CAS 703 Final Project Report

A Walking Speed Detector

Ye Li
Couputing and Software
McMaster University
li554@mcmaster.ca

ABSTRACT
The mobile application development industry is increasingly growing up because the heavy demand for mobile applications used in the daily life. Using MDE approach in development of smart phone apps can improve the work efficiency. This report illustrate the case study that developing a walking detector on Android cellphones by using the MDE tool ADT supplied by Google company. It will show the details to use model based technology to develop an Android application.

Categories and Subject Descriptors
D.2.8 [**Software Engineering**]: Development—*model-driven engineering, Android*

General Terms
Project

Keywords
Android, MDE, model, walking speed detector

1. INTRODUCTION
Mobile devices such as smartphones and tablets have dramatically increased in popularity. There are three popularity mobile terminal platforms: iOS, Android, and Windows. Most of the devices are running on Android Operating System. Before talking about Android platform, I will illustrate the basic concept of Model Driven Engineering.

1.1 Model Driven Engineering
Model-drien engineering (MDE) is a software development methodology which focuses on creating and exploiting **domain models**, rather than on the computing (for example algorithmic) concepts.[1] The domain model is created in order to represent the vocabulary and key concepts of the problem domain. It also identifies the relations among all the entities within the scope of the problem domain, and commonly identifies their attributes. The domain model provides a structural view of the domain that can be realized by other dynamic views, such as use case models.[2] An advantage of adoption of the domain models is that it describes and constrains the scope of the problem domain. Domain models define a precise vocabulary and help the teammates communicate with each other.

A **domain-specific language (DSL)** is a computer language specialized to a particular application domain. A domain-specific language is created specifically to solve problems in a particular domain and is not intended to be able to solve problems outside it. DSLs can usually cover a range of abstraction levels for a particular domain. For example, a DSL for mobile phones could allow users to specify high-level abstractions for the user interface. There are a wide variety of DSLs, ranging from widely used languages for common domains, such as HTML for web pages, down to languages used by only a single piece of software.[3] A **meta-model** is a model of models which is used for defining modelling languages. One can use meta-modelling to create DSL. A DSL is a modelling language made to model a particular problem domain and it is based on a meta model that captures essential concept of the domain.

Meta-model and DSL have a tight relationship with MDE. In MDE, meta-model defines the DSL as a modelling language. DSL generate the code according to the models.

1.2 Android Platform
The Android is an Operating System (OS) based on the Linux kernel held by Google to mobile devices. Its applications are developed in Java, but run on a specific virtual machine called Dalvik[4] The Android platform provides an API with some components such as activity, service, broadcast receiver, and content provider to develop a mobile application. There are one or more activities contained in an Android application. An activity is a visual user interface and it includes some interactive controls. Service can be treated as a special type of activity that runs in the background for an indefinite period of time.

An activity has strict lifecycle defined by six methods: *onCreate, onStart, onResume, onPause, onStop,* and *onDestroy. onCreate* method is invoked when starts an application. *onStart* show the activity in foreground in the screen, while *onResume* is invoked if the activity is lost on the screen. When the user change one application to another, *onPause* is invoked. The *onStop* method is running when the activity is invisible, and *onDestroy* is the last task to be operated before the activity is closed. However, as for the lifecycle of a service, only two method are used. The serviceâĂŹs *onBind* method is used to make a persistent connection with a service, while *onDestroy* to terminate it.

An activity may transfer control and data to another activity through an interprocess communication protocol called intents. To promote this, a Broadcast Receiver (BR) com-

ponent is used, which is a dedicated listener that waits for broadcasts messages. When an intent is broadcasted, the BR responds by executing a specific activity. Requesting for common datasets (e.g. contacts, pictures, messages, audio, and files) are handled by the Content Provider (CP), which is a data-centric service that makes persistent datasets available for Android applications. A data requesting is indicated through Uniform Resource Identifier (URI), which provides the standard access to CP. Intents is an interprocess communication protocol that lets an activity transfer control and data to another activity. To promote this, a *Broadcast Receiver* (BR) component is used, which is a dedicated listener that waits for broadcasts messages. When an intent is broadcasted, the BR responds by executing a specific activity.

1.3 ADT is a MDE tool

ADT (Android Developer Tools) is a plugin for Eclipse that provides a suite of tools that are integrated with the Eclipse IDE. ADT provides GUI access to many of the command line SDK tools as well as a UI design tool for rapid prototyping, designing, and building of application's user interface.[5] The following describes important features of Eclipse and ADT:

• Integrated Android project creation, building, packaging, installation, and debugging.
• Java IDE features as compile time syntax checking, autocompletion.
• XML editors let developers edit Android-specific XML files in a form-based UI. A graphical layout editor lets developers design user interfaces with a drag and drop interface.

ADT is a MDE tool to some extend. The reasons are as followings: Firstly, the graphical layout editor is a MDE tool. There are lots of form widgets provided by the layout editor. All of them are models defined by Android platform. Developers can assign property values of each model and the editor can generate the XML of each UI automatically. Secondly, the UIs, activities, system values, string values, etc. are stored in Android-specific XML files. ADT will generate these XML files into Java code. these UIs, activities, system values, string values are models defined by Android platform, so the model domain is Android. XML can be treated as the DSL and ADT transforms the models and generates the code by Android-specific XML. Thirdly, as mentioned above, activity, service, broadcast receiver, and content provider are four core components of an Android application. The four core components are also the models have already defined by Android. When we create an Android project, the ADT automatically generate the code of fore core components according to the values we set in the create guide interface.

From the above three reasons, we conclude that using ADT to develop Android applications employs MBSE technologies and ADT can be called a MDE tool as well. In this section, the report first gives the basic concepts of MDE and Android platform and then discussed that ADT is a MDE tool. In the next section, the report will illustrate how to employ MBSE approaches in ADT to develop my final project.

2. WALKING SPEED DETECTOR

This section we using a study case to show how to develop an Android application through MBSE technologies. The study case is an app called walking speed detector, it can test step numbers, walking length, and walking speed every second and show the information on the mobile screen. This section includes meta-model, interface design, code generation, algorithms, and the final result.

2.1 Meta-model

First we need to build the model of the application, the meta-model is designed by using EMF as Figure 1 shows.

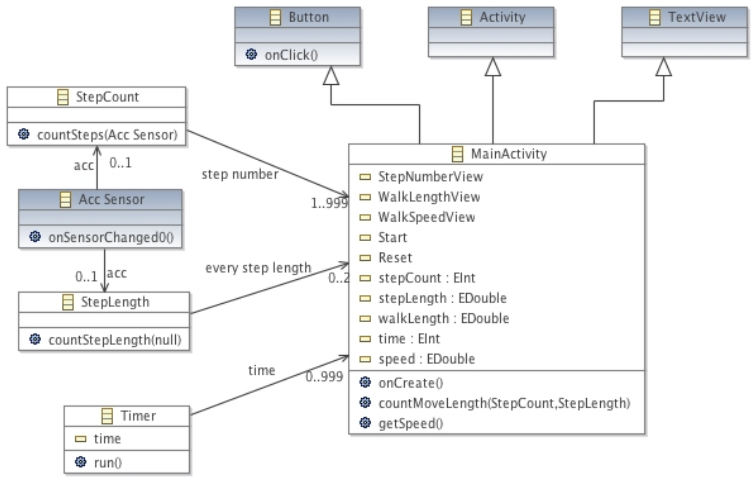


Figure 1: The meta-model of Walking Speed Detector

The Android classes, Activity, Button, TextView, and Accelerate Sensor are highlighted in grey in this model, while the application classes StepCount, StepLength, Timer and MainActivity are illustrated as white-boxes. The MainActivity is the main application class and represents an Android activity, as indicated by the inheritance relationship. For this class, onCreate, countMove, getSpeed methods are explicitly defined, indicating that these methods will be customized. This class is responsible by the user interface and is associated to StepCount, StepLength, and Timer. Both the two classes, StepCount and StepLength, receive accelerator information from the Acc Sensor. Then they use the sensor data to calculate the step number and every step length respectively by countSteps and countStepLength methods. After that, the two class pass the results to MainActivity. The Timer class generate an instance timer that runs in the background. It operates the run method to count the seconds and passes the newest information to the MainActivity.

2.2 User Interface Design

We use the graphical layout editor provided by ADT to design the user interface. In my project, the application only contains one Activity. There are four modules on the MainActivity, Step Number, Move Length, Speed, and two control buttons. The four modules arranges from top to bottom, as shown in the Figure 2.

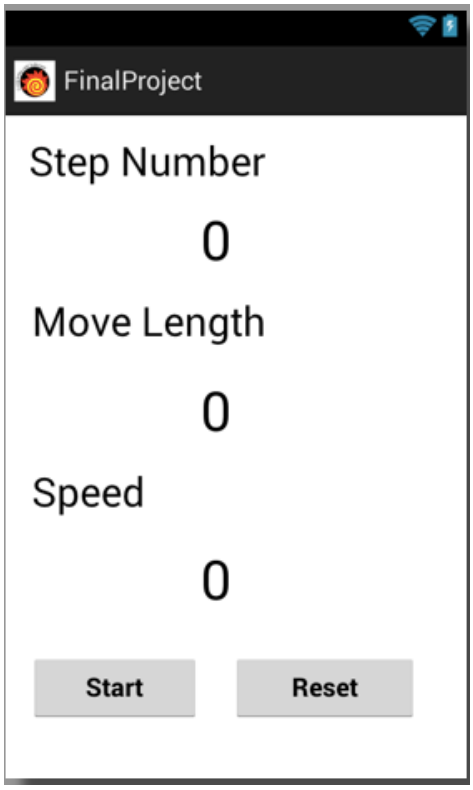


Figure 2: The User Interface of Walking Speed Detector

Form the Figure 2, each of the top three modules is made up with two textViews, one is used as label and another one is used to show the value. The control module includes two buttons, one is Start/Stop which is respond for controlling the application operation, the other one is Reset which make the values return to be 0. When press the Start button the text on itself changes to Stop and the Reset button becomes invalid. When Stop is clicked, itself changes to Start and the Reset button changes to valid again. No matter textViews, buttons or activities, they are all the models that have been defined by Android platform. As we designing the UIs on the layout editor, the ADT will generate the XML code. This feature will be illustrated in the next subsection.

2.3 Code Generation

This subsection mainly shows how ADT uses MBSE technologies to generate the code from the models in Android domain. It generates the code in three aspects which will be showed by the example of my final project.

Firstly, when create the the android application project, we need to assign the application name as Final Project which is shown in launcher. Then we assign the icon image and the background colour. The next is to assign the activity name and choose its style. After all these steps the Android project is created and we can find the ADT have generated the code for us. It generates the controller of MainActivity in Java language, the layout activity and app properties in xml as shown in Figure 3.

```java
package com.CAS703.finalproject;

import android.support.v7.app.ActionBarActivity;

public class MainActivity extends ActionBarActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        if (savedInstanceState == null) {
            getSupportFragmentManager().beginTransaction()
                    .add(R.id.container, new PlaceholderFragment()).commit();
        }
    }

    @Override
    public boolean onCreateOptionsMenu(Menu menu) {

        // Inflate the menu; this adds items to the action bar if it is present.
        getMenuInflater().inflate(R.menu.main, menu);
        return true;
    }

    @Override
    public boolean onOptionsItemSelected(MenuItem item) {
        // Handle action bar item clicks here. The action bar will
        // automatically handle clicks on the Home/Up button, so long
        // as you specify a parent activity in AndroidManifest.xml.
        int id = item.getItemId();
        if (id == R.id.action_settings) {
            return true;
        }
        return super.onOptionsItemSelected(item);
    }
```

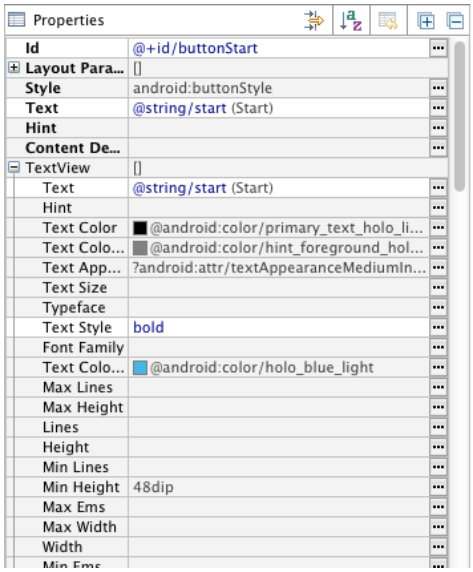
(a) MainActivity.java

```xml
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.CAS703.finalproject"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-sdk
        android:minSdkVersion="8"
        android:targetSdkVersion="19" />

    <application
        android:allowBackup="true"
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >
        <activity
            android:name="com.CAS703.finalproject.MainActivity"
            android:label="@string/app_name" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>

</manifest>
```

(b) Androidmanifest.xml

Figure 3: The code generated after creating the project

Secondly, ADT has defined the meta-data models of widgets such as textViews, buttons, pickerViews etc. We can see their properties and functions from the layout editor. After we finish design the interface, the editor has automatically generate the Android-specified XML code for the application. The Android-specified XML is the DSL of the Android application and the visible widgets can be treated as the meta-models. Figure 4 shows an example of the button meta-model and the Android-specified XML code generated by ADT.



(a) MainActivity.java

```
<Button
    android:id="@+id/buttonStart"
    android:layout_width="130dp"
    android:layout_height="wrap_content"
    android:layout_below="@+id/speed"
    android:layout_marginTop="28dp"
    android:layout_toLeftOf="@+id/speed"
    android:text="@string/start"
    android:textStyle="bold" />

<Button
    android:id="@+id/buttonReset"
    android:layout_width="130dp"
    android:layout_height="wrap_content"
    android:layout_alignBaseline="@+id/buttonStart"
    android:layout_alignBottom="@+id/buttonStart"
    android:layout_toRightOf="@+id/speed"
    android:text="@string/reset"
    android:textStyle="bold" />
```

(b) Androidmanifest.xml

Figure 4: The code generated by layout editor

Finally, when build the application, ADT transform the xml code into java language, then executes the java code on the Android runtime to generate the Android application apk. The BuildConfig.java and R.java is responsible for generating the xml to java. Figure 5 shows the two files.

Although ADT can automatically generate the code for us, we still need to complete the method by coding. The next section discusses the algorithms of how to calculate the walking speed.

2.4 Algorithms

To calculate the walking speed, the resource is based on the data detected by accelerate sensor embedded in the mobile. The main procedure of this application is shown in the Figure 6. It can be divided into five parts. When the app is started, the system first detect the step. When one step is detected, the system use the sensor data around this step to calculate the length of it. Whenever a step is detected, the step length added to the move length and the step number plus one. As the detector runs, the timer runs in the background to count the seconds. Every second the system refresh the walking speed. Here we will describe two algo-

```
/** Automatically generated file. DO NOT MODIFY */
package com.CAS703.finalproject;

public final class BuildConfig {
    public final static boolean DEBUG = true;
}
```

(a) MainActivity.java

```
/* AUTO-GENERATED FILE.  DO NOT MODIFY.
 *
 * This class was automatically generated by the
 * aapt tool from the resource data it found.  It
 * should not be modified by hand.
 */

package com.CAS703.finalproject;

public final class R {
    public static final class attr {
    }
    public static final class dimen {
        /**  Default screen margins, per the Android Design guidelines.

         Customize dimensions originally defined in res/values/dimens.xml (such as
         screen margins) for sw720dp devices (e.g. 10" tablets) in landscape here.

         */
        public static final int activity_horizontal_margin=0x7f040000;
        public static final int activity_vertical_margin=0x7f040001;
    }
    public static final class drawable {
        public static final int ic_launcher=0x7f020000;
    }
    public static final class id {
        public static final int action_settings=0x7f080008;
        public static final int buttonReset=0x7f080007;
        public static final int buttonStart=0x7f080006;
        public static final int moveLength=0x7f080003;
        public static final int speed=0x7f080005;
```

(b) Androidmanifest.xml

Figure 5: Generate xml to java

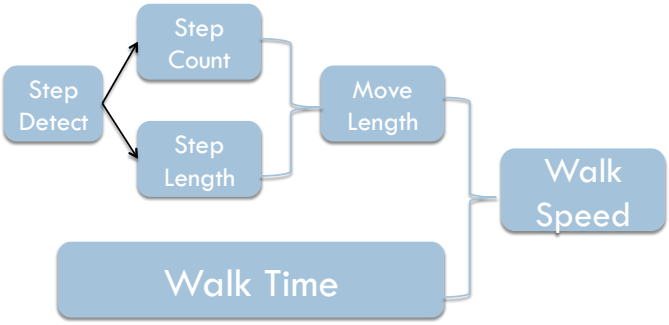


Figure 6: The procedure of Walking Speed Detector

rithms for step detection and step length calculation.

As for the step detection, it includes three main steps, collect data, handle data, analyze data. In collect data procedure, the data should be passed low-pass filter and high-pass filter to eliminate the gravity contribution and noise signal respectively. Then handle with the data by the following steps: norm the values of three axis, square the result to augment the single, integrate the data. At last the detector analyze the data to find the peak in the defined time interval and check it with threshold.

The algorithm of step length is similar to step detection, it also contain three steps: collect data, handle data, and calculate. The details of each sub steps are described in Figure 7. The calculate equation is the following:

$$SL = K \times \sqrt[4]{max - min} \quad K = 0.364$$

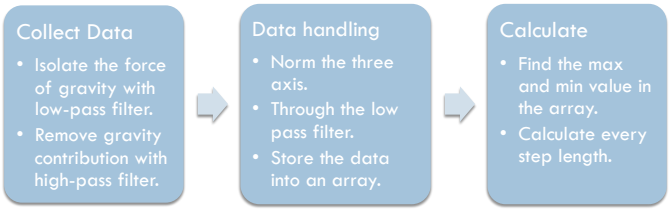


Figure 7: The procedure of Walking Speed Detector

2.5 Result

After we did all the jobs above, the project is finally completed. We connect the mobile phone with computer and Run&Build the code into an Android application. Here is the image of the application running on the phone.

Figure 8: The final Result

3. CONCLUSION

The number of mobile applications increases significantly in recent years due to smart phone platforms such as Android, iOS, Windows Phone. In order to improve the productivity, it is necessary to adopt MBSE in the mobile apps development. ADT is the Android development tool provided by Google. It is a MDE tool and using MBSE technologies to help developers to build Android applications. In this paper, we have described the procedure to build walk speed detector to show how ADT adopt MDE approaches in software development.

4. REFERENCES

[1] Model-driven engineering `http://en.wikipedia.org/wiki/Model-driven_engineering`

[2] Domain-specific language `http://en.wikipedia.org/wiki/Domain-specific_language`

[3] Domain model `http://en.wikipedia.org/wiki/Domain_model`

[4] Abilio G. Parada, Lisane B. de Brisolara ”A model driven approach for Android applications development”