Software Development Document

Internal workings of PickUApp

Group NULL:
Elvira Jonsson
Ivar Josefsson
Francine Mäkelä
André Samuelsson
Malin Thelin
Joakim Thorén

Chalmers University of Technology
Software Engineering Project (DAT255)
2014-10-30

1. Introduction

This documents instructs how to build and install PickUApp, it describes and explains the general architecture of PickUApp aswell as some headsup that needs to be considered during development. It does not intend to motivate the technicalities. It does not intend to describe every technical aspect of the application. Clean and self-explanatory code over documentation is a principle rule.

The software architecture-chapter is split into five subchapers - each subchapter delves deeper into the actual topic.

2. Building

The .apk file can be built from command line using gradle.
Below is a description on how to do this:

1. Download and add the android sdk to path. https://developer.android.com/sdk/installing/index.html?pkg=tools
2. Clone repository from https://github.com/cannonbait/dat255
3. Get gradle v1.12 from http://www.gradle.org/downloads
4. Add gradle to your path. (or simply refer to "gradle-1.12/bin/gradle" when you want to build)
5. Navigate to DAT255/project and run "gradle build"
    - If you haven't added gradle to path.
      run "[Path_To_Gradle]/gradle-1.12/bin/gradle build"

You should now have "app-debug-unaligned.apk" located in: ../app/build/outputs/apk/

6. Google Maps requires a private-key when building in order to authenticate towards google-maps servers. This key is not available on the gitrepo. You need to create an Google API-key on your own, follow this tutorial: https://developers.google.com/api-client-library/python/guide/aaa_apikeys

3. Installation

1. You need to have either an android device running a minimum android verison of 4.2, or you need to emulate an android device. Google play is required which may require some extra setup when running from an emulated device.
2. On the device, make sure Android debugging is active in the developer options under settings.
3. Use "android-sdk/platform-tools/adb" to install and get logging from the application.
4. Run "adb devices" to show currently available devices.
5. "adb install [path to .apk]" installs apk to device. Use '-r' flag to reinstall if a previous version is present.

4. Software Architecture

The software architecture is derived from a flowchart, as opposed to traditional object-oriented-programming. This means that each step in the PickUApp flowchart is represented by a class, instead of subjectives from an application-description. This step is called a “Sequence”.

4.1 Sequences

A brief description of a Sequence would be that “A Sequence contains logic on a specific set of events. Another sequence contains different logic on another set of events”. A detailed description is as follows:

The application should be thought of as a state-machine, where each Sequence knows what to do on a specific event (more on events at Eventbus-chapter). The sequence could either step-forward to another sequence, manipulate some data or whatever intention the sequence has, on recieving an event.

For example: The starting sequence is “ChooseModeSequence”. This sequence has an unique view, which shows two buttons - “Driver” and “Hitchhiker”. These buttons sends an event on the eventbus whenever they’re pressed. The sequence is listening to the eventbus and knows itself how to handle these events. If “Driver” was pressed, “ChooseModeSequence” knows that it should ask the model to step forward to another Sequence called “DriverSetRouteSequence”. DriverSetRouteSequence in turn has logic for another set of events which are generated for button-presses on the DriverSetRouteSequence’s view. So, each sequence knows how to process events generated by it’s own view through the Eventbus.

As stated eariler, the application should be thought of as a state-machine where each state (Sequence) sets the view and defines behaviour on interacting with the view through the Eventbus. Hence the software architecture is closely related and dependent on the application-flowchart.

4.2 Eventbus

The Eventbus is a singleton-class “EventBus” which can register listeners, unregister them and report events to listeners. Any listener needs to implement the EventListener-interface and realize onEvent(Event event)-method. The Eventbus is available to any class and any class is able to report events - classes do not need to be registered as publishers or senders.

4.3 Activities

There are two activities who stands out from the rest. First there is MasterActivity, which is the only activity whose life span is equal to that of the application. This activity knows about all other activities that can possible be created when the application is running, and also which sequence class it belongs to. Each sequence is represented by an unique activity. This relationship is set manually. The information is used to change activity and therefore change the view.
The other activity worth to be mention is MapsActivity whose purpose is to show a map and, if any location set, it should also show markers for the place the driver and hitchhiker are going to meet and where the driver are supposed to drop of the hitchhiker. Unlike most of the activities, MapsActivity isn't created by MasterActivity, instead that responsibility lies on the activity who is running at the moment.

4.4 MVC

The application uses an MVC pattern where the View contains both the traditional View aswell as the Controller (sending user input etc.). The View is heavily based on the Android framework and heavily dependent on an Eventbus of the kind that is implemented. The EventBus is altogether stand-alone. The Model, on the other hand, has plenty of dependencies on the EventBus. It is made this way to have close to no dependency on the view i.e the android framework. The Model heavily uses Sequences, described above and recieves information only through events. Every buttonpress is sent as a unique event to the eventbus which in turn forwards it to all listeners, mainly Model, View and Network.

4.5 Network

The network is started separately and is given opportunity to register listeners and establish connection to the server etc. Currently there is one interface and two classes in the network package. There is a wrapper, NetworkWrapper, an interface, Network, and a MockNetwork class realising the Network interface. The wrapper is responsible for listening to the eventbus and catching events relevant to the network; aswell as calling appropriate methods on a realisation of the Network interface. Currently the Network interfaces defines five required methods that the implementation must fulfill. Currently there is a MockNetworkClient class that sends no data to the server, instead starting a thread that sleeps for a set amount of time before sending a response to the eventbus.

5. Implementing a Sequence

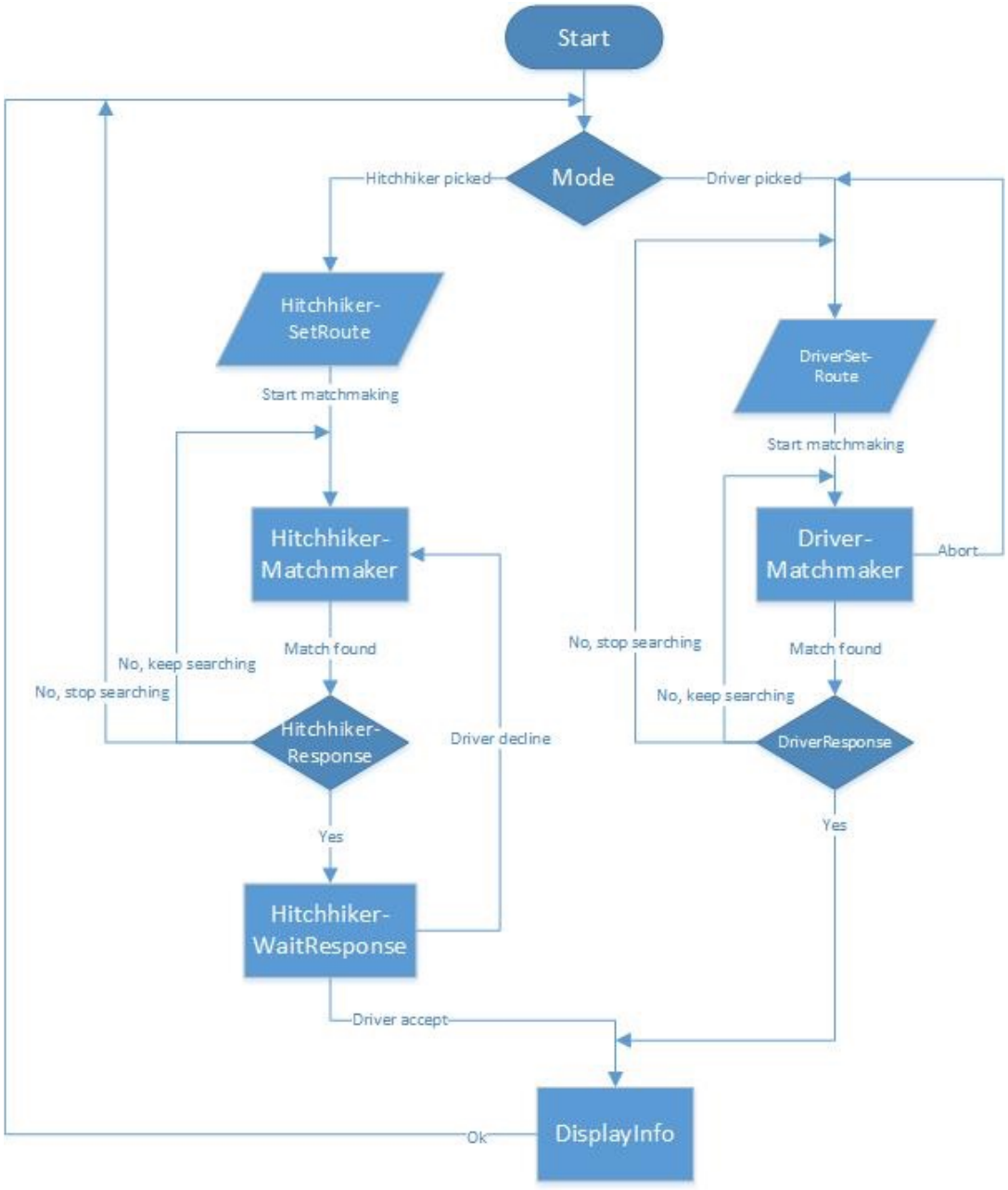
Implementing a sequence is not bounded or limited by much, but some things must be considered. There are overhead when implementing a sequence:

- The sequence must be paired with an activity within the MasterActivity.setupAvailableActivities()-method. Otherwise the MasterActivity won't know what activity to start when stepping into a new sequence.
- An instance of each sequence must be paired with the sequence's class-variable within model constructor.

Data-transfer between sequences can be handled by creating a transferData(Data myData)-method within MySequence which is called from the previous sequence (which has the data that should be transferred from previous to MySequence). Throughout the application these types of methods are occasionally called "insert()".

6. Flowchart

Each step in the flowchart is represented by a Sequence in the application. All sequences in the application is represented by a step in the flowchart aswell, it is a 1:1 relationship.



7. Test Driven Development

Tests are carried out using JUnit-4.10. Test-sources are located in DAT255/project/app/src/main/test/edu/chalmers/pickuapp/app/model/. Black-box testing is the primary methodology of testing. While building with gradle tests are automatically executed and a report.html is created and saved in DAT255/project/app/build/reports/tests/index.html. All tests should be succesful in order for a class to be considered finished. If any tests fail during development, they should be fixed and succesful before pushed to master.

8. Gradle

Gradle is a build-system which manages project-dependencies. Gradle is configured using a build.gradle-file, located in DAT255/project/app/build.gradle. All dependency-management should be handled via gradle.