. User can pick which room to open and get to the `RoomFragment` which will show all devices that can be controlled from the application.

3.2.3 WebSocketService

The `WebSocketService` is an implementation of a WebSocket client provided by Java-WebSocket library. It defines `OnWSMessageListener` interface, that is used to inform about a new WebSocket meesage received by the client.

3.2.4 WearUpdateService

`WearUpdateService` is subclass of `WearableListenerService` that is implemented in Google Wear API. The service receives all messages sent from wearable to the handheld. During the creation of this service a `ServiceConnection` to the `DataUpdateService` is created, so all messages are redirected to it. If the `ServiceConnection` is not created yet and some message is received, the message is put in a queue and delivered after the creation.

3.2.5 DataUpdateService

`DataUpdateService` is the main service of the mobile application. This service keeps all data about rooms and devices. The data classes are described in section 2.2 Design of data classes. When the service is created, it starts an `ApiCall`(described in subsection 3.2.6) to the server and fetches all information about rooms and devices and also an address of the WebSocket server. The address is used to start a `WebSocketService`. The `DataUpdateService` impelemts `OnWSMessageListener` interface, so every new WebSocket message is delivered to it and the service itself handles every message.

The service also handles all messages received from handheld as was described in the previous subsection.

3.2.6 ApiCall

An `ApiCall` is a class that extends `AsyncTask` and handles all requests to the local server. Basically it is a HTTP client that takes a `Request`, executes it and creates appropriate `Response`. The main work takes place in a background thread and the result is sent back in the UI thread.

3.2.6.1 Request

`Request` is an abstract class that represents a request that should be sent to a server. It contains:

- address to which it should be sent
- HTTP method that should be used (default GET)
- content type (default set to `application/json`)

The `Request` class defines two abstract methods, `toBytes` which should return data that will be sent to the server and `createResponse` whose return type is an abstract class `Response`.

The following list contains all subclasses of `Request`.

- **GetAPIRequest** - A request that retrieves information about API from the server.
- **GetRoomsRequest** - Request that retrieves all rooms in the smart house.
- **GetRoomRequest** - Request that retrieves information about specific room.
- **GetDeviceRequest** - Request that retrieves information about specific device.
- **GetDeviceStateRequest** - Request that retrieves information about device's state.
- **UpdateDeviceRequest** - Request that sends new device state to the server.

3.2.6.2 Response

`Response` is an abstract class that contains all data retrieved from server after a specific `Request` was sent. The `Response` contains the URL of the request and HTTP status code. This class defines abstract method `parse(String input)` which should handle the input data retrieved from server and parse them into corresponding data class.

In the following list are all subclasses.

- **GetAPIResponse** - Response that handles information about server API and returns `APIEndpoints` object.
- **GetRoomsResponse** - Response that returns list of all room's IDs.
- **GetRoomResponse** - Response that returns parsed `Room` object.
- **GetDeviceResponse** - Response that returns parsed `Device` object.
- **GetDeviceStateResponse** - Response that fills `Device` with information about its state.
- **UpdateDeviceResponse** - Response that returns a new device's state.

All the parsing is quite trivial except for parsing a `Device`, because it is a abstract class. The parsing has to handle creating corresponding subclass of the `Device` class. It is solved in a `Device`'s static method called

createDeviceFromJson that resolves the appropriate device type from actions list. As I have described the issues with JSON returned from the server in the subsection 3.1.1 Gson, the list of actions does not have a fixed structure. I therefore created a static method addActions that iterates through the actions list and if the item is also a list, the method recursively calls itselt, if the item is a String it is added to an output set. After the recursive method returns the output set, the corresponding subclass is created, depending on what actions the set contains.

3.3 Implementation of the wearable application

In the wearable application are two activities, MainActivity and RoomActivity, both extend from BaseActivity. The base parts of the application are also two services DataUpdateService and WearUpdateService.

3.3.1 BaseActivity

The abstract class BaseActivity contains methods used to bind to the DataUpdateService and create a ServiceConnection with the same purpose as MainActivity in the mobile application.

3.3.2 MainActivity

MainActivity is used in the same way as the SplashFragment in the mobile app. It waits until data is received in the DataUpdateService from the mobile application. If an error occurs, its description is shown and a restart button appears. If the wearable is not paired with any handheld, the data load does not even start.

3.3.3 RoomActivity

After all data is received the RoomActivity is opened. The activity contains a GridViewPager that is basically a 2D array of Fragments as described in subsection 2.3.2. Every row represents one room in the smart house. Fragments in a first column of every row contains name of the room and fragment in the last column contains button that opens the specific room in the mobile application. Every fragment between the first and last represents one device and takes care about its control.

The device fragments are:

- **LightFragment** - Fragment that represents a simple light with on/off functionality.

- **DimmableLightFragment** - Fragment that represents a light with adjustable brightness.
- **RGBLightFragment** - Fragment that represents a light that can change colors and brightness.
- **BlindsFragment** - Fragment that controls blinds. Blinds can roll up, down or stop the movement.
- **ThermoFragment** - Fragment that represents a thermostat. It shows actual temperature in room, requested temperature and allows correction of the requested temperature from -5°C to +5°C.

3.3.4 WearUpdateService

The `WearUpdateService` has the same purpose as in the mobile application described in subsection 3.2.4.

3.3.5 DataUpdateService

`DataUpdateService` from the wearable app is also very similar to the `DataUpdateService` from the mobile app, but it does not communicate directly with the server. The wearable service communicates with the mentioned `DataUpdateService` from the mobile application and it provides the wearable service all data. The mobile service acts as intermediary in all communication. If a user of the wearable app wants to switch a light off, a request is sent to the mobile service and it forwards the request to the server. It is because the wearable application itself can not open any network connection.

When the wearable `DataUpdateService` is created, it checks if the wearable is paired with a handheld. If it is paired, the wearable service initiates a data transfer from mobile service by sending an asynchronous request. The mobile service should send back a response containing all rooms and devices. If an error occurs the response contains an error code describing what error happened. If no response is received until a timeout is reached, the wearable service assumes that the connection was corrupted and a loading error is shown.

3.4 Communication between wearable and mobile applications

The wearable and mobile applications are two separate modules that have to communicate with each other to be kept informed. The communication is performed through Google Wear API contained in Google Play Services, specifically through Messages API. The Messages API offers point to point message delivery.

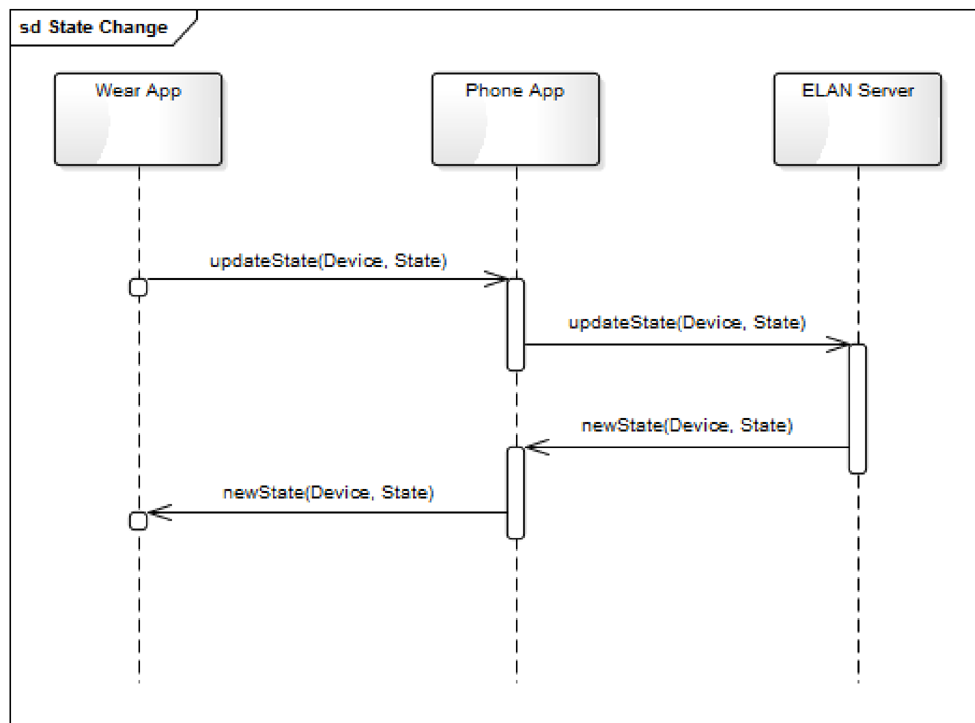


Figure 3.2: The procedure of a state change from wearable application.

3.4.1 Device state change

When the device state is changed from the mobile application, the mobile service sends the state change request to the server. If the request successes, the mobile service sends a state update to the wearable service if there is a wearable connected.

If the state is changed from the wearable application, the wearable service has to send request to the mobile service. The mobile service then sends the state change request to the server. If the request successes, the mobile service sends a state update back to the wearable. This procedure is described in the sequential diagram in figure 3.2.

When the state change happens outside of the wearable and mobile applications, the mobile `WebSocketService` should receive a message from ELAN WebSocket server that a change occurred. The message contains URL of the resource that has changed state. The mobile service should fetch new state and then send it to the wearable service if there is a wearable connected. The procedure initiated by WebSocket message is described in the sequential diagram in figure 3.3.

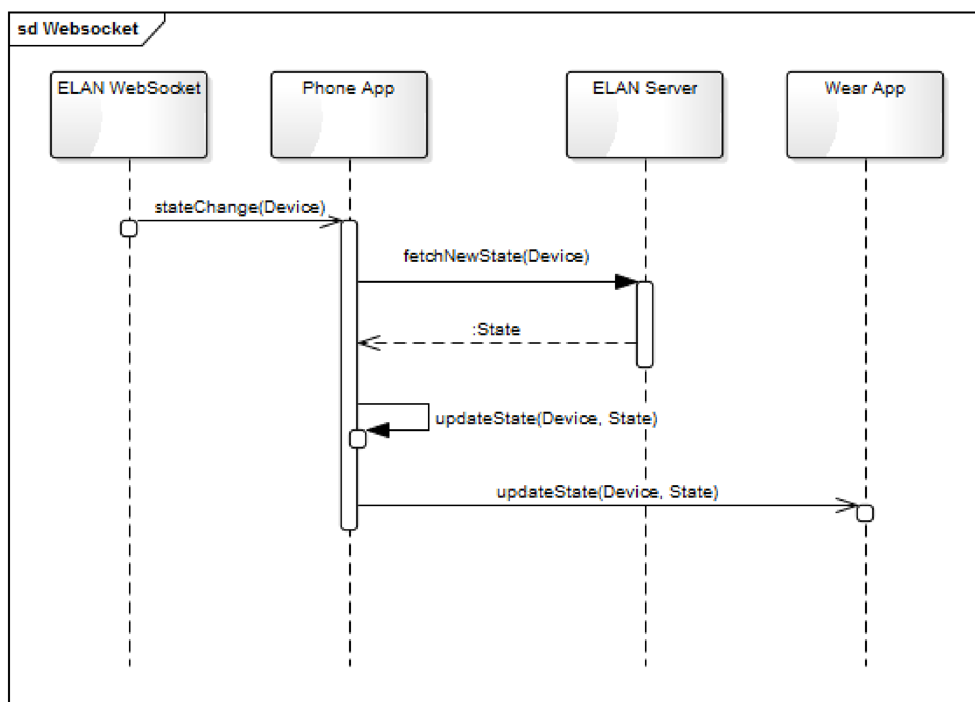


Figure 3.3: The procedure of a state change from WebSocket message.

Chapter 4

Testing

This chapter describes possibilites of testing an Android application and how the methods were applied. The chapter also contains section about usability testing of the application on users.

4.1 Testing on Android

The Android SDK provides powerful testing tools based on JUnit, extended with Instrumentation framework and specific Android testing classes such as `AndroidTestCase` or `ActivityUnitTestCase`[18]. The diagram 4.1 summarizes the Android testing framework. The SDK also provides monkeyrunner, an API for testing devices with Python programs, and UI/Application Exerciser Monkey, a command-line tool for stress-testing UIs by sending pseudo-random events to a device.

For testing the application two types of tests were used. First type are JUnit tests based on base class `AndroidTestCase`. These tests are targeted on testing communication with the server, which is performed by `ApiCall` class, described in subsection 3.2.6. Second type are Instrumentation tests based on class `ActivityInstrumentationTestCase2`. These tests are testing Fragments and their UI, for example if a Widget responds correctly and is on right place. Instrumentation testing is independent on Android lifecycle.

4.2 Usability testing

Jakob Nielsen (a usability guru) considers User tests more useful than Usability tests. He wrote: *Elaborate usability tests are a waste of resources. The best results come from testing no more than 5 users and running as many small tests as you can afford* [19].

According to the graph 4.2, testing with 5 users reveals about 80% of all usability problems. With every other user, we learn less about the usability,

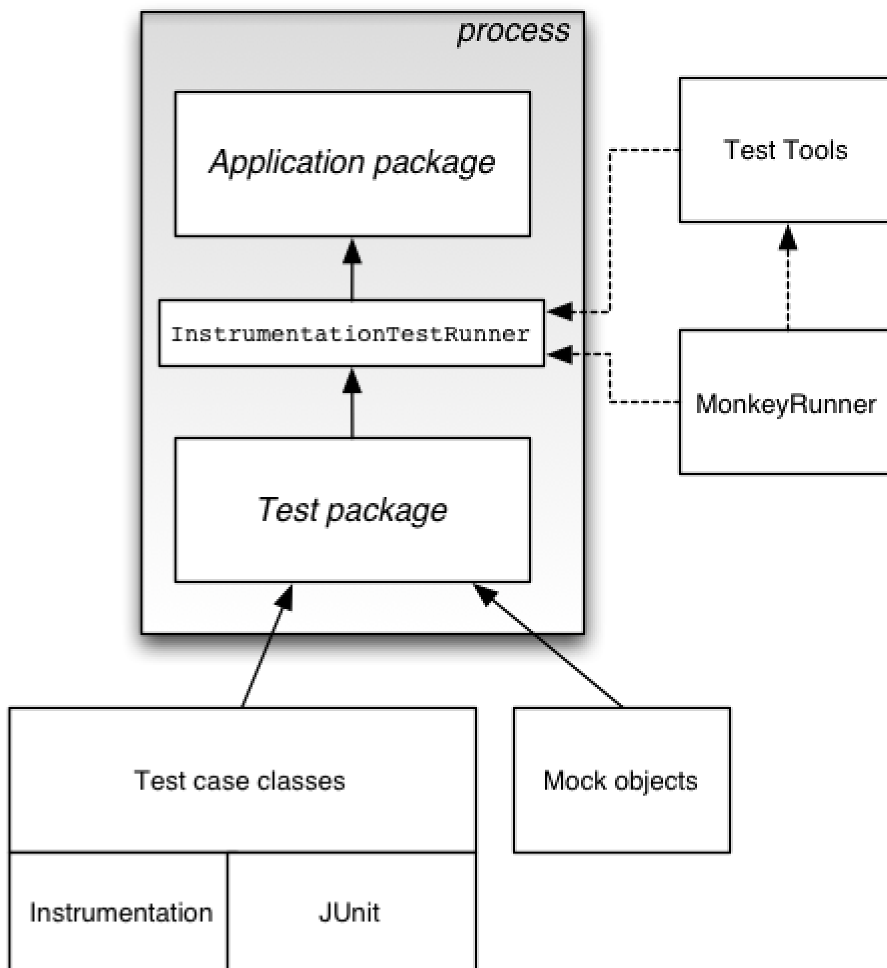


Figure 4.1: The Android testing framework.

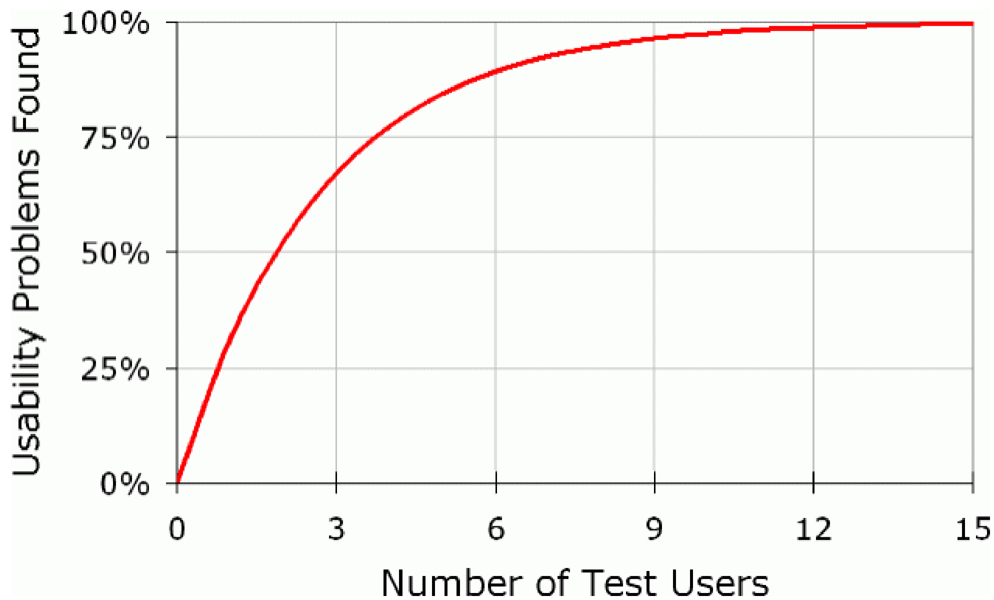


Figure 4.2: Number of problems found depending on number of users.

because the problems will overlap.

4.2.1 Usability Heuristic

Before the testing on actual users I performed a usability heuristic evaluation that introduced Jakob Nielsen[20]. The heuristic has following points:

- **Visibility of system status** - The system should always keep users informed about what is going on, through appropriate feedback within reasonable time.
- **Match between system and the real world** - The system should speak the users' language, with words, phrases and concepts familiar to the user, rather than system-oriented terms. Follow real-world conventions, making information appear in a natural and logical order.
- **User control and freedom** - Users often choose system functions by mistake and will need a clearly marked "emergency exit" to leave the unwanted state without having to go through an extended dialogue. Support undo and redo.
- **Consistency and standards** - Users should not have to wonder whether different words, situations, or actions mean the same thing. Follow platform conventions.

- **Error prevention** - Even better than good error messages is a careful design which prevents a problem from occurring in the first place. Either eliminate error-prone conditions or check for them and present users with a confirmation option before they commit to the action.
- **Recognition rather than recall** - Minimize the user's memory load by making objects, actions, and options visible. The user should not have to remember information from one part of the dialogue to another. Instructions for use of the system should be visible or easily retrievable whenever appropriate. (Read full article on recognition vs. recall in UX.)
- **Flexibility and efficiency of use** - Accelerators – unseen by the novice user – may often speed up the interaction for the expert user such that the system can cater to both inexperienced and experienced users. Allow users to tailor frequent actions.
- **Aesthetic and minimalist design** - Dialogues should not contain information which is irrelevant or rarely needed. Every extra unit of information in a dialogue competes with the relevant units of information and diminishes their relative visibility.
- **Help users recognize, diagnose, and recover from errors** - Error messages should be expressed in plain language (no codes), precisely indicate the problem, and constructively suggest a solution.
- **Help and documentation** - Even though it is better if the system can be used without documentation, it may be necessary to provide help and documentation. Any such information should be easy to search, focused on the user's task, list concrete steps to be carried out, and not be too large.

The only problem in my application is that it does not have Help screen. However it is quite simple app, so I assume that it is not a fatal error.

4.2.2 User testing

The user testing was performed on six users of different ages and different technical skills. Testing was also attended by representatives of the client. The testers received list of tasks and I watched their progress. The tasks were:

- Start the mobile application and find local server.
- Open Bathroom and change the temperature to 25°C on mobile phone.
- Change the bulb color to green on mobile phone.

- Turn off the main light in Living room on the smart watch.
- Change the brightness of a bulb to 60% on a smart watch.

The first problem that occurred was that the users were not familiar with smart watches. At first the had problems with navigation in the wearable application. However no changes were made, because the navigation is made according to standards defined by Google. After few minutes of work with the smart watches the users learned to control the wearable application quite well. With the mobile application, there were no problems at all.

The testing was quite positive and the representatives of the client were really pleased by the work that has been done.

Conclusion

After a lot of work done on desing and development of this application prototype, the prototype is fully functional. The development of the mobile and wearable applications was successful and all planned features were implemented. Also all functional and non-functional requirements were met. The analysis revealed few problems with the server API developed by the client, but all of them were solved. Durign design of the application, my supervisor and I discussed a lot about user interface and its correct design, because the usability on smart wacthes was the most importatnt requirement for the application.

I have chosen this masters's thesis topic beacause I wanted to learn how to develop applications for Android Wear platform. I think that in the near future the demand for Android Wear developers will grow. Also the topic is really interesting, it is expected to use the latest technology and develop an application for high tech fast growing company ELKO EP s.r.o..

When the development was finished and we (my supervisor and I) presented the final application prototype to the clients, they were really pleased. They want us to continue to cooperate and participate in the further development of a new application based on this prototype.

Bibliography

[1] Loxone. THE LOXONE SOLUTION. 2015, [Cited 2015-04-15]. Available from: http://www.loxone.com/enen/start.html

[2] Nest. 365 Days with Nest. 2015, [Cited 2015-04-15]. Available from: https://nest.com/thermostat/life-with-nest-thermostat/

[3] Lucis Technologies Inc. NuBryte FAQ. 2015, [Cited 2015-04-15]. Available from: http://www.nubryte.com/faq

[4] IDC. Smartphone OS Market Share, Q4 2014. February. Available from: http://www.idc.com/prodserv/smartphone-os-market-share.jsp

[5] Ben Elgin. Google Buys Android for Its Mobile Arsenal. August 2005, [Cited 2015-04-10]. Available from: http://www.businessweek.com/stories/2005-08-16/google-buys-android-for-its-mobile-arsenal

[6] Google Inc. App Framework. February 2015, [Cited 2015-05-01]. Available from: http://developer.android.com/about/versions/index.html

[7] Google Inc. Platform Versions. April 2013, [Cited 2015-04-16]. Available from: http://developer.android.com/about/dashboards/index.html

[8] Google Inc. Activities. 2015, [Cited 2015-04-25]. Available from: http://developer.android.com/guide/components/activities.html

[9] Google Inc. Fragments. 2015, [Cited 2015-04-25]. Available from: http://developer.android.com/guide/components/fragments.html

[10] Google Inc. Services. 2015, [Cited 2015-04-25]. Available from: http://developer.android.com/guide/components/services.html

[11] Dante D'Orazio. Google reveals Android Wear, an operating system for smartwatches. March 2014, [Cited 2015-04-17]. Available from: http://www.idc.com/prodserv/smartphone-os-market-share.jsp

[12] Google Inc. Android Studio Overview. 2015, [Cited 2015-04-22]. Available from: http://developer.android.com/tools/studio/index.html

[13] Internet Engineering Task Force (IETF). The WebSocket Protocol. December 2011, [Cited 2015-04-10]. Available from: http://tools.ietf.org/html/rfc6455

[14] Internet Engineering Task Force (IETF). The JavaScript Object Notation (JSON) Data Interchange Format. March 2014, [Cited 2015-04-10]. Available from: http://tools.ietf.org/html/rfc7159

[15] Google Inc. Google-gson. May 2015, [Cited 2015-05-01]. Available from: https://code.google.com/p/google-gson/

[16] Trey Robinson. 5 Reasons You Should Use Butterknife For Android. September 2013, [Cited 2015-05-01]. Available from: https://code.google.com/p/google-gson/

[17] Wayne Chang. Milestone Achieved: Over 1 Million Apps! April 2015, [Cited 2015-05-01]. Available from: http://www.crashlytics.com/blog/milestone-achieved-over-1-million-apps

[18] Google Inc. Testing. May 2015, [Cited 2015-05-02]. Available from: http://developer.android.com/tools/testing/index.html

[19] Jakob Nielsen. Why You Only Need to Test with 5 Users. March 2000, [Cited 2015-05-01]. Available from: http://www.nngroup.com/articles/why-you-only-need-to-test-with-5-users/

[20] Jakob Nielsen. 10 Usability Heuristics for User Interface Design. January 1995, [Cited 2015-05-01]. Available from: http://www.nngroup.com/articles/ten-usability-heuristics/

Appendix A

Acronyms

GUI Graphical user interface

UI User interface

app Application

apps Applications

API Application programming interface

SDK Software development kit

JSON JavaScript Object notation

RAM Random access memory

VM Virtual machine

IDE Integrated development environment

GUID Globally unique identifier

XML Extensible markup language

REST Representational state transfer

RGB Red, green, blue

HTTP Hypertext transfer protocol

URL Uniform resource locator

FAQ Frequently asked questions

Appendix B

Contents of enclosed CD

```
readme.txt ...................... the file with CD contents description
bin ...................................... the directory with binaries
src .................................. the directory of source codes
  app ........................................ implementation sources
  thesis ............. the directory of LaTeX source codes of the thesis
text ...................................... the thesis text directory
  thesis.pdf ........................... the thesis text in PDF format
```