

T-110.5130 Mobile Systems Programming Final Report

A.M. Ahsan Feroz
abdullah.feroz@aalto.fi

Md. Mohsin Ali Khan
md.m.khan@aalto.fi

Elena Oat
elena.oat@aalto.fi

Hasan M. A. Islam
hasan.islam@aalto.fi

Md. Mesbahul Islam
mdislam@cs.helsinki.fi

Tutor: Antero Juntunen

May 8, 2013

Contents

1	Introduction	3
1.1	Report Outline	3
2	Evaluation of the Results	5
2.1	What was our goal?	5
2.2	Results	5
2.2.1	Day view	5
2.2.2	Week view	5
2.2.3	Month view	6
2.2.4	Preference Settings	6
2.2.5	Action Bar	7
2.2.6	Database	7
2.2.7	Service Granularity in Database	8
2.3	Prospective Development Ideas	9
2.4	Challenging Technologies	9
2.4.1	SQLiteDatabase	9
2.4.2	Layout	10
2.4.3	Preference Settings	10
2.4.4	Alarm Manager	11
2.4.5	Application Navigation	11
2.4.6	XML Layouts	11
2.4.7	Screen Orientation	12
2.4.8	NotificationManager	12
3	Metrics	13
3.1	Used Resources	13
3.2	Quality Metrics	13
3.3	Software Size	13
3.4	Analysis	13
4	Work Practices and Tools	15
4.1	Code Repository / Version System	15
4.2	Issue Tracking	15
4.3	Group Discussion	15
4.4	Testing and Code Review	15

5	Preliminary business model	16
5.1	Service Design	16
5.2	Technology Design	17
5.3	Organization Design	17
5.4	Finance Design	17

Chapter 1

Introduction

Nowadays people are so busy with their daily activities and hence, it is more likely to forget about an import task. In the present hi tech world mobile phones could be a great medium to keep track of those and notify the user at the right time about a task. We have developed 'Task Scheduler' application specifically to manage these needs. The first version of our Task Scheduler is a typical one, which maintains and keeps track of all events. The user inputs some data to application and the app is responsible of reminding the user by means of alarm about the events. The alarm service is always running in the background and the reminder pops up as a small screen at the desired time. The application also has a view screen which can be used as an widget in the desktop with the list of tasks. The main motivation behind choosing Android platform are the following:

- Android is the most widely used mobile platform from the perspective of the number of users and number of application
- Android app developers use the classic open source Linux OS. Any open source provides help for users to access its source code transparently and is available to any developer who wants to modify it or see how it works.
- Google provides all Android developers with an open source software development kit for the Android OS. They can create applications for Android and then test them on an Android simulator before loading them onto an actual Android phone.
- The future of the Android platform looks bright for Android application developers who love to have access to everything. We can expect to see some interesting innovations both from Google and from the public as Android gains popularity.

1.1 Report Outline

Chapter 1 will evaluate the results of our implementation with prospective development ideas and list of all challenging technologies we used in our app. Summary of used resources, quality metrics, software size, etc., will be outlined in chapter 2 with some comparison analysis to standard Google Calendar. Work

practices and tools, as well as the preliminary business model will be presented in chapter 3 and 4, respectively.

Chapter 2

Evaluation of the Results

2.1 What was our goal?

The primary goal of the project was to learn how to develop a mobile application for Android platform. The scope of the project was to develop an application that provides the user functionality to do the following activities:

1. View Existing Events
 - Day view
 - Week view
 - Month view
2. Create New Events
3. Edit/Delete Existing Events
4. Alarm and Bar notifications of the Events at the due time

2.2 Results

2.2.1 Day view

When user clicks on an empty slot on the day view layout, the application will ask user whether he or she wants to edit or delete the selected event. In case there is no event for the current time slot, user cannot delete it, but can enter task title, description, etc. to create a new task for the current slot. If the user clicks on the slot which has some data, he/she can edit or delete the event.

2.2.2 Week view

This view gives a snapshot of tasks for one week. User can also navigate to following weeks using the arrows in the top part of the view.

- Initially the week view is loaded with the current week days. There are two buttons, 'prev' and 'next', for navigating to previous or next weeks, respectively.

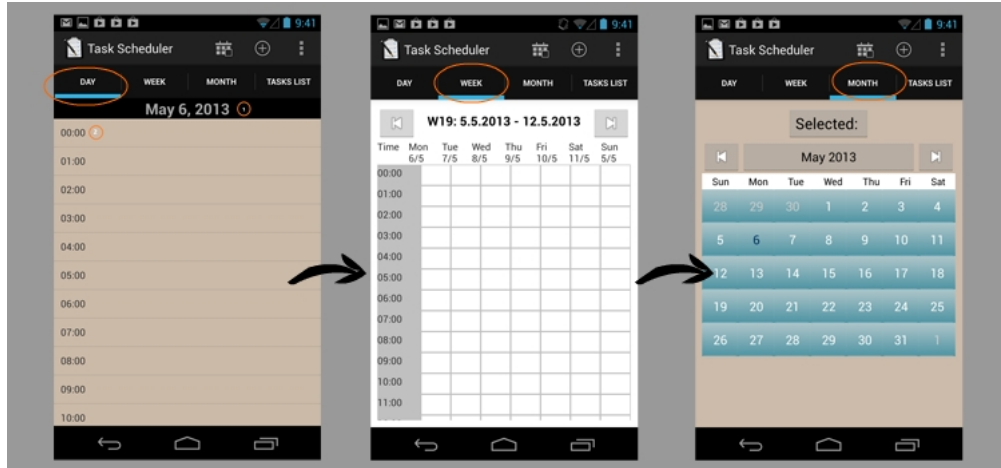


Figure 2.1: Day, Week and Month View

- The week days are partitioned into 24 hours. Thus, there are 168 (24×7) 'Cell's in this view. Each cell points to a specific hour. If there are saved tasks which are created for these week days then the corresponding cells are marked 'Green' when created.
- Each cell has two modes, 'Selection' and 'Navigation'. When a user touches a 'Cell' for the first time, it goes into 'Selection' mode. The cell will be in the 'Navigation' mode when same cell is touched again. If the Cell is empty then it takes the user to the "Create new task" view. Otherwise, a dialog pops-up with associated saved task data and three action buttons Edit, Delete and Cancel.

2.2.3 Month view

This view presents a month layout.

- Initially this view shows current month. A user can navigate to next or previous months through 'next' and 'prev' image buttons.
- If user selects a date in the month view then the selection will redirect user to the corresponding day view.

2.2.4 Preference Settings

Our application includes settings feature that allows users to modify app's features and behaviour. For example, users could choose such preferences as when the user has to be reminded about a task, e.g. 30 minutes before, 45 minutes before, etc. The settings set are default for all new events. These can be modified individually for each task when creating the new event or when editing it.

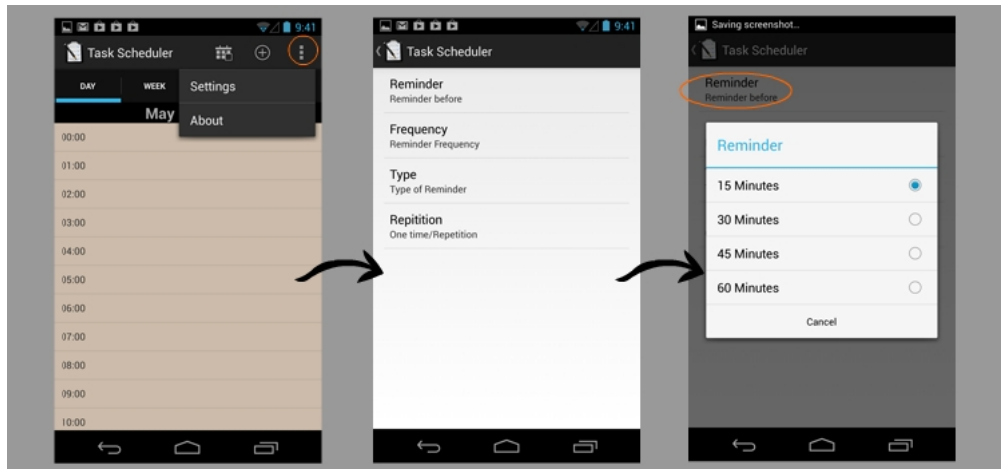


Figure 2.2: Preference Settings

2.2.5 Action Bar

Action bar with Navigation has been implemented with the following features:

- Action bar MODE is a TAB navigation. There are four TABS. Among these three TABS are responsible for three different views and the fourth TAB is responsible for showing all saved data.
- There is a menu on the top right corner, which includes **Settings** and **About**. Users can set own preferences through **Settings** that include the followings:
 - Reminder
 - Notification Before
 - Repetition
 - Type of Reminder

2.2.6 Database

The application uses an SQLite database to store information about all the tasks created by the user, as well as to store default values of the global properties of the application. Methods for database access and data modification can be found in DatabaseAdapter class.

Table names are the following:

- master_event: to store all the events
- global_config: to store all the default values for the application settings: notification before (minutes), notification frequency, notification type. Whenever a new event is created, the values specified in this table will be used to create a new event, unless the user explicitly modifies these.

2.2.7 Service Granularity in Database

To integrate the above mentioned functionalities of our app at the front end with the database tables, we implemented an interface between the front end and the database. The interface includes the following basic primitive services:

- `createEvent`
- `deleteEvent`
- `editEvent`
- `getEventByDate`
- `getEventByID`
- ...
- ...
- etc.

`createEvent` method is responsible for inserting a task into the database. Its processing details are as follows:

- If the `recurrenceFlag` parameter received from front end is null, it inserts a new row in the `master_event` table using all the other given inputs. The `id` column is automatically populated by the database. In this case the `parentID` column of this table is populated as 0.
- If the `recurrenceFlag` is not equal to "Once", i.e. it is "Daily", "Weekly", or "Monthly", the responsible method will check the `eventEndDayTime` and will find/calculate all the child events. Then it will insert one row as a parent event for the first occurrence and one row for each child event. The method will also populate the `parentID` column of the parent event with 0. The value of `parentID` column of the child events are populated with the value of the `id` column of the parent event.

`deleteEvent` deletes all the rows from the `master_event` table which has the value `id` (received as input) in their `id` column or in their `parentID`.

`editEvent` uses `createEvent` method to edit a task.

- If the event that needs to be edited is a one-time event or a parent event of a recurrent event, it deletes the event by calling the `deleteEvent` method and creates a new Event by calling the `createEvent` method
- If the received event to be edited is a child event of a recurrent event, then it deletes the event by calling the `deleteEvent` method and then creates a new Event by calling the `createEvent` method. While calling the `create event` method, it sets the recurrence flag to null and sets the `editparentID` variable to the ID of its `parentID`. This `editParentID` ensures that the edited child event still remains the child of its parent.

2.3 Prospective Development Ideas

2.4 Challenging Technologies

List of technologies used and challenges we had to undergo during development of our app are briefly outlined in this section.

2.4.1 SQLiteDatabase

Trouble with properly setting up SQLiteDatabase At the beginning of our database implementation we frequently faced the following error:

```
ERROR/Database(234): Leak found
ERROR/Database(234): Caused by: java.lang.IllegalStateException:
    SQLiteDatabase created and never closed
ERROR/Cursor(31526): Finalizing a Cursor that has not been
deactivated or closed.....
ERROR/Cursor(31526): android.database.sqlite.
DatabaseObjectNotClosedException: Application did not close
the cursor or database object t
hat was opened here
```

We realized that this exception is thrown when we have opened more SQLiteDatabase instances than we have closed. The same is true for Cursor which is used for retrieving data from the database. Managing the database can be complicated when first starting out with Android development, especially to those who are just beginning to understand the Activity lifecycle. Our learning experience tells us that the easiest solution would be to make a database instance as a singleton instance across the entire application's lifecycle. This will ensure that no leaks occur and will make your life a lot easier since it eliminates the possibility of forgetting to close your database in the code. Our application uses DatabaseAdapter class for the whole application.

The approach we used for the Singleton Database is creating the Abstract Factory to instantiate the SQLiteOpenHelper.

Using ADB shell ADB shell works only after application launch on the device or emulator.

```
hasan@elitemorm:~$ cd android-sdk-linux/platform-tools/
hasan@elitemorm:~/android-sdk-linux/platform-tools$ ./adb shell
#
# cd /data/data/com.taskmanager/databases
#
# ls
taskManager.db
#
```

```

# sqlite3 taskManager.db
SQLite version 3.6.22
Enter ".help" for instructions
Enter SQL statements terminated with a ";"
sqlite>
sqlite> .schema

CREATE TABLE android_metadata (locale TEXT);
CREATE TABLE global_config(property TEXT PRIMARY KEY, valueType TEXT,
    textValue TEXT, integerValue TEXT, dateValue TEXT);

CREATE TABLE master_event(id INTEGER PRIMARY KEY AUTOINCREMENT,
    recurrenceFlag TEXT, recurrenceEndDate TEXT, parentID TEXT, name TEXT,
    description TEXT, createTime TEXT, dueDate TEXT, notificationB4 TEXT,
    notificationFreq TEXT, notificationType TEXT);

sqlite>
sqlite> select * from global_config;
NotificationB4|int||Fifteen|
NotificationFreq|int||Five|
NotificationType|text|Both|Five|
sqlite>

```

When we became familiar with checking our app database using ADB shell, it significantly simplified the task of running the queries on the database. A database and its data can be removed, inserted, updated using ADB shell. Thus, ADB shell is good for experimentation.

2.4.2 Layout

2.4.3 Preference Settings

We have used Android's **Preference** APIs to build an interface that is consistent with the user experience in other Android apps (including the system settings). Each **Preference** appears as an item in a list and provides the appropriate UI for users to modify the setting.

Our app requires to store the changes in preferences. For this we had to use **SharedPreferences**. Each Preference you add has a corresponding key-value pair that the system uses to save the setting in a default **SharedPreferences** file for your app's settings. When the user changes a setting, the system updates the corresponding value in the **SharedPreferences** file. The app needs to directly interact with the associated **SharedPreferences** file, when it reads the preferences set in the settings of the application.

In order to make use of the preferences framework, the first step is to extend the **PreferenceActivity** class. This is just a convenience class that derives

from `ListActivity` and allows to bootstrap the preferences UI from an XML file. Additionally, it automatically saves those to the `SharedPreferences` behind the scenes. Note that `SharedPreferences` is the interface responsible for accessing and modifying preference data and that we can manually manipulate it by calling the `getSharedPreferences` method of an Activity. In order to tie together our `PreferenceActivity` class and the XML layout file, we use the `addPreferencesFromResource` method.

2.4.4 Alarm Manager

Prior to a set time of the start time of an event/task, the application notifies the user with an alarm. The user can set this prior time while creating an event/task. Otherwise, the prior time will be set from the default settings of the application. When the alarm is generated, a new activity is loaded and the default alarm ringtone of the application starts to play. In the activity layout there is a button named “Notified”. When user taps this button, the phone gets back to the context where it was and the audio stops to play.

The default `AlarmManager` class has been used to implement this functionality. Whenever an alarm is created in the database, a Pending Intent is created. This Pending Intent is created with a normal intent that fires an activity. The activity loads the Alarm layout and plays the default ring tone. Then the Pending Intent is sent to the Android system through the `AlarmManager` to be executed at the specified time.

The Pending Intent remains in the system even if the application is not running and loads the activity when the specified time comes. It also does the same even if the phone is switched off.

2.4.5 Application Navigation

`ActionBar` is a very practical Android feature, which enriches the application with the actions bar and a smooth navigation between tabs. At the beginning of this application development we used only Activities and buttons to navigate between activities. But later we realized that `ActionBar` provides the navigation feature in a smarter way. It improves the UI also. But `ActionBar` works for API level 11 and above. There is also a support library which could be used to make the `ActionBar` compatible with older devices.

We have also implemented ‘Option Menu’ for general ‘Settings’ and ‘Action Items’ to give the user immediate action for a specific situation. For example, ‘Save’ or ‘Cancel’ action when creating a new task or ‘Edit’ and ‘Delete’ actions in the Day View.

The challenges we faced include working with `FragmentManager` as we need to navigate and communicate between Tab fragments.

2.4.6 XML Layouts

We experimented with different layout types in our application. Most of the XML files use Linear Layout. We have avoided more complex and nested layouts

in our application. If the nested level is high then it will be a performance issue and the application will take time to render the view.

In the Day View we used `ListView` to show the hour slots of a Day. The Week View and Month View are created dynamically at the run time of the application.

2.4.7 Screen Orientation

The activity stops and restarts every time the device orientation changes. To save the state of the application, Android calls the `onSaveInstanceState(Bundle bundle)` method before it destroys the Activity. The bundle object is passed in the `onSaveInstanceState()` callback method to save small amount of data. Bundle is not designed to handle large amount of data. Restoring the saved state is done with the `onCreate()` or `onRestoreInstanceState()` method.

We can also handle screen orientation in the following way:

- In Activity's manifest node we need to add:

```
android:configChanges="keyboardHidden|orientation"
```

- Then within the Activity override the `onConfigurationChanged` method and call `setContentView` to force the GUI layout to be re-done in the new orientation.

```
@Override
public void onConfigurationChanged(Configuration newConfig) {
    super.onConfigurationChanged(newConfig);
    setContentView(R.layout.myLayout);
}
```

2.4.8 NotificationManager

A notification is a message that notifies the user when he or she is outside of our application's UI. We need to set a timestamp to tell the system to issue a notification, it first appears as an icon in the notification area. Our application notifies the user about a task through status bar notification and alarm manager.

Chapter 3

Metrics

3.1 Used Resources

We have implemented our project in Eclipse and tested with Samsung phone and Tablet, HTC mobile phone and Emulator. Some of our team members build application in windows operating system and some in Linux operating system.

Every group member has an account on **Github**. A repository was created, which serves as a place for common development. All commits are accompanied by a message which states what changes have been made to the current version.

3.2 Quality Metrics

- User friendly
- Error density
- Robustness
- Liveness
- Application Security

3.3 Software Size

Total lines of Java source code are approx. 3500. The whole package weighs approximately 2.7 MB.)

3.4 Analysis

We have compared our application with Google's Calendar.

In Google calendar a user can create an event/task. Our application can do the same. When a task approaches its due time, Google calendar notifies the user through an e-mail (as an option). In comparison, our application generates

an alarm with the phone's default ringtone.

In Google calendar a user can create a repeated event based on the day's of a week. For example, every Monday, or every Tuesday or every Monday and Friday. In our application user can create repeated events based on the daily, weekly, monthly recurrence.

In Google calendar user can edit or delete an existing event. In our application user can do the same.

In Google calendar whenever the user edits the occurrence of a repeated event, it asks the user about editing only that particular event or the whole series. In our application, when the user edits the child event, only that event is edited. And when the user edits the parent event, the whole series is edited.

In Google calendar you can view your calendar from a day view or a month view. In the month view you can click on a week and it shows the events of that particular week. In our application, we have separate tabs for the day view, month view and week view.

In Google calendar from the month view you can click on a particular day and it opens the day view for that particular day. It is the same case in our application.

In Google calendar if you click on a particular time slot in the day view it opens the create event dialogue to create a new task. Our application does the same.

Chapter 4

Work Practices and Tools

4.1 Code Repository / Version System

As we have worked in a group, we have to save and share code among group members. Moreover, synchronization and code merging is needed after completion of a defined milestone. There are open source services/servers which provide a good way for maintaining those.

- Git
- SVN
- CVS

We have used github repository for version control and code synchronization.
<https://github.com/elenaoot/msp>

4.2 Issue Tracking

Issue tracking is a way to create, monitor, manage and filter different issues throughout project development or maintenance of an application. There are different types of issues, for example, bugs, code enhancement, question, fixing etc. At first we were maintaining a shared Google Sheets for this purpose. But later we discovered the 'Issues' feature of github which is integrated with project repository.

4.3 Group Discussion

Knowledge, best practices and important notice sharing through WordPress wiki. Weekly meeting, bi-weekly day long code camp.

4.4 Testing and Code Review

Testing tasks have been performed to identify the bugs and correct them.

Chapter 5

Preliminary business model

The primal idea behind the service is intended to provide help for users to manage their day to day important tasks and schedules. In this chapter we analyse our development according to STOF framework.

5.1 Service Design

Intended Values: The intended values are as follows:

- Facilitate the user to to make entries of future schedules,
- Help the user to analyse the task load and vacancy for a new schedule,
- Notify the user prior to the start time of the schedule.

Technical Architecture: To make the intended value easily achievable by the user, a platform is required that user have very frequent access. Now a days, a smart phone is part of the life of billions of mobile phone user. And a significant portion of these users use Android Operating System.

It is very likely that through the popularity of android Operating System the intended service can reach to the maximum number of users possible.

As a result the service is delivered to the users as and application runnable in android operating system on a mobile phone

To make the intended value easily achievable by the user, a platform is required that user have very frequent access. Now a days, a smart phone is part of the life of billions of mobile phone user. And a significant portion of these users use Android Operating System.

It is very likely that through the popularity of android Operating System the intended service can reach to the maximum number of users possible. As a result the service is delivered to the users as and application runnable in android operating system on a mobile phone. There is already numerous applications available in the android market trying to provide the same kind of service. In some cases, those are not very user friendly and in some cases those are heavy weight. Users have still the appetite to get this service through a more light weight and user friendly android application.

All these factors have driven the group to develop an android application that can deliver the values in section 2.2.

5.2 Technology Design

All group members have considered the service to be provided through an application runnable in Android Operating System. Android is a open source platform and the only requirement the users have to meet is to have a mobile phone with Android OS. The user can download it from the appstore and use it. It is a stand alone application that do not rely on some other application or systems except the Android Operating System itself. It doesnt need to have an Internet connection as well except the case of downloading it initially from the appstore. The group members have used GIT as the central repository for the application code to synchronize their coding effort.

The service owners were required to have different Android Mobile phones to test their application, to make sure that the application runs in the Mobile Phone comes from different companies and have different API levels.

In all point of view the service doesn't incur any technology cost for the owners except the Human Resources to develop it, workstations with necessary development IDE and Internet connection to access the Android appstore and GIT.

5.3 Organization Design

Our group will monitor the popularity of the application, get feedback on user's experience and think further on how the application can be made more simple, user friendly and be integrated with latest Android functionalities or be compatible with the upgraded API level.

The group members identify potential development need by analyzing the value network, technology and user experience. Depending on the this, the group members will modify the Service design, Technology design and Finance design if required.

5.4 Finance Design

We can monitor the download count of the application from the appstore. If the delivered service add values to the daily life of the users the count goes up we can make the application as a paid application to download. Or we can arrange some advertisement in the application. The user doesn't have to pay, and we get money. It's a win-win situation. Currently, all AppsGeyser apps run ads for the application owners, and when the app gets enough uses, we get paid. Depending upon the popularity and the incurred operational and constant development cost, we can revise the price of the application.