VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

PORT OF OPTAPLANNER ON ANDROID

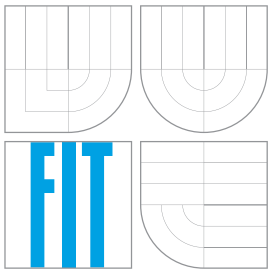
DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

TOMÁŠ DAVID

BRNO 2015

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

PORTACE NÁSTROJE OPTAPLANNER NA ANDROID

PORT OF OPTAPLANNER ON ANDROID

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

TOMÁŠ DAVID

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. ZDENĚK LETKO, Ph.D.

BRNO 2015

Abstrakt

Tato práce se zabývá portací nástroje OptaPlanner na operační systém Android. OptaPlanner je nástroj pro řešení plánovacích problémů a je kompletně napsán v programovacím jazyce Java, který je také využíván pro vývoj aplikací operačního systém Android. Ten však neobsahuje všechny knihovny z Java Standard Edition Application Programming Interface a při portaci nástroje OptaPlanner na Android tak dochází k problémům se závislostmi. Výsledkem této práce je návrh a implementace řešení výše zmíněných problémů a ukázková aplikace věnující se problému okružních jízd, který je řešen pomocí portovaného nástroje OptaPlanner.

Abstract

This thesis deals with portation of the OptaPlanner tool to the Android operating system. The OptaPlanner is used for solving planning problems and it is completely written in the Java programming language which is also used for application development of the Android operating system. However, Android does not contain all of the Java Standard Edition Application Programming Interface libraries and porting of OptaPlanner to Android thus causes dependency problems. The result of the thesis is solution design and implementation of the problems mentioned above and model Android Vehicle Routing Problem application which uses ported OptaPlanner tool.

Klíčová slova

OptaPlanner, Android, Java, portace, problém okružních jízd

Keywords

OptaPlanner, Android, Java, portation, Vehicle Routing Problem

Citace

Tomáš David: Port of OptaPlanner on Android, diplomová práce, Brno, FIT VUT v Brně, 2015

Port of OptaPlanner on Android

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením pana Ing. Zdeňka Letka, Ph.D. Další informace a pomoc mi poskytl Geoffrey De Smet ze společnosti Red Hat. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Tomáš David
May 26, 2015

Poděkování

Na tomto místě bych rád poděkoval mému vedoucímu Ing. Zdeňku Letkovi, Ph.D. a mému konzultantovi Geoffrey De Smetovi za cenné rady, připomínky a za čas, který mi věnovali. Díky patří také mé přítelkyni a mým rodičům za jejich podporu a pomoc při studiu.

Contents

Chapter 1

Introduction

Porting of the applications becomes very current issue in the modern world of information technology. Every operating system uses its own interface and technologies for application development. However, there are cross-platform tools that allow porting of the applications without many modifications. One of these tools is the Java programming language.

Java language is developed by Oracle Corporation and it is distributed in several platforms. The most common platform is Java Standard Edition (SE) which contains many of the basic Java libraries commonly used in standard desktop applications. A set of these libraries is called the Java SE Application Programming Interface (API).

Android is an operating system for mobile devices developed by Google. It uses the Java programming language to an application development. Android runtime environment includes not only the libraries for development of graphical user interface but also a subset of Java SE API libraries.

OptaPlanner is an open-source software developed by JBoss community designed for solving planning problems. It is completely written in Java language and it is easily portable between desktop operating systems. However, Android API does not contain all of the Java SE API libraries and therefore porting of the OptaPlanner tool causes problems with dependencies. This thesis deals with these problems and shows an implementation of a simple Android application which uses OptaPlanner tool to solve the Vehicle Routing Problem.

Along with this introduction, this thesis is divided into another seven chapters. Chapter 2 presents the Java programming language and its platforms. Chapter 3 describes the Android platform, its architecture and the build process. Chapter 4 deals with the OptaPlanner tool and shows how it can be used. In Chapter 5, differences between Java SE API and Android API are described and the possible solutions of the JavaBeans problem on the Android platform are suggested. One of the solutions is selected and used for realization of the portation. Design and implementation of model Android Vehicle Routing Problem application using OptaPlanner tool is presented in Chapter 6. Performed testings and future work is described in Chapter 7 and the last Chapter 8 summarizes the entire work.

Chapter 2

Java

This chapter introduces the Java programming language. It is primarily focused on description of its platforms and specially on Java Standard Edition and its Application Programming Interface (API) which serves as the basis for Android API described in Chapter 3.

Section 2.1 presents the Java programming language, Java platforms are described in Section 2.2 and Section 2.3 introduces Java Standard Edition and its main parts.

2.1 Java language

Java [6, 3] is one of the most famous and most widely used computer programming languages in the world. It is developed by Oracle Corporation and its application is widespread. Java is used for programming smart cards, mobile and desktop applications, as well as large business and information systems. It is class based and object oriented language which is managed by the Java Language Specification and together with the supporting runtime forms programming environment.

The first public version of Java was released in 1996 and since this year, eight more versions was released. In 2010, Java changed its owner from Sun Microsystems to Oracle Corporation. The latest version of Java (Java 8) was released in 2014.

2.2 Java platforms

Java is published in four platforms. Each platform provides tools for development and running programs written in Java and consists of two main parts. The first part is Java Virtual Machine (JVM) which is connected to an operating system and thus Java programs can be executed. The second part is Java Application programming interface which provides many public classes of standard Java libraries.

The following paragraphs briefly describe four platforms: Java Standard Edition (Java SE), Java Enterprise Edition (Java EE), Java Micro Edition (Java ME) and Java Card.

Java Standard Edition Basic and the most famous platform which is designed for desktop and simple server application development. Currently, the most recent version is Java SE 8.

Java Enterprise Edition Extension of Java SE that contains special libraries for developing and running enterprise software applications and information systems. Java EE is

based on Java SE 7 in the current version.

Java Micro Edition Subset of Java SE for application development for small devices such as microcontrollers, mobile phones, set-top boxes, printers and other devices is called Java ME. Currently, the most recent version is Java ME 8.1.

Java Card This technology is designed for application development of smart cards or devices with limited memory and processing capabilities. For example, it is used for SIM cards of mobile devices, plastic smart card for Automated teller machine and similar devices. Last released version is Java Card 3.

2.3 Java Standard Edition

Java SE platform is distributed in two versions. The first version is Java Runtime Enviroment which is commonly used on personal computers for running Java applications. The second version named Java Development Kit is used mostly for application development. This platform has its own open-source implementation called OpenJDK [5]. Figure 2.1 shows parts of Java SE Development Kit (JDK) distribution in comparison with Java SE Runtime Environment (JRE) and in the following paragraphs, these components are briefly described.

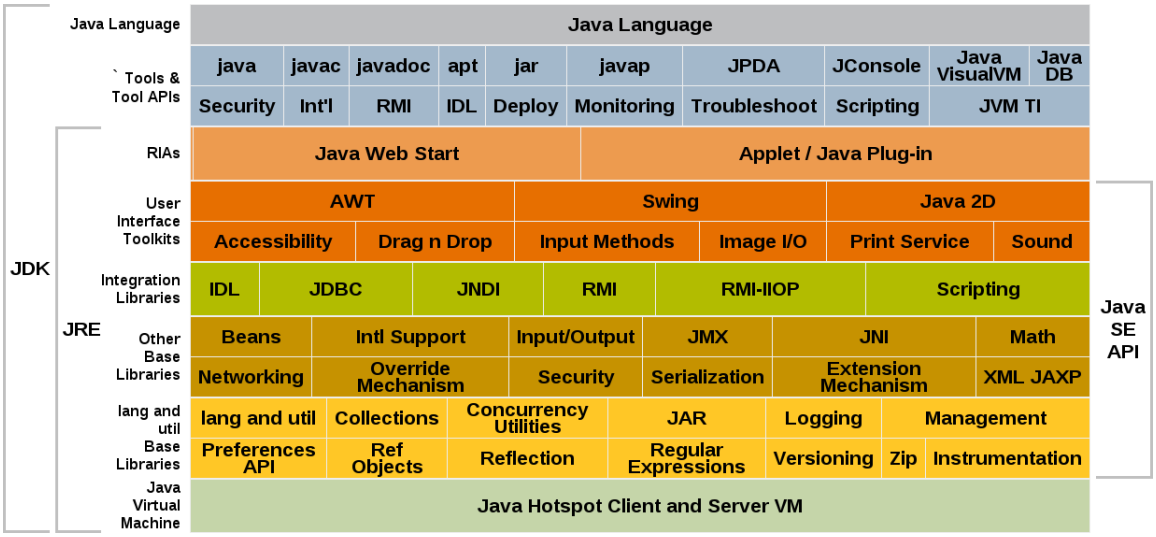


Figure 2.1: Components of the Java Standard Edition Development Kit 1.6 [3].

Java SE Development Kit Java SE Development Kit is sometimes known as Software Development Kit (SDK). It is tool containing everything necessary for developing Java application. The main part of JDK is JRE which is described below. Further, it contains the tools to create and build applications (such as compiler, documentation generator, etc.), security, localization and other tools. The last part of JDK is Java Language Specification which describes rules of this programming language.

Java SE Runtime Environment JRE is runtime environment for running Java applications. It consist from Java API, Java Virtual Machine and tools for creating rich internet applications.

Java SE Application Programming Interface Java API is set of public classes of standard libraries. These libraries include packages for creating graphical user interface, manipulation with databases, base language and utility libraries and many others.

Java Virtual Machine Java programs can not run without virtual machine. JVM is a program that provides the runtime environment necessary for executing of Java application. Figure 2.2 shows the lifecycle of Java program. It starts with Java source code which is compiled to bytecode by `javac` compiler. Bytecode is an instruction code and it is stored in `.class` files. These files go through classloading mechanism to JVM and then are ready for execution by the interpreter.

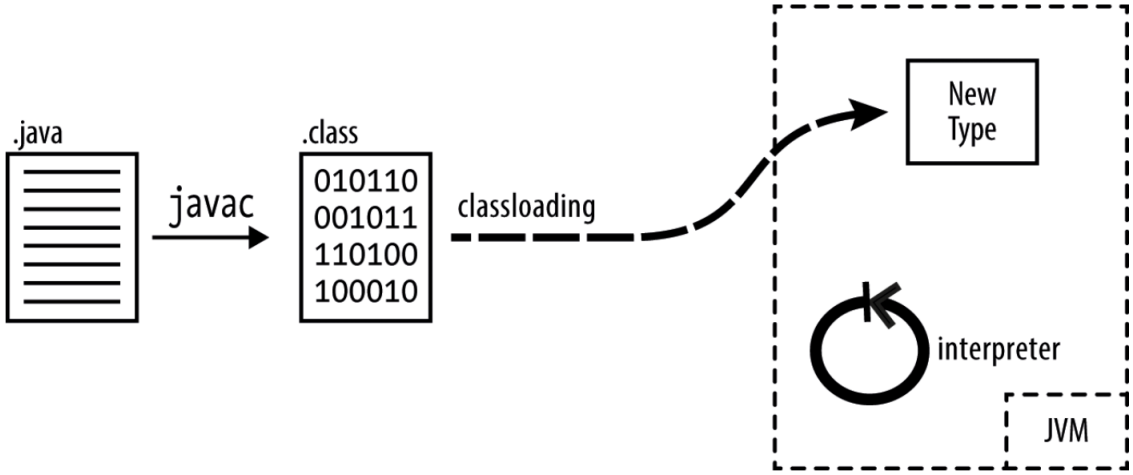


Figure 2.2: The lifecycle of a Java program [6].

Chapter 3

Android

This chapter deals with the Android operating system and its architecture. Android platform is quite different from Java platform. They use the same language for source code of applications and the build process is also similar but the differences cause that they are not fully compatible.

In Section 3.1, Android and its history is presented. Architecture of this platform is described in Section 3.2 and part of this architecture – Android runtime – is discussed in more detail in Section 3.3. Basic components of Android application are presented in Section 3.4 and the build process of Android source code and the development environment is described in Section 3.5.

3.1 Introduction

Android [1, 24] is a mobile operating system developed by Google. It is an open-source system based on the Linux kernel mainly used in mobile devices such as smartphones, tablets and smart watches but it can also be found in devices such as set-top boxes, media players and other electronics.

Android Inc. was founded in California USA in 2003. Google Inc. bought Android two years later. In 2007, Google acquired several patents in the field of mobile devices and at the same year on November 5, there was an official presentation of companies association (Open Handset Alliance) which aims to create open standards for mobile devices. The first smartphone with Android released on October 22, 2008.

Table 3.1 presents a brief history of the Android operating system. It describes release dates of major versions, version numbers, code names, API levels and distributions in the last column. As can be seen from version 1.5, every major version has its own codename represented by a popular food. API level indicates that version brings changes in its programming interface and the most important column shows the distribution of each version. According to this, it may be decided for which versions still makes sense to develop an application.

3.2 Architecture

This section deals with the architecture of Android system [23] which consists of six layers. Figure 3.1 and following paragraphs describe this architecture starting with the kernel layer and ending with the applications.

Release date	Version	Codename	API level	Distribution [%]
September 23, 2008	1.0 – 1.1	–	1 – 2	< 0.1
April 27, 2009	1.5	Cupcake	3	
September 15, 2009	1.6	Donut	4	
October 26, 2009	2.0 – 2.1	Eclair	5 – 7	
May 20, 2010	2.2 – 2.2.3	Froyo	8	0.3
December 6, 2010	2.3 – 2.3.7	Gingerbread	9 – 10	5.7
February 22, 2011	3.0 – 3.2	Honeycomb	11 – 13	< 0.1
October 18, 2011	4.0 – 4.0.4	Ice Cream Sandwich	14 – 15	5.3
July 9, 2012	4.1 – 4.3	Jelly Bean	16 – 18	39.2
October 31, 2013	4.4 – 4.4.4	KitKat	19 – 20	39.8
November 12, 2014	5.0 – 5.1.1	Lollipop	21 – 22	9.7

Table 3.1: Android version history [14].

Linux kernel The lowest layer stands between hardware device and other architecture layers. Android is based on a special version of Linux kernel and several accessories such as memory management system, the Binder IPC driver and others. Since the beginning, Android was built on the Linux 2.6 kernel. The latest version 5 from Table 3.1 runs on the kernel 3.4.

Hardware abstraction layer Hardware abstraction layer (HAL) is standard interface which allows to Android system calls drivers layer while it does not have to care what is the implementation of drivers and hardware in lower layers. For each piece of hardware should be a driver and matching HAL providing hardware options.

Libraries Above the HAL is a layer of native libraries. These libraries are written in C or C++ language and they can be accessed through the Android Standard Development Kit (SDK) but if direct access is required, it is possible to do that through the Native Development Kit (NDK). The layer includes the following libraries:

- **Surface manager** – library for composing different drawing surface windows on the screen.
- **Media Framework** – provides various multimedia codecs for playing and recording video in various formats.
- **SQLite** – a database engine for the use in data storage.
- **WebKit** – a browser engine for displaying web content.
- **Libc** – the standard library of the C programming language.
- **OpenGL ES** – a library for support 2D and 3D graphics and hardware accelerated rendering.
- **Audio Manager** – a library for working with audio of device.
- **FreeType** – a library for bitmap and vector font rendering.

- **SSL** – a library for the use of encryption protocol for secure internet communications.

Android runtime Android runtime layer is located next to the native libraries. This layer consist of two parts. The first one is the Core Libraries and the second one is the Dalvik Virtual Machine. The Core libraries can be further subdivided into two parts: Java libraries and Android libraries. Section 3.3 describes the details of this layer.

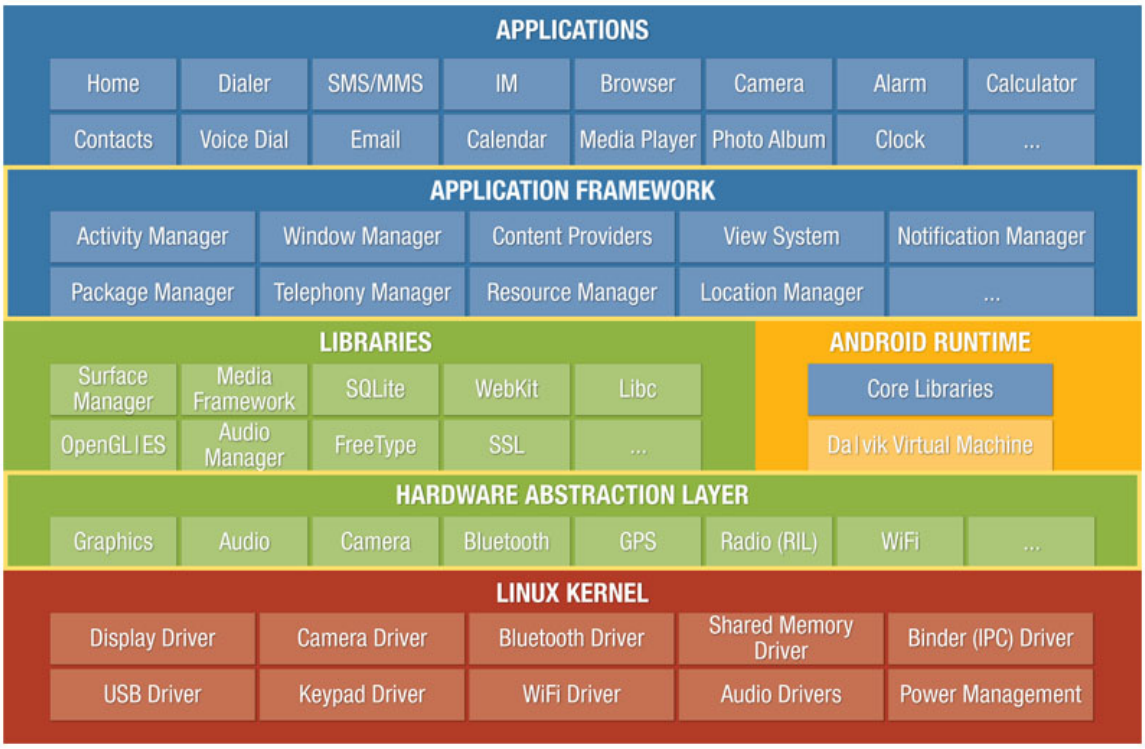


Figure 3.1: Android architecture [21].

Application framework Application framework layer provides many high-level services to applications in the form of Java libraries. For developers, this is the most important layer that allows access to the device services. The Application framework includes the following parts:

- **Activity manager** – controls all aspects of running activities. It manages all Services and Activities described in Section 3.4.
- **Windows manager** – provides services for windows management such as visibility and arrangement. It also takes care about animation on the screen.
- **Content Providers** – allows to work with a content of other applications, it encapsulates the data and provides mechanisms for defining data security.
- **View System** – a set of basic blocks that is used to build the application user interface. The basis of every graphical component is the `View` class which is represented by a rectangular area on the screen.

- **Notification manager** – provides the possibility to inform the user about some action that happened in the background. Notifications can take different forms like LED flashing, vibrating, ringing or notification on the screen.
- **Package manager** – allows obtaining of various information about applications that are currently installed on device.
- **Telephony manager** – provides access to telephone services of device. These services include SIM card, network, cellphone or other information.
- **Resource manager** – allows access to non-code resources such as color settings, layouts and strings. Separating of resources from code helps to better manage the various characteristic without modifying the code.
- **Location manager** – provides access to the location services. These services allow periodical receiving of geographic coordinates and therefore it is possible to track device current location.

Applications The last and highest layer consists of the applications themselves. These comprise both pre-installed applications and applications that have been added over time from the Android store or another way.

3.3 Android runtime

Both Android runtime and Java Runtime Edition include a virtual machine and Java SE API. In case of Android, its runtime contains special virtual machine named Dalvik Virtual Machine (DVM) and API lacks several Java packages and classes and contains special Android libraries.

Dalvik Virtual Machine

DVM is a virtual machine that is developed since 2005 and it is included into the system due to the fact that JVM was not licensed as open-source in the past. The second reason is the optimization for mobile devices.

Each application runs on an Android device within its own instance of DVM (i.e. not as a process in the Linux kernel).

Running applications on a virtual machine has many advantages. First, it operates in a sandbox and thus can not interfere with the operating system or other applications. Secondly, it makes the application platform independent and therefore can be run on any hardware. Advantages also include memory usage which makes the DVM better adaptable for use on mobile devices.

An application code must be always transformed from a standard `.java` file into byte-code but it is different in Java applications after this point. Bytecode is not executed by DVM but instead of that it is converted by Dex tool to Dalvik executables (`.dex` format). The whole process is described in Section 3.5.

Java libraries Most of Android applications are written using Java. Android contains libraries based on Apache Harmony Project that is an open-source Java implementation. These libraries are a subset of the Java SE platform. They do not contain all of the

packages. For example, `java.awt` or `java.swing` libraries are replaced by Android user interface classes and packages. More detailed comparison can be found in Chapter 5.

Android libraries Android libraries contain specific packages that provide all the functionality of Android devices. Libraries are written in Java and they include the following packages:

- `android.app` – provides access to the application model and it is the base of all applications.
- `android.content` – classes for accessing and publishing data applications.
- `android.database` – classes for data access and database manipulation.
- `android.graphics` – library for screen low-level 2D graphics drawing.
- `android.hardware` – provide access to hardware features such as cameras and sensors.
- `android.media` – library for handling with multimedia.
- `android.text` – library for manipulation and rendering of strings.
- `android.util` – common tools such as data manipulation and time utilities, conversions between numbers and strings and other classes.
- `android.view` – basic library for building a graphical user interface.
- `android.webkit` – libraries for working with web content.

3.4 Application structure

In this section, anatomy of application is described. Application consist of the various components and blocks whose description follows.

Activities Activities represent one single screen of user interface. Typically after an application start, the main activity is displayed and from there, another activity can be started or other operations can be performed. When a new activity is started, the previous one is stored in a Last In First Out stack. After pressing the back button, the stored Activity is invoked again.

Fragments Fragment is a modular section of an activity which has its own lifecycle. They represent a portion of Activity, one Activity can contain more fragments and they are used for adapting the layout of components on various screens.

Services Services are components that run in the background performing long-term tasks. They do not provide a user interface. Services may be still active in the background while other applications is running. For example, a service might be playing music or downloading from the Internet.

Content providers Content providers store and load data to make them available for other applications. Through the content providers, other applications may modify or manipulate specific data. An example might be a content provider that manages the contact information on the device.

Intents Intents are messages between one component and other component. Intent can be sent to inside or outside of an application and it means that one application can communicate with another. Three types of application components are supported for Intents communication – Activities, Services, and Broadcast receivers – and there are two types of intents – Explicit and Implicit. Explicit intents specify the component to start by name. Implicit intents do not specify the component, another application can handle it and makes appropriate response.

Broadcast receivers These components are used to listen message notifications (Intents) from outside (other application or system) or from inside of the application. After receiving the message, receivers can initiate appropriate reaction. An example might be an incoming Short Message Service (SMS) or low battery notification.

Application resources Android application consists not only of Java files but also of the resources that are separated from the source code. These resources include:

- **Animation resources** – define predefined animation.
- **Color state list resources** – define color change based on the `View` class state.
- **Drawable resources** – define different bitmaps or XML (eXtensible Markup Language) graphics files.
- **Layout resources** – define the layout of application components.
- **Menu resources** – define content of application menus.
- **String resources** – define string and string arrays.
- **Style resources** – define the appearance and format of ui elements.

Application manifest file Each application must have an `AndroidManifest.xml` file. This file contains information about application with regard to Android and it has to be placed in root directory of project. It has to contain a unique package name for the application, the declaration of used Activity and Service components, application permissions to the protected parts of the API (such as access to the camera, etc.). A minimum API level, list of connected and used libraries and other important information about application have to also be declared. Android manifest is together with resources compiled by `aapt` tool to `R.java` file. Thanks to this, the manifest information are available in the source code.

3.5 Build process

Build process is the way how `.apk` package is produced from Android project. APK (Android application package) file is a file format used for distribution and installation of

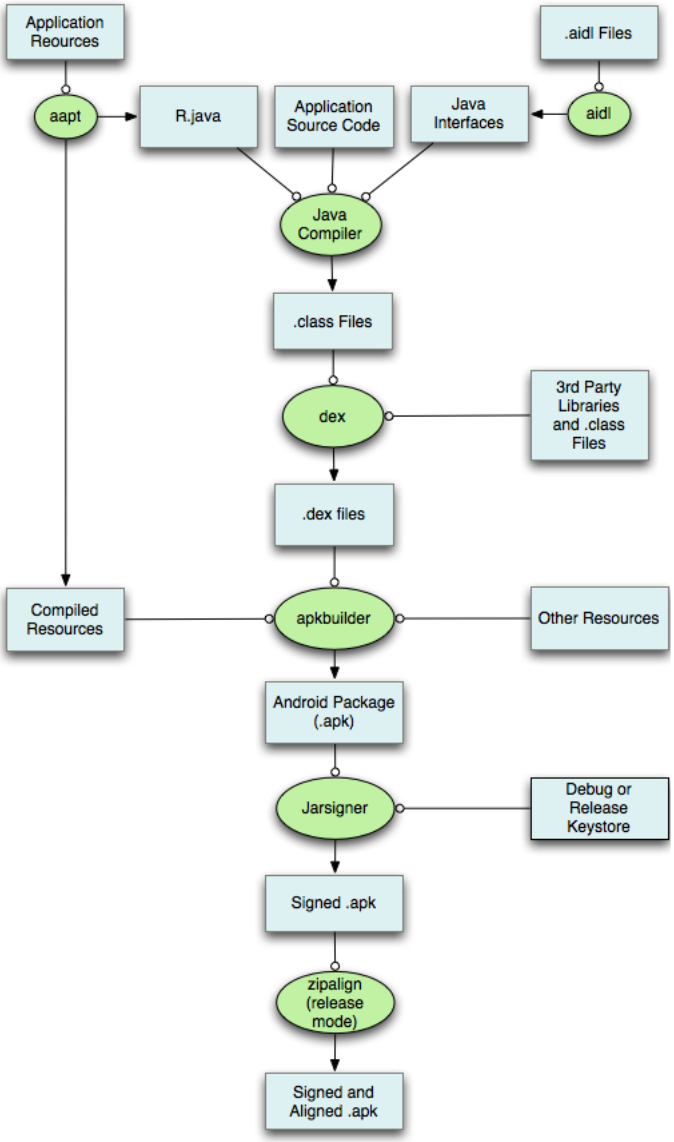


Figure 3.2: Android build process [13].

application software to Android devices. It contains all of the necessary files to run an application. Figure 3.2 shows how APK package is created from Java source code and resources including Android manifest file. The individual steps are:

1. The Android Asset Packaging Tool compiles resource files and `AndroidManifest.xml` and produces `R.java`. Resource references from Java code are then linked to this file.
2. `aidl` tool coverts all `.aidl` interfaces to Java interfaces. Aidl files are interfaces used for interprocess communication between client and service.
3. Java compiler takes and compiles `R.java` file, application source code and Java interfaces to bytecode. Bytecode is stored in `.class` output files.
4. Any third party libraries and `.class` files are converted by Dex tool to Dalvik byte code which is stored in `.dex` files.

5. Other uncompiled resource (eg. images), compiled resource and `.dex` files are processed by `apkbuilder` tool which packs them to `.apk` file.

6. Once the `.apk` file is created, it must be signed by debug or release key. `jarsigner` tool provides signing and creates signed `.apk` file.

7. Finally, if the application is being signed in release mode, `zipalign` tool align the `.apk` file. It creates signed and aligned `.apk` file which is not so memory consuming.

Following paragraphs presents environment for developing of Android applications and its build tool. These two systems automate development and the build process which is described above.

Android studio In the past, Google has supported two developer tools for developing Android applications. From the very beginning, Eclipse with Android plugin was primary environment. Nowadays, the official environment for development of Android applications is called Android studio [13]. It is a multi-platform application which provides tools to facilitate application development. Android studio is based on the IntelliJ IDEA Java IDE (Integrated Development Environment) and Gradle language is used as default build system.

Gradle build system Gradle build system [15] is a project automation tool which is built on the concepts of Apache Ant [8] and Apache Maven [9] tools. It is used as default build system in Android studio and using Tasks from Android Gradle plugin automates the entire build process described above. A Task represents a process which launches some sections of Gradle build code. Standard Tasks for every Android application are:

- `gradle clean` – cleans project and deletes the build directory.
- `gradle assemble` – builds the project.
- `gradle check` – runs checks and tests.
- `gradle build` – runs both assemble and check.
- `gradle assembleDebug` – creates debug `.apk` file.

Chapter 4

OptaPlanner

In this chapter, the planning engine Optaplanner is presented. OptaPlanner [16, 2] is an open-source project developed by JBoss community since 2006 and it is software designed for solving planning problems. Optaplanner is a part of the KIE (Knowledge Is Everything) group project [18, 17] and it combines optimization algorithms with the core of rule engine – Drools Expert [17].

Section 4.1 describes what planning problem is and Section 4.2 presents basic terminology. Introduction of basic phases of the OptaPlanner configuration is described in Section 4.3.

4.1 Planning problem

In everyday life, at work or in another occasion, people meet problems for which they have limited resources (time, money, etc.). Also, organizations need to face these problems at a larger scale. Planning mechanisms help them to save these resources and time.

Planning problems can be for example N-Queen problem, Vehicle Routing Problem, Course timetabling or Hospital bed planning. More examples can be found in [20]. Following paragraphs presents two of these problems.

N-Queen problem One of the planning problems is N-Queen problem. Although it is not very realistic, it is an ideal example. In this case, it is necessary to place `n` queens to the chess field of size `n`. It is known from the Chess game that queen can move vertically, horizontally and diagonally. The goal is to achieve that none of the queens should threaten another.

Vehicle Routing Problem The second example of the planning problems is Vehicle Routing Problem (VRP) [22]. Problem consists of depots, customers and vehicles. Vehicles depart from depots, collect objects of each customer and bring them back to the depot. Each vehicle can service multiple customers but it has a limited capacity. The goal is to minimize total traveled distance. There are several variants of this problem, for example Vehicle Routing Problem with time windows.

4.2 OptaPlanner terminology

In OptaPlanner, the planning problem is represented by Java classes, XML (eXtensible Markup Language) configuration file and optional DRL (Drools Rule Language) files. Java classes together form the model of the planning problem. Configuration file is written in XML and it serves to describe the configuration of Solver. Configuration contains links to classes of declared model, score configuration and used optimization algorithm. DRL files which are optional contain special rules and can be used for calculation of a score. Otherwise, the score can be calculated by a Java function. Important concepts from OptaPlanner terminology are described in the following paragraphs.

Solver A tool in OptaPlanner which solves planning problems is called Solver. It uses the problem model and calculates the score of possible solutions. Except of calculating the score, Solver uses optimization algorithms to find the best score of planning problem. End of calculation can be caused for example by finding of the best solution or by reaching of a time limit.

Solution Solution is an instance of the problem. There are two basic types of solutions in OptaPlanner – uninitialized and initialized solutions. In contrast to second type, the first one does not have the calculated score yet.

Score Score is a way how to compare two solutions of a problem. Every solution has own score and solution with higher score is better. There is significant difference between best and optimal solution. Solver always finds solution with the highest score from possible solutions – the best solution. Although it is not always the optimal solution which is the best solution of current problem. There are several techniques for comparing scores:

- **Score constraint signum** – based on constraints. Solver finds the highest score for the positive constraint and try to reduce a negative value for the negative constraints.
- **Score constraint weight** – technique where constraints may not have same weight and thus some of them can be more important then others. For example, the first condition is three times more important than the second condition.
- **Score level** – based on levels of score. Some scores are more important than others (Hard scores). Therefore, they are compared first and then it can be decided by some of the less important scores (Soft scores).
- **Pareto scoring** – Score constraints cannot be weighted against each other therefore they are compared individually and score with the most dominating score constraints wins.
- **Combining score techniques** – All the previous techniques can also be combined.

Drools and DRL Drools [17] is a business rules management system which provides way how to use rules, workflow and event processing. DRL file is one of the methods of storing business rules. OptaPlanner uses Drools as one of the options of score calculations which are are described in Section 4.3.2.

Optimization algorithms Every individual calculation of the solution takes some time. OptaPlanner does not count only with one solution but it is looking for the best solution and there may be a lot of solutions. The search space can grow to astronomical proportions and the calculation time as well. More information about optimization algorithms can be found in [20]. Following list shows some of the algorithms that can be used:

- **Exhaustive Search** – Brute Force, Branch And Bound.
- **Construction heuristics** – First Fit, Weakest Fit, Strongest Fit, ...
- **Metaheuristics**
  - **Local Search** – Hill Climbing, Tabu Search, Simulated Annealing, ...
  - **Evolutionary Algorithms** – Evolutionary Strategies, Genetic Algorithms.

4.3 OptaPlanner configuration

In this section, OptaPlanner configuration is described. It can be divided into following five basic steps that are required to get the best solution:

1. **Modeling of planning problem** – creation of a class that implements the `Solution` interface and definition of planning domain classes.
2. **Solver configuration** – settings of a score function, optimization algorithms and other parameters of Solver.
3. **Loading of problem data set** – insertion of planning entities and variables instances into Solver.
4. **Activation of Solver** – activation of mechanism for problem solving and automatic calculation of scores.
5. **Acquiring the best solution** – invocation of the method which returns the best obtained solution.

4.3.1 Modeling of planning problem

Modeling of planning problem consists of defining individual parts of the problem and creation of corresponding Java classes for problem fact, planning entity and variable, planning solution and other.

Problem fact Problem fact is a class which contains getters returning its properties. This class does not contain special OptaPlanner code (it can be ordinary Java class) and during planning it does not change. In case of N-Queen problem, rows and columns classes are problem facts.

Planning entity Planning entity is a class which change during planning. It has to be marked with `@PlanningEntity` annotation. Each planning entity has one or more planning variables. In case of N-Queen problem, Queen class is planning entity because its row position changes.

Planning variable Planning variable is a property of planning entity class with required getter and setter. In case of N-queen problem, row property is planning variable. It must be marked with `@PlanningVariable` annotation which contains `valueRangeProviderRefs` property which defines possible values of the planning variable.

Planning value and planning value ranges Planning value is a possible value of a planning variable. Usually, a planning value is a problem fact but it can be any object, for example a `Double`. Planning value range is a set of possible planning values of planning variable. This set can be a countable (for example row 1, 2, 3 or 4) or uncountable (for example any `Double` between 0.0 and 1.0). Value Range is marked with `@ValueRangeProvider` annotation which has property `id` pointing to `valueRangeProviderRefs` of `@PlanningVariable` property. `@ValueRangeProvider` annotation can be placed on two types of methods – on the Solution or on the planning entity. Usually, first type is used. `Collection` or `ValueRange` are two types which can be used as return type of the method.

Planning problem and planning solution Each planning problem has to be defined as a class which is used by the Solver to solve the problem. In the case of N-Queens problem, class must contains column, row and queen lists. The planning problem corresponds with unresolved planning solution. The solution must be described by a class that implements the `Solution` interface. This interface requires to implement `setScore`, `getScore` and `getProblemFacts` methods.

4.3.2 Solver configuration

The second part of OptaPlanner configuration is configuration of Solver. It is described by XML file and it can also be changed dynamically at runtime using `SolverConfig` API. Basically, it can be divided into three parts:

1. **Model definition** – consists of the name of a class that implements the `Solution` interface and a name of class that represents the planning entity.
2. **Score function definition** – consists of settings such as a type of score and a class which calculate the score (or DRL file with a rules that are used for calculation).
3. **Optimization algorithms definition** – contains settings of algorithms that are used to optimize the calculation for obtaining the best score of problem.

Thanks to the score calculation, all of the initialized solution can be evaluated by a score and these are three following ways how it can be implemented:

- **Easy Java score calculation** – The calculation is performed using single method of a class which implements the `EasyScoreCalculator` interface and this method should return score of solution. This simple way how to calculate score is slower and less scalable than other methods.
- **Incremental Java score calculation** – The calculation is performed using several specific methods of computation. This is a quicker approach, but more difficult for implementation. Class which defines this type of calculation must implements the `IncrementalScoreCalculator` interface.

- **Drools score calculation** – The calculation is performed using DRL rules. They are stored in `.drl` file. More information about rules can be found in [2, 17]. This approach is well optimizable but DRL language must be used.

4.3.3 Loading of problem data set

Last step before the start is the loading of problem data set into Solver. This is done by uploading planning entities and planning values to the appropriate collection in a class that implements the `Solution` interface. From this step, everything is ready for Solver activation.

In N-Queen example, all queens and all rows have to be initialized and uploaded to the appropriate collections of N-Queens solution class.

4.3.4 Activation of Solver and acquiring the best solution

Activation of Solver takes place simply by calling the `solve()` method of the Solver instance class with the parameter containing reference to an instance of a class that implements the `Solution` interface.

After the calculation by calling `getBestSolution()` method of the Solver instance, the best solution is returned. In the case of N-Queen problem, a solution where each queen has assigned one row should be obtained and if optimal solution is found, then no two queens are threaten to each other.

Termination Not all calculations terminate automatically and therefore it is sometimes necessary to add conditions which causes the termination. Subsequent options can stop calculation:

- **Time limit termination** – occurs after exceeding the time limit.
- **Best score termination** – terminates when a certain best score is reached.
- **Step count termination** – occurs after exceeding the limit of step count of calculation.
- **Combining of multiple terminations** – previous termination methods can also be combined.
- **Asynchronous termination from another thread** – can be used if it is necessary to terminate calculation differently than by automated methods.

Chapter 5

Porting of OptaPlanner to Android

This chapter deals with porting of OptaPlanner to Android. Portation is a modification of software for the purpose of usage on different platforms. OptaPlanner is designed for Java Standard Edition (SE) platform and for integration to Android it is necessary to compare both platforms and resolve potential problems.

In the first Section 5.1, requirements for OptaPlanner portation are introduced. Comparison of Java Application Programming Interface (API) and Android API are described in Section 5.2 and discovered JavaBeans problem is discussed in Section 5.3. Section 5.4 summarizes possible solutions of JavaBeans problem. Section 5.5 deals with design of the portation and the last Section 5.6 focuses on its implementation.

5.1 Requirements for OptaPlanner portation

In this section, three essential requirements for OptaPlanner portation are introduced. The first one focuses on automatic build process, the second one discusses the use of Drools library and the last requirement targets on OptaPlanner usability.

Automatic build process First of the portation requirements is ensuring of the automatic build process. When an application is built, it is good to make all the steps automatic. OptaPlanner libraries should be imported correctly and all the dependencies should be included or should be prepared by build script. Any other problems of the portation has to also be resolved in a way which does not required much effort from users of the scripts. These procedures have to be described and demonstrated on a model application.

Drools library As described in Chapter 4, one of the ways of computing function in OptaPlanner is by the Drools rules. Drools library is distributed as a part of OptaPlanner and also as a standalone project. Using this library, it is possible to write rule prescriptions for score calculation. Another way how to compute the score is standard Java calculation which does not require additional dependencies. Because Drools project is not optimized for mobile platforms and there is another way how to calculate the score, it should not be included in the portation of OptaPlanner on Android.

OptaPlanner usability In summary, it is necessary to prepare OptaPlanner for the Android platform to enable possibility of using tools for solving planning problems. Mobile

devices have limited computing capabilities and storage space in comparison to desktop computers. Ported OptaPlanner should consider these capabilities and adapt to them.

5.2 Java API packages comparison

This section compares two Java Application Programming Interfaces – Java 6 Standard Edition API and Android API. Depending on the comparison, OptaPlanner dependencies on Java SE API are identified on Android API and the consequent problems are highlighted.

Java SE API and Android API Android API is based on Apache Harmony Java 6 SE API [7] (the open-source version of Java SE). First column of Table 5.1 shows all the packages in Java 6 SE API. In second column of the table, it is described whether or not are the packages included in Android API. If both APIs are compared, it is certainly possible to say that Android API is not complete Java SE API. Out of 38 Java SE API packages, 20 packages are completely missing and 9 packages are incomplete in Android API. For example, packages for graphical user interface such as `java.swing` and `java.awt` are not included and provided for Android development because they are replaced with the Android graphical elements.

Android API and Optaplanner The third column in Table 5.1 describes whether the package is needed by OptaPlanner. This tool directly uses only 6 of the total number of 38 packages of Java SE API, namely: `java.beans`, `java.io`, `java.lang`, `java.math`, `java.net` and `java.util`. Three of the used packages are incomplete in Android API but only one of them affects OptaPlanner, speciffically `java.beans` package. Due to the fact that some of the `java.beans` classes are missing in Android API, direct OptaPlanner integration is impossible. Other missing packages do not affect OptaPlanner use on Android.

5.3 JavaBeans problem

JavaBeans package allows to reuse components written in the Java programming language. Primarily, its classes are used for creation of graphical user interface but they can also be used for introspection of methods, properties and events. More information about JavaBeans can be found in [4].

OptaPlanner is completely written in the Java language and one of its dependencies is `java.beans` package. As described in Section 5.2, this package is incomplete and classes which OptaPlanner requires are missing. In case of OptaPlanner use on Android, the compiler throws `ClassNotFoundException` and with this error, OptaPlanner project cannot be built. In this section, the possible solutions of this problem are presented and one of the solutions is selected for the needs of the portation.

The proposed solutions can be divided into two groups. First one contains solutions which do not interfere in OptaPlanner source code and they are described in Section 5.3.1, Section 5.3.2 and Section 5.3.3. Solutions which change the source code of OptaPlanner belong to the second group and they are described in Section 5.3.4 and Section 5.3.5.

5.3.1 Repacking of JavaBeans redistribution to Java core namespace

The first of the possibilities how to complete the missing `java.beans` package is use of a JavaBeans redistribution. These libraries are specially designed for the Android platform

Java 6 SE Package	Included in Android API	Needed by OptaPlanner
java.applet	No – missing completely	No
java.awt	Yes – incomplete	No
java.beans	Yes – incomplete	Yes
java.io	Yes – complete	Yes
java.lang	Yes – incomplete	Yes
java.math	Yes – complete	Yes
java.net	Yes – complete	Yes
java.nio	Yes – complete	No
java.rmi	No – missing completely	No
java.security	Yes – incomplete	No
java.sql	Yes – complete	No
java.text	Yes – complete	No
java.util	Yes – incomplete	Yes
javax.accessibility	No – missing completely	No
javax.activation	No – missing completely	No
javax.activity	No – missing completely	No
javax.annotation	No – missing completely	No
javax.crypto	Yes – complete	No
javax.imageio	No – missing completely	No
javax.jws	No – missing completely	No
javax.lang	No – missing completely	No
javax.management	No – missing completely	No
javax.naming	No – missing completely	No
javax.net	Yes – complete	No
javax.print	No – missing completely	No
javax.rmi	No – missing completely	No
javax.script	No – missing completely	No
javax.security	Yes – incomplete	No
javax.sound	No – missing completely	No
javax.sql	Yes – incomplete	No
javax.swing	No – missing completely	No
javax.tools	No – missing completely	No
javax.transaction	No – missing completely	No
javax.xml	Yes – incomplete	No
org.ietf.jgss	No – missing completely	No
org.omg	No – missing completely	No
org.w3c.dom	Yes – incomplete	No
org.xml.sax	Yes – complete	No

Table 5.1: Java 6 SE API packages in Android API and OptaPlanner dependencies.

to support JavaBeans or other missing packages. If OptaPlanner source code should stay the same, it is necessary to repackage these libraries to Java core namespace. Java core namespace is an identification of Java API classes which belongs to `java.*` or `javax.*` namespace. In the following paragraphs, two redistribution and the Jar Jar Links tool

for repacking Java Archive (JAR) files are introduced. Last paragraph describes problem which is appearing during usage of classes from Java core namespace on Android.

OpenBeans OpenBeans project [10] is a redistribution of `java.beans` package based on the Apache Harmony project. It was created because of missing JavaBeans on the Android platform. Used namespace of this package is `com.googlecode.openbeans`. OpenBeans is an open-source project and it is distributed as JAR file that can be included into a Java or an Android project.

Mad Robot A similar project to OpenBeans is called Mad Robot [12]. As well as OpenBeans, it contains redistribution of `java.beans` package in `com.madrobot.beans` namespace but it also includes some additional packages for database, graphics or geometry manipulation. This project is distributed as Maven dependency.

Jar Jar Links Jar Jar Links [11] is a utility for repackaging Java libraries. It enables to repack Java classes from one namespace to another. For proper use, it is necessary to define rules which describe way how Java classes should be repacked and how the classes have to be placed in a JAR file. Jar Jar tool uses this file to create new JAR with repacked classes.

Core library problem Compilation of an Android project that contains a class from namespace `java.*` or `javax.*` crashes during the translation which is highlighted by message about using the classes from Java core namespace. It is a protection against unauthorized use of the namespace. This can be avoided by using `--core-library` flag. Adding the flag allows translation of the application. One of options how to use the flag is modifying the `dx` script in Android SDK (Software Development Kit). Listing 5.1 shows the flag added to the last line of the `dx` file.

```
exec java $javaOpts -jar "$jarpath" --core-library "$@"
```

Listing 5.1: Core library flag in the last line of `dx` script.

5.3.2 Use of OpenJDK distribution source code

This solution is based on the use of available source code of OpenJDK Java SE [5] which is an open-source distribution of Java SE. By adding sources to an Android project, it is possible to get the necessary libraries. The advantage of this solution is that the dependency failures are seen in translation and not when the application runs. This enables to choose only the required classes. However, this adjustment is not trivial. It needs to be done by a special tool that removes unused dependencies or it must be done manually.

5.3.3 Use of pruned rt.jar from OpenJDK distribution

The last option without interference to the source code is use of the `rt.jar` file which is part of the Java SE libraries. This package contains JavaBeans compiled classes and other parts of Java SE. Due to its size, it is not well suited for an Android applications and it also includes libraries that are already contained in Android API and can causes collisions. Therefore, it has to be pruned. The advantage of pruning is that there is no need to worry

about dependencies that are not needed for OptaPlanner tool because these files are not again compiled. On the other hand, it may happen that an application hits some missing required dependencies during runtime and the application crashes.

5.3.4 Use of OpenBeans in OptaPlanner project

This is the first solution which intervenes to the OptaPlanner source code and it consists of replacing all `java.beans` dependencies for the `com.googlecode.openbeans` by rewriting all imports and by addition of `OpenBeans.jar` archive to the OptaPlanner core project. This causes redirection of all dependencies to OpenBeans. The disadvantage of this solution is the intervention in the OptaPlanner source code. In terms of Android application developers, it is needed to create anew fork (or branch) of OptaPlanner and maintenance of such copy is complicated and complex.

5.3.5 Removing and replacing JavaBeans from OptaPlanner

Last option to solve the JavaBeans problem is its elimination from the source code and its replacement by another technology. This is the biggest intervention to OptaPlanner code of all the offered solutions and it is also the major disadvantage. As in the previous solution, it is necessary to create a new fork of OptaPlanner and take care of its maintenance.

5.4 Summary of approaches

Table 5.2 shows summary of the JavaBeans problem solutions. The first column contains solution name. In the second column, licenses which should be respected when using concrete approach are placed. The need for modification of OptaPlanner source code is marked in the third column. Assumed solution level of difficulty is placed in the fourth column and advantages and disadvantages of each approach are described in the last column.

License and avoiding the modification of OptaPlanner are the essential requirements of the selection. License has to permit commercial and private use and modification of the code. Furthermore, the easier solution is better because then each developer has an opportunity to easily use OptaPlanner on Android without making any special modifications.

The best solution of JavaBeans problem seems to be repacking of the OpenBeans redistribution of JavaBean to Java core namespace. In this approach, suitable Apache license must be respected. This license is free software license and it allows easy usage of the code. It is not necessary to modify the OptaPlaner code and generally, this approach requires less effort from the programmer.

5.5 Design of the portation

This section describes design of the portation based on the requirements from Section 5.1. Major task is to make the entire process automatic and show how OptaPlanner can be ported into Android project of a developer tool.

Developer tools As described in Chapter 3, the official environment for development of Android applications is called Android studio which brings new build system which is based on Gradle language. Android studio is selected as primary developer tool for portation needs and Gradle laguage is chosen for automatization of all the required processes.

Approach name	License	OptaPlanner modification	Level of difficulty	Advantages and disadvantages
Repacking OpenBeans redistribution to Java core namespace	Apache License 2.0	No	Easy	+ standalone jar file, no problems with dependencies
Repacking of Mad Robot redistribution to Java core namespace	LGPL 2.1	No	Easy	+ same as in previous case
Use the OpenJDK distribution source code	GPL 2.0	No	Medium	+ dependency failure occurs in translation, source code control - difficult adjustment which can cause problems with dependencies
Use of pruned rt.jar from OpenJDK distribution	GPL 2.0	No	Hard	+ standalone jar file - difficult adjustment which can cause problems with dependencies, incosistent jar, prunning
Use of OpenBeans in OptaPlanner project	Apache License 2.0	Yes	Easy	+ easy adjustment - need of modification of OptaPlanner source code and the subsequent maintenance of OptaPlanner fork
Removing and replacing JavaBeans from OptaPlanner	–	Yes	Medium	- same as in previous case

Table 5.2: Summary of the JavaBeans problem solutions.

Importing of OptaPlanner libraries OptaPlanner is distributed as a Maven dependency or it is packed in JAR files. Because one of the requirement is automatic build process, first type of distribution is selected. Gradle language supports Maven dependencies and it allows complete automation of importing process. Because OptaPlanner is daily developed, its last snapshot version should be always used to secure that the portation is compatible with Android for the future releases of OptaPlanner.

Exclusion of unnecessary libraries As written in Section 5.1, Drools libraries should not be included in portation. It can be assumed that also another libraries can be excluded because they can be already part of Android API. These exclusions has to be performed by Gradle language to secure the process automation. It also helps to reduce size of the potential applications.

Completion of missing libraries Section 5.3 describes JavaBeans problem and one of the suggested possible solutions was selected in Section 5.4. Task of the chosen solution is to repack OpenBeans redistribution to Java core namespace and following steps should be performed by Gradle language to automate the entire process:

- Download JarJar tool for repacking libraries.
- Download OpenBeans package which is in `com.googlecode.beans` namespace.
- Run JarJar tools and repack OpenBeans package to Java core namespace (`java.beans`).
- Clean temporary files and move final JAR into proper directory.
- Add `--core-library` flag to allow use of the Java core classes.

5.6 Implementation of the portation

In this section, implementation of the portation is described. The first part focuses on the two main issues and the second part on the Gradle scripts which implement the portation itself are introduced.

5.6.1 JavaBeans and Core library flag

Two main issues need be to solved before it is possible to use OptaPlanner on Android. First one is completion of missing libraries to a project and second one is adding the `--core-library` flag to allow use of the Java core classes.

Completion of JavaBeans library Completion of JavaBeans library is implemented by Gradle scripts described in Section 5.6.2. These scripts can be used in any Android Gradle project and they enable to automatically download required tools and packages from the Internet, repack OpenBeans library and create JavaBeans JAR file which is added to the project.

Core library flag If some classes from Java core namespace are present in an Android project, its compilation fails. Proposed solution where `--core-library` flag is added to `dx` script in Android SDK directory as shown in Listing 5.1 does not work from version 1.1.0 of Android Gradle build tools. The build tools are changed and they no longer use the `dx` script. Therefore, the flag must be directly embedded in settings of Android plugin in `build.gradle` file as shown in Listing 5.2.

```
project.tasks.withType(com.android.build.gradle.tasks.Dex) {
    additionalParameters=['--core-library']
}
```

Listing 5.2: Addition of Core library flag in `build.gradle` script.

5.6.2 Gradle scripts

The entire process of the portation is automatized by Gradle scripts. Gradle is default building tool for Android studio which is currently a primary application for creating Android projects.

This section discusses how OptaPlanner can be added to an Android project and describes tasks which perform operations required to complete missing libraries described in Section 5.5. All the described tasks are used in application described in Section 6.

OptaPlanner dependency As in any other project which does not use only standard Java classes, it is necessary to include third party libraries. Gradle can use Maven dependencies and this is one of the ways how OptaPlanner is distributed. Libraries can also be added directly as JAR files but it would be worse for automatization. As mentioned in Section 5.1, Drools library should not be included and also `xmlpull` library has to be excluded because it is already contained in Android API. Listing 5.3 shows OptaPlanner dependency with excluded libraries.

```
compile ('org.optaplanner:optaplanner-core:6.3.0-SNAPSHOT') {
    exclude group: 'xmlpull'
    exclude group: 'org.drools'
}
```

Listing 5.3: OptaPlanner Maven dependency in Gradle build script.

Downloading and removing tasks There are two downloading tasks `downloadJarJar` and `dowloadOpenBeans`. First task downloads repacking tool Jar Jar links and second one downloads OpenBeans package in JAR format from the Internet. The tasks for removing temporary files after their use are namely: `deleteJarJar`, `deleteOpenBeans`, `deleteRule` and `deleteJavaBeans`.

Task for creating of repacking rule file Jar Jar tool needs to know how to repack `OpenBeans.jar`. For this purpose, there is `createRuleFile` task which creates file with rule as shown in Listing 5.4. This rule tells that all OpenBeans namespaces should be replaced with JavaBeans namespace.

```
rule com.googlecode.openbeans.** java.beans.@1
```

Listing 5.4: Jar Jar Links rule for repacking OpenBeans to Java core namespace.

Task for repacking OpenBeans Last task for manipulation with OpenBeans package is `repackOpenBeans` task. This task launches all the previous tasks and also executes command which creates JAR file with repacked OpenBeans to Java core name space. The new file is placed into project `/lib` directory. This command is shown in Listings 5.5. Finally, removing tasks are activated and temporary files are deleted.

```
java -jar jarjar.jar process rule.txt openbeans.jar javabeans.jar
```

Listing 5.5: Command for repacking `openbeans.jar` file.

Task for adding Core library flag This task is originally created for modification of `dx` script in build tools directory in Android SDK. It finds `dx` file and adds `--core-library` flag to the last line of the file. In latest versions of Android Gradle build plugin, this approach does not work and it is replaced as described above in this section.

Chapter 6

Vehicle Routing Problem application

One of the goals of this thesis is creating of a simple application which demonstrates functionality of OptaPlanner tool on Android system. Previous Chapter 5 shows how to port the tool to the mobile platform. In this chapter, OptaPlanner tool is used to creating the Vehicle Routing Problem application.

First Section 6.1 introduces application requirements. Application design is described in the second Section 6.2. Implementation itself is divided into two parts. The first part which is described in Section 6.3 present inner structure of the application and the second part deals with graphical user interface and its layout.

6.1 Requirements for Android application

This section deals with requirements for Android application. First part introduces features of the application and the second part discusses support for different versions of the Android operating system.

6.1.1 Application features

The following paragraphs describe essential requirements and features of created application. They focus on inner structure, graphical user interface and the licence under which the application is written.

Vehicle Routing application Standard OptaPlanner distribution [19] contains demonstration examples. One of the examples is Vehicle Routing application. It is often used for presentation of OptaPlanner and it is one of the real world examples and thus it is also a good choice for demonstration on Android. Created aplication should present the Vehicle Routing Problem in a similar way as the original OptaPlanner application.

Vehicle Routing model The source code of the original application already contains OptaPlanner Vehicle Routing Problem model which should be included in this application. Furthermore, it contains some tools for importing specific `.vrp` files. The model specifies classes which are required for OptaPlanner tool.

Graphical user interface Graphical user interface cannot be ported because application is written by Awt and Swing libraries which are not included in Android API as described in Section 5.2. Therefore, new application graphical user interface should be created and adjusted to fulfill aspects of Android application development.

Application settings Application without any settings is too static and uninteresting for the users. Therefore, they should at least be able to choose problem solving algorithm. Another option could be setting of calculation time limit. Thanks to that, process can be terminated earlier.

File opening The original Vehicle Routing application contains `.vrp` example files which contains tasks of problems. These files should be compatible with created application and some of them should be included. It should be possible to open these files and display them in a similar way as in the original application.

Solution displaying The application must be able to display unsolved solution on the application screen. Furthermore, it should be possible to display new best solution everytime when it is found and at the end of the process, last best solution should remain on the screen. The application should also be able to display actual statistic information about solved problem.

License The entire application should be distributed as an open-source software and it should be written under Apache License 2.0. Source code must be publicly available on the Internet for guidance of other people in their own OptaPlanner projects.

6.1.2 Android devices support

Before the development starts, it is good to clarify which version of Android will be supported by application. Every version of Android comes with new API and new functionality. Biggest changes comes when the first number of version is changed. Actual distributions of Android versions on devices can be seen in the last column of Table 3.1.

Versions 2.x.x are on decline. Currently, the most used versions are with 4.x designation and distribution of the newest 5.x versions grows. Therefore, it is decided that application should support Android from version 4.0 (API 14). The version decides which resource can be used for application design and development and how application should be tested.

6.2 Application design

One of the critical points of creating an application is design. It can be divided into two parts. The first one is design of an inner structure and it is presented in Section 6.2.1. The second one is design of graphical user interface and application component layout which is described in Section 6.2.2.

6.2.1 Application features

In Section 6.1.1, inner structure requirements of the application are described. According to them, the application features design is written and introduced in following paragraphs.

Application settings Designed application supports several setting options. It is possible to select one of three algorithms: First fit decreasing with Late acceptance, Branch and bound and Brute force. Simultaneously, it is required to select time limit of calculation in seconds. Calculation stops after time limit is reached or it is possible to stop it earlier by stop button. Furthermore, Vehicle Routing example can be chosen. Application contains list of these files and after user selects one of them everything should be prepared to calculation start. The setting should be placed on the first screen of the application.

VRP files Mentioned list of example files consists from files used in original OptaPlanner application. These files are named the same way and user can compare results between both applications. Because default Android system does not contain file browser and user probably does not have his own `.vrp` files, application does not support opening files from device storage. List of files should be placed on the second screen of the application.

Porting of Vehicle Routing Problem model In Section 6.1.1, it is described that original application already contains Vehicle Routing model for OptaPlanner tool. These classes have to be embedded into application and they have to be used for calculation of the problem. Furthermore, they should be used for displaying of current solution.

Problem solving Graphical user interface cannot wait until calculation is finished and therefore these two parts must be separated from each other. While solving is in progress, user controls the application and also can use some of its parts. When screen is turned or application is hidden in background, solving process has to be still active and not terminated.

Solution displaying New best solution should be displayed every time when it is found. For that purpose, application contains third screen especially for displaying founded solution which should be similar to original application. Lines represents roads, vehicles and depots have their own icons. Customers differs from the original for saving screen space. Instead of points with numbers, customers should be represented as circles with inner number describing its demand. Time window circles should be displayed in a similar way as the original but not separated from customer.

Solution data Every solution has data which cannot be displayed graphically or they are too important and have to be be displayed separately. Hence, it is designed that score of solution and actual load of vehicles should be drawn on side menu which is described in Section 6.2.2.

6.2.2 Design of screens

This section introduces design of application screens and describes component layouts. The overall concept and links between screens are shown in Figure 6.1.

Application top bar Top bar is placed on the top of the each screen as shown in Figure 6.1. It contains name of the application and quick function buttons. These buttons can start the calculation or call the informational dialogs. Buttons are not visible all the time but only on the screens when it is necessary.

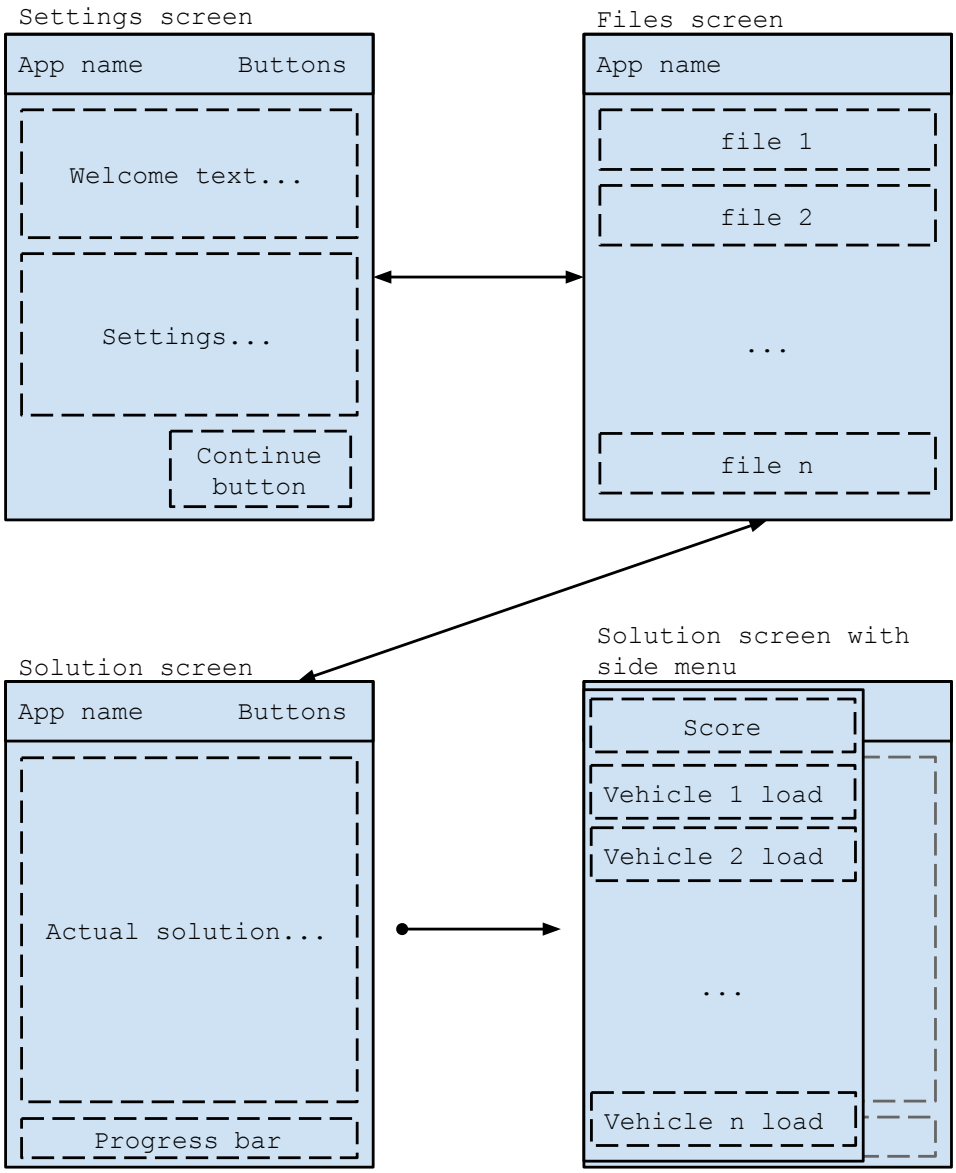


Figure 6.1: Design of screens and links among them.

Settings screen The first screen of Figure 6.1 is main screen of the application. It consists of top bar, welcome text and part with setting elements where calculation options can be set. Last element on the screen is button for switching to the next screen where one of the VRP files can be selected.

Screen with VRP files list The second screen in Figure 6.1 contains only list of VRP files and top bar. Top bar does not contains any buttons on this screen because there is no need for them. The list contains all of the included VRP files and after click on one of them, it is switched to the last screen – Solution screen.

Solution screen The most important screen of the application is the last screen. Solution and its gradual progress is displayed on this screen. On the top bar, button for start of calculation is included and under the actual solution representation, progress bar for displaying actual time is placed. When user swipe with finger from left to right on the screen, side menu with actual solution statistic should be displayed.

Side menu with statistics The last section of Figure 6.1 is side menu which is displayed on right side on the solution screen. It contains statistic data of actual solution. First item on the menu is solution score and next items represents every vehicle of the problem and its current load and capacity. Menu can be closed when user clicks somewhere outside of the menu.

Informational dialogs Application design contains two informational dialogs for better understanding of the application content. First one contains information about application itself. Second one consists of legend which describes all displayed components of Vehicle Routing Problem on the screen.

Material design One of the new features which Android version 5 brings is material design. It is very sophisticated study that shows how to handle elements, layouts, colors and others. Although this feature is not fully backward compatible, it is partially possible to bring this design to earlier devices with older versions of Android. It is designed that application should use material design as much as possible.

6.3 Application implementation

In this section, implementation of Vehicle Routing Problem application is described. The first part introduces application structure and its important components. Second part focuses on porting of Vehicle Routing Problem and its model from the original OptaPlanner application. Graphical user interface is presented in the next Section 6.4.

6.3.1 Application structure

Application consists of one activity and three fragments as shown in Figure 6.2. Activity is represented by `MainActivity` class in the application code and it contains Action bar and space where fragments are placed. Every time when action of fragment change is invoked, the space is replaced by new fragment.

Settings elements are placed on the first fragment which is defined by `MainFragment` class. `VrpFileListFragment` class represent the second fragment containing list of VRP files. The last fragment is defined by `VrpFragment` class and it is used for displaying current solution.

In the following paragraphs, important components of application are described. Especially, the components which are related to the background processes. Graphical components and their layout are described in Section 6.4.

List of files List of VRP example files is placed on the second fragment. This list is implemented by `RecyclerView` component which simplifies displaying of data to the list and provides basic patterns of behavior.

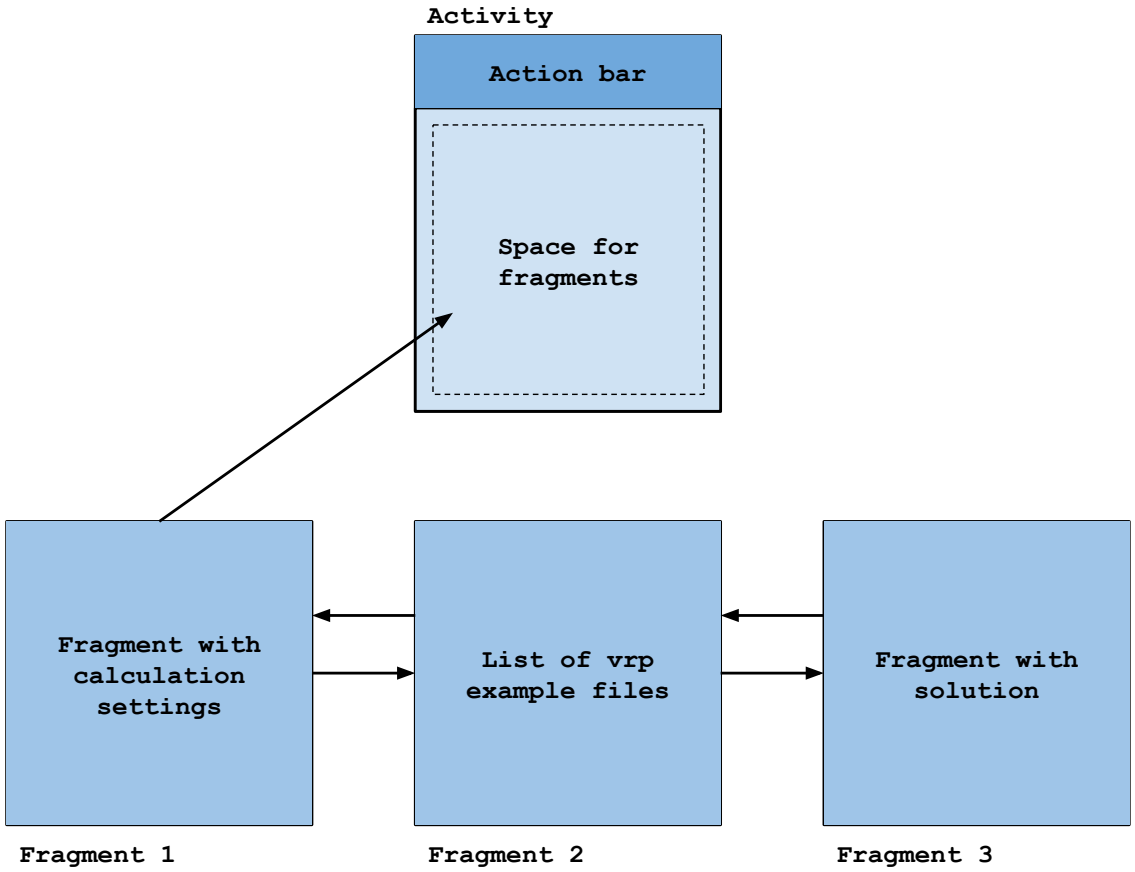


Figure 6.2: Activity and fragments in application.

Solver asynchronous process After the button for calculation start is pressed, asynchronous process is created. This process sets, builds and activates Solver with required parameters. Listener is added to Solver to publish process every time when new best solution is found. The process is represented by `VrpSolverTask` class which extends `AsyncTask`. `AsyncTask` class enables changes of graphical user interface, perform the background operations and publishes results.

Solution painter Solution painter is a component which draws a solutions on the screen. It is represented by `VrpPainter` class which is modified `VehicleRoutingSolutionPainter` class from the original OptaPlanner project. Because Android does not support Awt and Swing Java graphic libraries, `VehicleRoutingSolutionPainter` was rewritten to use Android methods for drawing on the screen. Solution painter draws two types of solution:

1. **Unsolved solution** – painted when a file is selected from the list in the second fragment.
2. **New best solution** – painted every time after Solver is activated and new best solution is found.

6.3.2 Porting of Vehicle Routing Problem

Original Vehicle Routing application contains Vehicle Routing Problem model for OptaPlanner tool. These files are taken and modified to fit in the created Android application. Following paragraphs present these files and show the way how they are used.

Vehicle Routing Problem model Without model of the problem, application cannot work. Vehicle Routing Problem is defined by `VehicleRoutingSolution` class which implements `Solution` interface. This class contains all information about solved problem (list of all customers, depots and vehicles) and it is used together with solver configuration by Solver to calculation of the problem. `Customer` class is marked as planned entity and contains `Standstill` planning variable. More detailed information about Vehicle Routing Problem definition can be found in OptaPlanner documentation [20].

Solver configurations In this application, it is possible to use three algorithms and set time limit of calculation. These configurations include link to problem definition and link to score calculator and they are stored in `.xml` file which are used for building the Solver. For each algorithm, there is one XML file and it is applied according to the choice in the application. Time limit is additionally set after Solver is created.

Score calculator Every solution has its own score and this score must be calculated by one of the three methods described in Section 4.3.2. For score calculation in this application, `VehicleRoutingEasyScoreCalculator` class which implements `EasyScoreCalculator` is used. This class calculates hard and soft score of solution. Hard score is computed as a load of vehicles above their capacity and soft score is calculated as negative total vehicle distance. In case of time window variant, delay against due time of arrival is added to the hard score.

VRP example files Example `.vrp` files are used as problem datasets. These files are taken from the original OptaPlanner Vehicle Routing Problem application. They contain information about number and capacity of vehicles, position and demands of customers and position of depot. This application includes 36 example files in total.

Vehicle Routing importer Example files are stored in specific `.vrp` text format and have to be loaded into classes that describe Vehicle Routing problem. For this purpose, `VehicleRoutingImporter` class is imported and used from the original OptaPlanner application.

6.4 Graphical user interface

This section presents graphical user interface implementation of the application as designed in Section 6.2.2. The first part of this section describes application screens and the second part introduces three important components.

6.4.1 Application screens

Every application consists of Fragments or Activities which are collectively called screens. Using controls, it is possible to move from one screen to another or to change its appearance

or behavior. This application is composed from three screens. These screens can be seen in Figure 6.3. The third screen is displayed with unsolved problem and with ongoing solution process.

Main screen Main screen is displayed after the application starts. It consists from Action bar, welcome text, setting elements and button to continue to another screen. Action bar contains application name and buttons for displaying legend dialog and application information dialog. Welcome text provides some basic instruction for the users. Two controls are present for calculation options of the problem. It is possible to set time limit in seconds using Number picker and Spinner allows to select one of the three supported algorithms. Last element on this screen is Open file button which opens screen with list of VRP files.

Screen with VRP files list After Open file button on the main screen is pressed, screen with VRP files is displayed. It also contains action bar but it is very limited because no controls are required on this screen. Rest of area is filled with list of VRP files from original OptaPlanner application [19]. After the selected file is pressed, it is switched to last screen and the problem with its solutions is displayed.

Solution screen This screen is used to display unsolved, ongoing and solved solutions. It contains Action bar with all of the control items as shown in Figure 6.3. Compared to the main screen, Action bar has an additionally button for displaying Navigation drawer and button for start and end of the solution process. On the bottom of the screen, Progress bar is placed. This component is used for displaying time which approximately remains. Rest of the screen is filled with component which draws current solution. At the beginning, the component draws unsolved solution and after the start button is pressed, it always draws the best solution after it is found. Individual elements of this component are:

- **Circle with a number** – customer with his demand.
- **Building image** – depot from where vehicles depart.
- **Car image** – vehicle with its color.
- **Solid line** – vehicle road to a customer.
- **Dashed line** – vehicle road to a depot.
- **Sector on a circle** – time windows for vehicle arrival.
- **Line on a circle** – vehicle arrival time.

6.4.2 Application components

This section introduces and describes three important components of the application: Action bar, Navigation drawer and Dialogs.

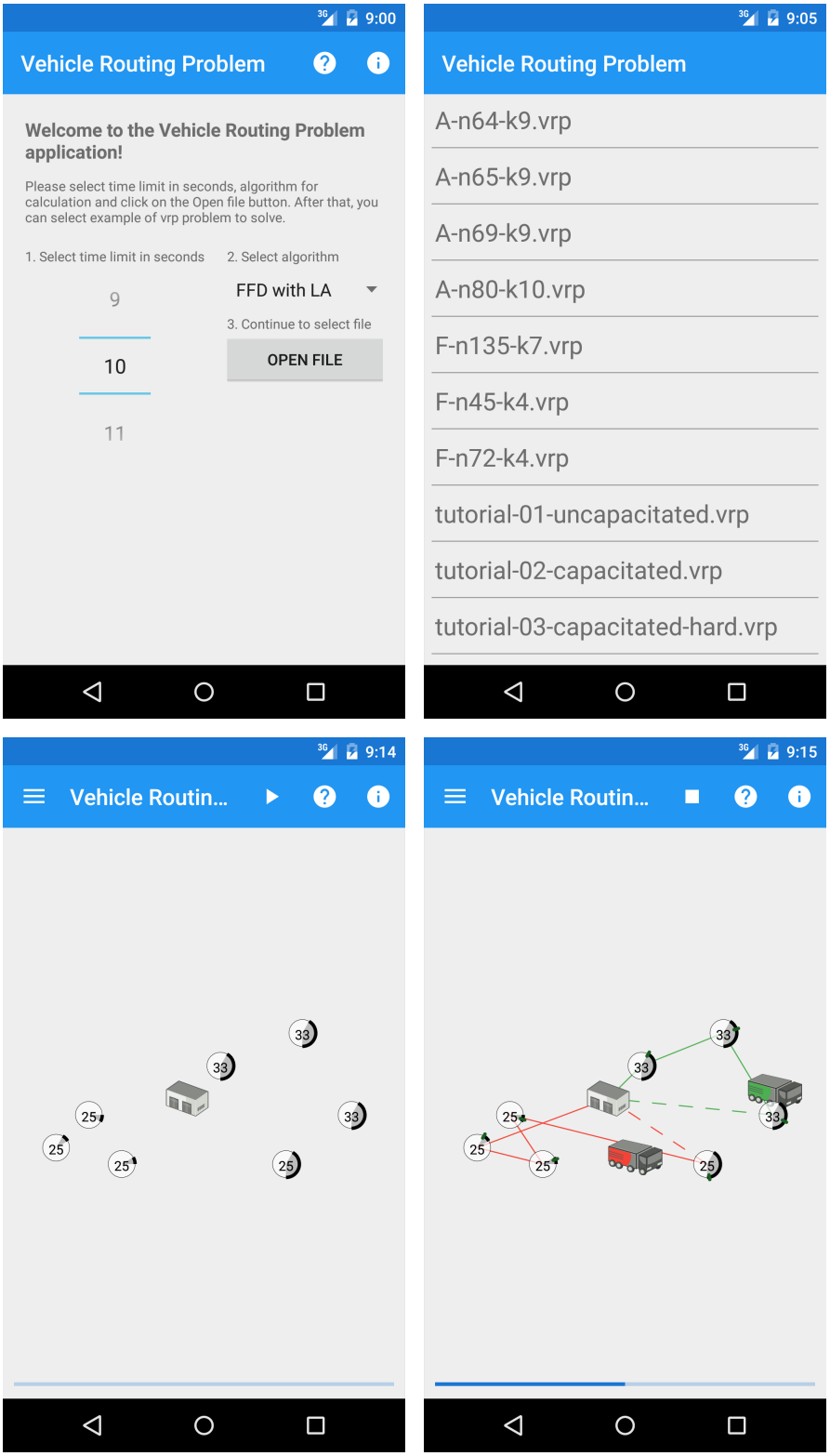


Figure 6.3: Application screens – main screen, list of VRP files, screen with unsolved problem and ongoing solution process.

Action bar

Action bar is a panel on top of the screen that provides basic user action and information about user navigation. It always contains application name, optionally action buttons for quick invocation of application functionality and overflow button on the right side for displaying the other applications options.

Action bar is displayed on every screen of this application but it changes depending on required functions on actual screen. Figure 6.3 shows action bars of each screen.

Navigation drawer

Navigation drawer is a panel that displays application navigation on the left edge of the screen. By default, it is hidden and it could be displayed by touching the left icon on the action bar. Also, it could be displayed when a user swipes with a finger from the left edge of the screen to the right. Opposite procedure makes navigation drawer invisible.

This application uses navigation drawer for displaying important statistic data. Figure 6.4 displays visible panel on the left side of the application. The first item on the panel shows hard and soft score of displayed solution. Second item holds total distance traveled. Other items are linked to vehicles of the problem. Each of them has its own parameters – color, name and capacity. These three items are static and do not change during the calculation. Last parameter is actual load of the vehicle.

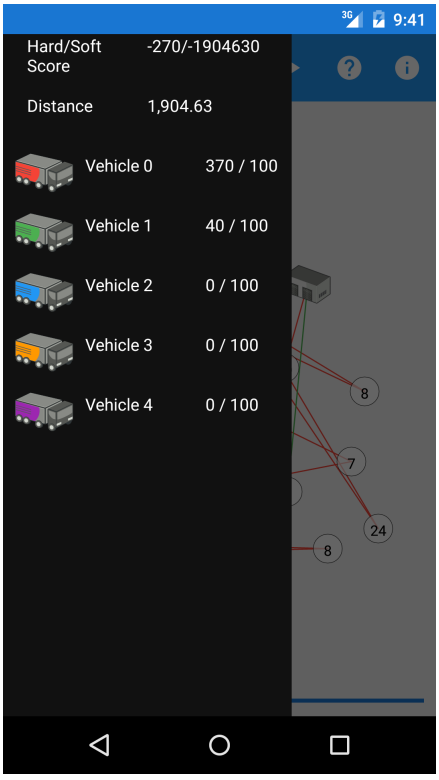


Figure 6.4: Navigation drawer with actual data.

Dialogs

Dialogs are small windows which display some significant information or they are used for user interaction with decisions that define further actions. Dialogs are always located above all other parts of the application.

Figure 6.5 shows all three dialogs used in the application. The first one and the second one can be retrieved directly from the action bar by clicking on the icons with question mark or informative icon. The first dialog contains application legend for understanding what is displayed on the screen and the second dialog briefly describes the application. The last dialog is displayed only when calculation runs and user clicks on the back button. Dialog then asks the user if he wants to end the ongoing calculation.

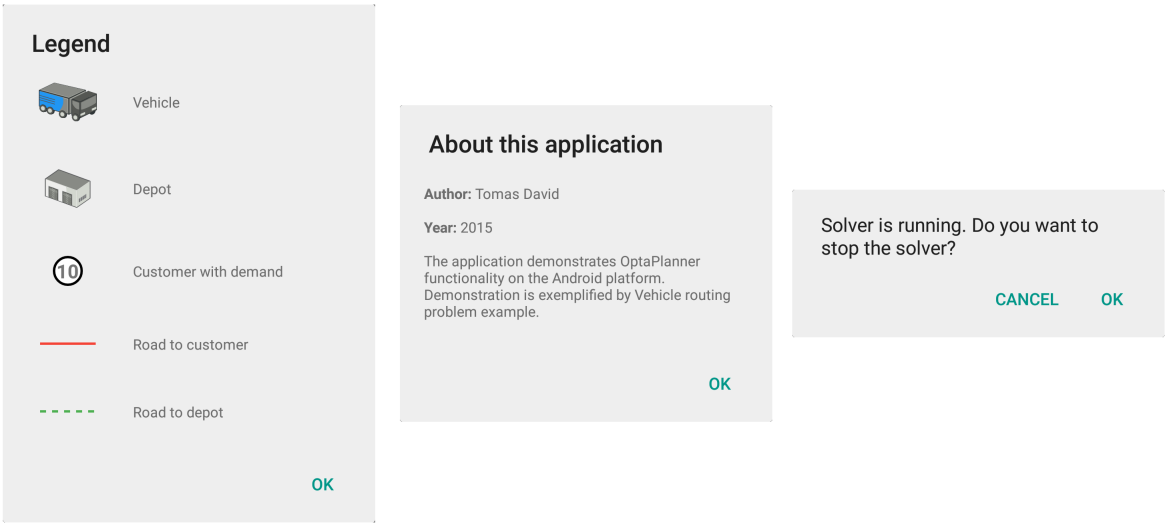


Figure 6.5: Screenshots of dialogs used in the application.

Chapter 7

Testing and future work

Testing of applications is very important part of the development. During this phase, a large number of bugs is often detected and critical problems can be also discovered. Measurement is as well counted to the testing because it can reveal performance problems.

Section 7.1 is focused on testing devices and describes performed comparative measurements. Section 7.2 describes future work which shows direction of the continuation of the project and points out to the important parts which can be implemented or improved.

7.1 Testing

This section introduces testing devices and describes performed comparative measurements. It focuses on comparison between mobile devices and desktop computer which differ especially in their performance.

Devices

The application is tested on several devices during and also after development. Table 7.1 shows testing devices and their important parameters. Some issues which are associated to them are described in following paragraphs.

First of the parameters is Android version which specifies supported parts of Android features. Most of the problems are related with unsupported functions and they are usually detected by compiler. However, from time to time some problems can appear and therefore it is better to test application on various versions of Android system.

Next parameters are display size and display resolution which together define density of the screens points. Every device has different display density and component layout must be well designed to fit to many devices screens. Another problem appears after the screen rotation. Display sides are exchanged and components layout must be adapted.

Last parameters are number of CPU cores, CPU and RAM. They define device speed and computing capabilities. Because this application calculates with many values, some tests and measurements which compare these devices are performed.

The application is tested on the five devices in total. Four of them are mobile phones, one is tablet and the last one is Android emulator. Emulator is a virtual mobile device that runs on a desktop computer. Parameters of emulator can be selected as necessary except of CPU which is inherited from host computer and adjusted according to operating system capabilities.

Device	Android version	Display size [inches]	Display resolution [pixels]	CPU cores	CPU [GHz]	RAM [GB]
LG Nexus 5	5.1.0	4.95	1080 x 1920	4	2.3 GHz	2
Asus Nexus 7 2013	5.1.0	7.0	1200 x 1920	4	1.5 GHz	2
Samsung Galaxy Xcover	4.1.2	4.0	480 x 800	2	1 GHz	1
Sony Xperia active	4.0.4	3.0	320 x 480	1	1 GHz	0.5
Emulator	4.4.2	4.5	720 x 1280	–	–	1.5

Table 7.1: Configuration of testing devices.

Comparative measurements

After testing and debugging the application, two comparative measurements are performed on the devices mentioned in Table 7.1 and on one desktop computer with parameters described in Table 7.2. On the desktop computer, simple OptaPlanner project calculating the same Vehicle Routing problems has been created. On the mobile testing devices, graphical part is excluded and only calculation itself is measured. The goal of these measurements is to compare capabilities of Android devices with a standard desktop computer and it should verify if Android devices are capable to use OptaPlanner.

Parameter	Value
Device	Notebook Lenovo ThinkPad T430s
CPU	Intel Core i7-3520M 2.90GHz
RAM	16 GB
OS	Fedora 21 64b

Table 7.2: Configuration of testing computer.

For both tests, three testing VRP files are used. They differ in the number of customers, the number of vehicles and the capacity of vehicles. Parameters of the three used files are shown in Table 7.3. The same files are also used in the application on the desktop computer.

VRP file	Number of customers	Number of vehicles	Vehicle capacity
A-n32-k5.vrp	31	5	100
A-n64-k9.vrp	63	9	100
F-n135-k7.vrp	134	7	2210

Table 7.3: Testing files.

First test deals with a time of the first found solution. Each test case is measured five times and the arithmetic mean value (μ) and standard deviation (σ) was calculated. The values are written in Table 7.4. As can be seen, differences between the desktop computer and the mobile devices are quite substantial. Thus, the more complex problems the difference between them rapidly grows. It can be also observed that more powerful mobile devices (Nexus 5 and 7) surpass older devices (Samsung Xcover and Sony Xperia) with worse CPU and performance of emulator is approximately equal to the most powerful mobile device (Nexus 5).

Device	A-n32-k5.vrp		A-n64-k9.vrp		F-n135-k7.vrp	
	μ	σ	μ	σ	μ	σ
LG Nexus 5	0,162	0,034	0,561	0,031	2,760	0,063
Asus Nexus 7 2013	0,228	0,027	0,795	0,023	4,132	0,070
Samsung Galaxy Xcover	0,411	0,058	1,434	0,012	8,219	0,217
Sony Xperia active	0,755	0,136	2,705	0,027	16,334	0,062
Emulator	0,119	0,021	0,386	0,056	1,647	0,015
Desktop computer	0,075	0,006	0,130	0,013	0,284	0,017

Table 7.4: Time of first found solution in seconds (less is better).

Second test is displayed in Table 7.5 and it is based on 10 seconds time limit. After the time limit the best soft score was saved and of five measurements the largest was selected. Soft score shows a distance of the all vehicles which is traveled between customers and depot as a negative value. A higher value means a smaller distance. Quite substantial difference is appears again between mobile devices and desktop computer. Although the Nexus devices have different parameters, they reached same results in two cases. The oldest device (Sony Xperia active) did not reach any result in the appointed time limit for the last case.

Device	A-n32-k5.vrp	A-n64-k9.vrp	F-n135-k7.vrp
LG Nexus 5	-857311	-1597400	-1411795
Asus Nexus 7 2013	-857311	-1624700	-1411795
Samsung Galaxy Xcover	-879018	-1631923	-1420843
Sony Xperia active	-894553	-1633708	–
Emulator	-855010	-1597400	-1411448
Desktop computer	-787082	-1424148	-1292057

Table 7.5: The best soft score after 10 seconds time limit (larger is better).

Although there is a significant difference between the desktop computer and Android phones or tablet, these mobile devices are still sufficiently fast to use OptaPlanner tool and solve such kind of problem.

From testing and measuring, it can be said that portation of OptaPlanner is successful. Standard Java score calculation of OptaPlanner is used and the calculation process is able to achieve satisfying times on the real mobile devices. Also, there were no problems recorded with OptaPlanner tool and its use.

7.2 Future work

This section describes future work which comes after this project. There are already plans how to improve OptaPlanner integration to Android system and in addition, there are two main projects which will be primarily processed.

OptaPlanner game Because OptaPlanner works on Android, there are many ways how to use it. One of them is to create a simple game where user goal is to defeat the Opta-

Planner by finding shortest way for vehicles. By clicking on the customers user can create road for vehicle to the depot. Meanwhile, time is measured and when user finishes his task, OptaPlanner do the same and results are compared. This is the preliminary proposal of the game that follows this work.

Drools As noted in Chapter 5, Drools package is not included in the poration due to its size and complexity. There are plans to port Drools tool to Android separately. If the portation is successful, it will be deployed and tested together with OptaPlanner. Although Drools tool consumes a lot of computer resources, current mobile devices already have high performance and it might be interesting to have such instrument on Android platform.

Chapter 8

Conclusion

Libraries of Java SE API and Android API were compared and it was found that they differ significantly. Out of 38 Java SE API packages, 20 packages are completely missing and 9 packages are incomplete in Android API. On closer inspection, it was revealed that one of the OptaPlanner tool dependencies is missing in Android API. This missing package is named JavaBeans.

For JavaBeans problem were suggested five solutions, namely: Repacking JavaBeans redistribution to Java core namespace, Use of OpenJDK distribution source code, Use of pruned rt.jar from OpenJDK distribution, Use of OpenBeans in OptaPlanner project and Removing and replacing JavaBeans from OptaPlanner. Due to comparison of advantages and disadvantages of individual solutions, it was selected a solution which does not change source code of OptaPlanner tool, has a license that allows easy application and generally is suitable for usage on Android.

Chosen solution for further progress consists of repacking of OpenBeans redistribution to Java core namespace. During this process, new JAR file consisting of the missing JavaBeans libraries is created and could be subsequently inserted into an Android project. However, this faces the problem which does not allow to use classes of Java core namespace on Android. The problem was resolved by using the Core library flag and the entire process of the portation was designed, implemented and automated by Gradle language – the default build tool for Android projects.

According to the written requirements, model Android Vehicle Routing Problem application which uses the ported OptaPlanner was designed and implemented. The application can use one of the three algorithms, set calculation time limit and solve attached problem examples. Actual problem is always displayed in text and graphic form on the screen of the application together with the newest found solution. The application is publicly available on Google Play Store on the Internet and the presentation video was created and was presented along with solution how to use OptaPlanner on Android to developers community on OptaPlanner website.

On the Vehicle Routing Problem example, comparative measurements that focus on performance comparison between mobile devices and desktop computer were performed. These measurements proved that OptaPlanner is working and can be used on Android.

The challenge for the future work is portation of Drools tool. It is standalone project which can be used by OptaPlanner as one of the option for calculation the score and in the future, it may be interesting to have such tool on Android.

Bibliography

[1] Grant Alleb. *Android 4: Průvodce programováním mobilních aplikací*. Computer Press, 2013. ISBN 978-80-251-3782-6.

[2] Lucas Amador. *Drools Developer's Cookbook*. Packt Publishing, 2012. ISBN 978-1-84951-196-4.

[3] Oracle Corporation. Java standard edition 6 documentation. [online]. URL: http://docs.oracle.com/javase/6/docs/, 2014 [cit. 2015-01-17].

[4] Oracle Corporation. The javabeans api. [online]. URL: http://openjdk.java.net/groups/swing/beans/index.html, 2015 [cit. 2015-01-10].

[5] Oracle Corporation. Openjdk. [online]. URL: http://openjdk.java.net/, 2015 [cit. 2015-01-10].

[6] Benjamin J. Evans and David Flanagan. *Java in a Nutshell*. O'Reilly Media, Inc., 2015. ISBN 978-1-449-37082-4.

[7] Apache Software Foundation. Apache harmony. [online]. URL: http://harmony.apache.org/, 2010 [cit. 2015-01-10].

[8] Apache Software Foundation. Apache ant project. [online]. URL: https://ant.apache.org/, 2015 [cit. 2015-05-24].

[9] Apache Software Foundation. Apache maven projec. [online]. URL: https://maven.apache.org/, 2015 [cit. 2015-05-24].

[10] Code Google. Openbeans. [online]. URL: https://code.google.com/p/openbeans/, 2011 [cit. 2015-01-04].

[11] Code Google. Jar jar links. [online]. URL: https://code.google.com/p/jarjar/, 2012 [cit. 2015-01-04].

[12] Code Google. Madrobot. [online]. URL: https://code.google.com/p/mad-robot/, 2013 [cit. 2015-01-04].

[13] Google Inc. Android app developer resources. [online]. URL: http://developer.android.com, 2014 [cit. 2015-01-04].

[14] Google Inc. Dashboards. [online]. URL: https://developer.android.com/about/dashboards/index.html, 2015 [cit. 2015-05-23].

[15] Gradle Inc. Gradle. [online]. URL: https://gradle.org/, 2015 [cit. 2015-05-24].

[16] Red Hat Inc. Optaplanner – planning engine. [online]. URL: http://www.optaplanner.org, 2014 [cit. 2015-01-04].

[17] Red Hat Inc. Drools. [online]. URL: http://www.drools.org/, 2015 [cit. 2015-01-10].

[18] Red Hat Inc. Kie group. [online]. URL: http://www.kiegroup.org/, 2015 [cit. 2015-05-25].

[19] Red Hat Inc. Optaplanner distribution. [online]. URL: http://download.jboss.org/optaplanner/release/6.2.0.Final/optaplanner-distribution-6.2.0.Final.zip, [cit. 2015-05-08].

[20] Red Hat Inc. Optaplanner user guide. [online]. URL: http://docs.jboss.org/optaplanner/release/6.2.0.Final/optaplanner-docs/html_single/index.html, [cit. 2015-05-08].

[21] RTC Magazine. Android architecture. [online]. URL: http://rtcmagazine.com/files/images/3420/RTC05_TS_Viosoft_Fig01_original.jpg, 2012 [cit. 2015-01-04].

[22] NEO-University of Málaga. Vehicle routing problem. [online]. URL: http://neo.lcc.uma.es/vrp/, 2015 [cit. 2015-05-24].

[23] Neil Smyth. *Android 4.4 App Development Essentials*. CreateSpace Independent Publishing, 2014. ISBN 978-1495358067.

[24] Jiří Vávrů and Miroslav Ujbányai. *Programujeme pro Android*. Grada Publishing, 2013. ISBN 978-80-247-4863-4.

Appendix A

Content of the DVD

- **app/** – directory with APK file of the VRP application.
- **text/** – directory with PDF file of this master's thesis.
- **video/** – directory with the presentation video.
- **app-src/** – directory with source code of the VRP application.
- **text-src/** – directory with LaTeX source code of this master's thesis.
- **video-src/** – directory with source files of the presentation video.
- **README.md** – file with instructions.