Rapid Mobile Development: Build Rich, Sensor-Based Applications using a MDA approach

Mohamed LACHGAR and Abdelmounaïm ABDALI,

Laboratory of Applied Mathematics and Computer Science (LAMAI),
Faculty of Science and Technology (FSTG), Cadi Ayyad University,
Marrakech, Morocco

Summary
Mobile phones are increasingly playing a crucial role in our daily lives. Nowadays, most smartphones are equipped with various embedded sensors such as motion, environmental, and position sensors. Therefore, many functionalities in mobile applications need to use these sensors. In this work, we are motivated to provide a meta-model to show the different embedded sensors then generate mobile applications that use various features offered by these sensors. In order to achieve this, we are based on model driven architecture (MDA) proposed by Object Management Group (OMG). The MDA approach can help us to ensure the sustainability of expertise, the gain in productivity while dealing with the challenges of mobile platform fragmentation.

Key words:
Model Driven Architecture; Sensors; Domain Specific Language; Mobile development;

1. Introduction

The industry of mobile application development is increasing due to the intensive use of the latter in mobile devices. Most of these applications run on mobile operating systems such as Android, iOS and Windows Phone. These devices are equipped with a set of embedded sensors such as motion sensors (e.g. accelerometers, gyroscopes), environmental sensors (e.g. temperature, light) and position sensors (e.g. orientations, magnetometers, etc.) [1].

Fig. 1 Sensors in Mobile Phones [1]

These sensors are used by applications to support their new features, like Spirit Level in some applications related to the camera. However, the development of such applications requires more concerns such as code efficiency, interaction with devices, and rapid speed of flooding the market.

Due to the large variety of mobile technologies (e.g. Android, iOS, Windows Phone, etc.), developing the same application for this different platforms becomes an exhausting task. The model-driven engineering (MDE), a term proposed by [2] proposes to provide an effective solution to this problem. The MDE is a development approach that proposes to bring up the models in the rank of concept the first-class [3]. This is a form of generative engineering, which is characterized by a rigorous process from which everything is generated from a model. Thereby allowing puts the model status contemplative than productive.

The main purpose of this paper is the establishment of a meta-model to design a mobile application which makes use of the services provided by the on-board sensors in every mobile device. Then, using the MDA approach, we generate the native code targeting a specific mobile platform. Moreover, this approach will allow to generate the configuration files, declaration of objects, retrieval of data received over the sensor, and then forward them to targets such as embedded databases, web services, graphical interfaces, etc. This will allow developers to earn in terms of time, productivity, avoid programming errors and generate a code witch complies with coding standards.

This paper is organized as follows. The first section provides a brief description of embedded sensors followed by the MDA approach. Some related works are presented in the second section. The approach taken is described in the third part. The fourth section presents the proposed meta-model and different Template for code generation. The fifth section shows the applicability of our approach through an illustrating example. The last section concludes the paper and presents future work.

Manuscript received April 5, 2017
Manuscript revised April 20, 2017

2. Background

This background section is divided in four parts: the sensors in Android mobile devices, architecture of the Android sensor Framework, development stages under Android platform, and model-driven engineering.

2.1. The sensors in Android mobile devices

Android smartphones are equipped with an embedded sensors assembly. These sensors are used to monitor the motion of the equipment, position, or other surrounding environmental conditions. Android systems support many types of sensors [4, 5] (*see Table 1 for more details*).

Some sensors are hardware-based, which means that the sensor data is read directly from the physical components integrated into the smartphone. Other sensors are software-based, which means that the sensor data is read from one or more hardware sensors. The integrated sensors are widely used in third part applications. For example, a navigation application can use the magnetic field sensor to determine the scope of the compass.

Table 1: Sensor types supported by the Android platform

Sensor	***Type***	***Description***	***System Value***
Accelero-meter	Hardware	Measures the acceleration force	ACCELEROMETER
Gravity	Software or Hardware	Measure the force of gravity	GRAVITY
Gyroscope	Hardware	Measures a device's rate of rotation	GYROSCOPE
Light	Hardware	Measures the ambient light level	LIGHT
Orientation	Software	Measures degrees of rotation that a device makes around all three physical axes Measures	ORIENTATION
Pressure	Hardware	Measures the ambient air pressure	PRESSURE
Proximity	Hardware	Measures the proximity of an object relative to the view screen of a device	PROXIMITY
Temperature	Hardware	Measures the temperature of the device	TEMPERATURE
Ambient Temperature	Hardware	Measures the ambient room temperature	AMBIENT_TEMPERATURE
Linear Accelero-meter	Software or Hardware	Measures the acceleration force, excluding the force of gravity	LINEAR_ACCELERATION
Magnetic Field	Hardware	Measures the ambient geomagnetic fiel	MAGNETIC
Relative Humidity	Hardware	Measures the relative ambient humidity	RELATIVE_HUMIDITY
Rotation Vector	Software or Hardware	Measures the orientation of a device	ROTATION_VECTOR

2.2. Architecture of the Android sensor framework

The Android Sensor Framework architecture is shown in Figure 2. The main blocks of this architecture are:

- **Application framework:** Applications that use sensors use the "Application Framework" to get data from the devices. Communication begins in "sensor manager class" and then move to the lower layer by the sensor JNI (Java Native Interface).
- **Sensor libraries:** These libraries are intended to create a sophisticated interface to the upper layer. This is done by "the sensor class manager", "Service class sensor" and "sensor HAL".
- **Kernel:** in this layer we find the Linux device drivers created using the input subsystem, a generic Linux framework for all input devices such as mouse, keyboard, etc.

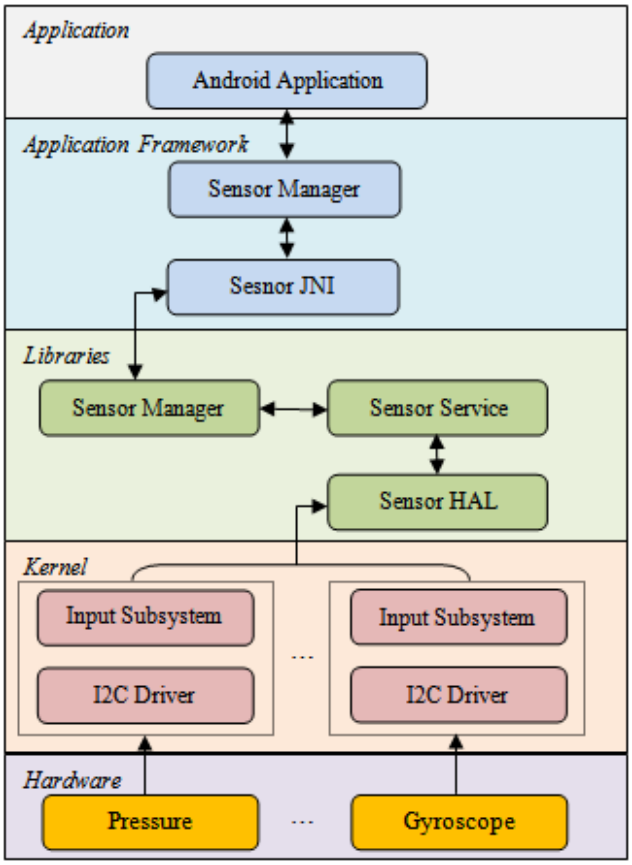

Fig. 2 Android sensor subsystem [4]

2.3. Development stages under the Android platform

The sensors integrated into the Android system are managed by the Android Sensor Framework. Unlike the camera, GPS and Bluetooth, which are protected by an

Android permission mechanism, embedded sensors can be directly used by applications without any requirement of permissions. With the help of Android Sensor Framework, an application can read the sensor data in the following steps.

- In the first place, creating the object of Sensor Manager Class. In this step, an application creates an instance of the sensor service. This class provides various methods to access the sensors.
- In the second place, creating the object of Sensor class by calling the getDefaultSensor() method of the Sensor Manager class. In this step, an application gets an object of the Sensor class with a specific type. The type of sensor can be specified in parameter of the method. In this example, the constant Sensor.TYPE PRESSURE describes an atmospheric pressure sensor [4].

  If the accelerometer sensor is needed, it can set the parameter of the method with the constant Sensor. TYPE_ACCELEROMETER.
- In the third place, instantiate an object of SensorEventListener interface and override two methods which are onAccuracyChanged () and onSensorChanged ().These two methods are used to receive sensor events when the accuracy of the sensor changes or when the sensor values change. The Android system will call onSensorChanged () when the data of the sensor switch automatically, and a sensor event object is placed in the method parameter.

  The sensor event object is created by the system. It contains the following information: the raw data of the sensor, the sensor type, the accuracy of data and time stamp for this event. Codes using sensor values can be written in this method.
- Finally, register the listener. By invoking the registerListener () method, the application saves a SensorEventListener in the system. The registered sensor is indicated by the second parameter of this method. When the value of a sensor changes, the system will notify the application that registered the sensor. The third parameter of the method is used to define the data delay. The data delay allows controlling the interval in which sensor events are sent to the application. In this example, the default data delay (SENSOR NORMAL DELAY) let this application receives the raw values of the sensor every 0.2 second.

Figure 3 shows the sample code to use a sensor on the Android platform.

2.4. Model-driven engineering

The results collected in recent years have shown the benefits of MDE compared to traditional development approach in terms of quality and productivity [6]:

- Quality: An overall reduction from 1, 2 to 4 times of the number of the anomalies and an improvement of the anomalies of 3 times in phase of maintenance of anomalies. The overall cost of the quality also fell due to decrease in time of inspection and test.

```
public class SensorActivity extends Activity implements SensorEventListener {
          private SensorManager sensorManager;
          private Sensor sensor;
          @Override
          public final void onCreate(Bundle savedInstanceState) {
                    super.onCreate(savedInstanceState);
                    setContentView(R.layout.main);
                    // Get an instance of the sensor service, and use that to get an instance of a particular sensor.
                    sensorManager = (SensorManager) getSystemService(Context.SENSOR_SERVICE);
                    sensor = sensorManager.getDefaultSensor(Sensor.TYPE_PRESSURE);
          }
          @Override
          public final void onAccuracyChanged(Sensor sensor, int accuracy) {
                    // Do something here if sensor accuracy changes.
          }
          @Override
          public final void onSensorChanged(SensorEvent event) {
                    float millibars_of_pressure = event.values[0];
                    // Do something with this sensor data.
          }
          @Override
          protected void onResume() {
                    // Register a listener for the sensor.
                    super.onResume();
                    sensorManager.registerListener(this, sensor,
                                        SensorManager.SENSOR_DELAY_NORMAL);
          }
          @Override
          protected void onPause() {
                    // Be sure to unregister the sensor when the activity pauses.
                    super.onPause();
                    sensorManager.unregisterListener(this);
          }
}
```

Fig. 3 Sample code to use a sensor

- Productivity: An improvement of the productivity from 2 to 8 times in term of lines of source code.

The MDA approach is proposed by the OMG [7] since 2001. This is a particular view of the Model Driven Development (MDD) [8]. The MDA (Model-Driven Architecture) approach offers significant benefits in controlling the development of computer applications and including productivity gains, increased reliability, significantly improvement of sustainability and greater agility dealing with changes. In order to clarify the concepts, the OMG has defined a number of terms around models namely meta-meta-model, meta-model, model, business model (CIM), functional model (PIM) which is independent from the technique and technical model (PSM) illustrated in Figure 4 and Figure 5.

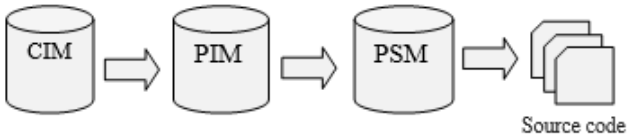

Fig 4 Key models in MDA

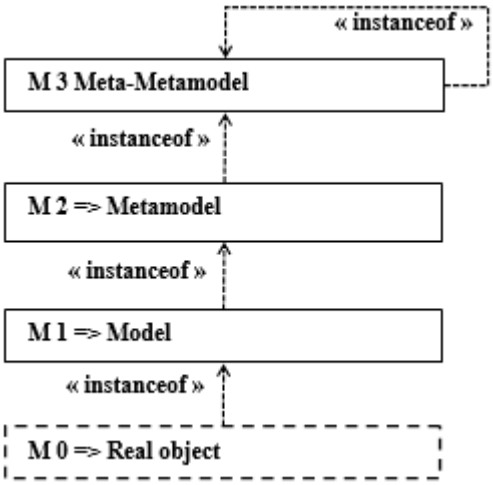

Fig. 5 Four-level Meta model Architecture

- A model, or terminal model, (M1) is a representation of a real object (in M0) conforming to a metamodel (M2),
- A metamodel (M2) is a representation of a set of modeling elements (in M1) conforming to a meta-metamodel (M3),
- A meta-metamodel (M3) is a set of modeling elements used to define metamodels (M2 & M1) conforming to itself.

Transformations between the different models are performed with tools compatible with the OMG standard called QVT (Query / View / Transformation) [9].

A model transformation is a process of converting a PIM, combined with other information, to produce a PSM. The MDA defines the following types of transformations based on the types of mappings:

- A transformation for a model type mapping is a process of converting a PIM to produce a PSM by following the mapping.
- A transformation for a model instance mapping is a process of converting a marked PIM to produce a PSM by following the mapping.

Figure 6 shows the application concepts of how one generally applies the MDA.

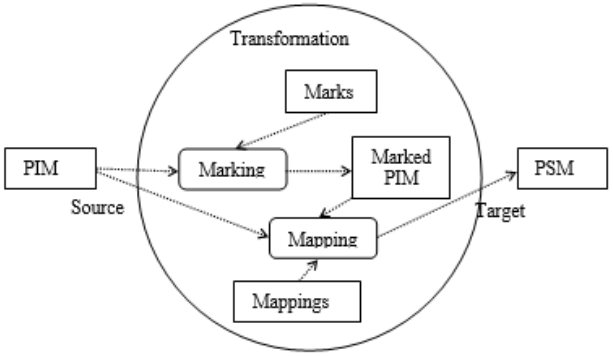

Fig. 6 Transformation concepts of the MDA [8]

A mapping is a determination (or transformation specification), including rules and other data, for transforming a PIM to deliver a PSM for a particular platform.

- A model type mapping indicates a mapping based on the types of model elements. It indicates mapping rules for how diverse types of elements in the PIM are converted to various types of elements in the PSM [10].
- A model instance mapping determines how particular model elements are to be converted in a specific way using marks. PIM elements are marked to show how they are to be changed. A mark from a PSM is applied to a PIM element to indicate how that element is to be transformed. A PIM element may likewise be marked several times with marks from various mappings and is, therefore, transformed according to each of the mappings. A PIM is marked to form a marked PIM that is then transformed to a PSM [10].

3. Related Works

The variety and number of mobile applications has increased due to the popularity of smartphones and app stores. Despite this growth, there is still a limited number of

applications that use embedded sensing devices that are available on different mobile platforms. The MDA approach aims are to provide application porting tools to adapt their code to different platforms. Many proposals of methods for the generation of native mobile applications according to the MDA approach have emerged. However, most of these methods allow the modeling and generation of graphical user interfaces and certain portions of code [13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24], without providing a mechanism for exploiting the native capabilities of a smartphone such as cameras, embedded sensors, etc.

The following table 'Table 2' shows a brief comparison of some cited approaches.

In this paper, we propose a metamodel for modeling applications that provide access to data from the embedded sensors and transmit them to various targets such as embedded databases, files, web services, etc.

This proposal will enhance the work presented previously by allowing them to also support the development of applications based on embedded sensors.

4. Android Sensor Code generetor

In this approach, we propose a metamodel for designing a mobile application based on embedded sensors. Then, M2M transformations (Model model) and M2T (Model to Text) are applied to generate the code targeting a specific platform. To do this, we opted for the Xtext framework [25, 26] to implement the Meta model and Xtend language [26, 27] to perform different transformations. Figure 7 shows the different stages that characterize our approach.

Table 2: Comparisons between some Approaches for Code Generation in Mobile Platforms

Approach	*Input Platforms*	*Output platforms*	*Based Modeling*	*Mapping*	*Design Type*	*Degree of automation*	*Support embedded sensors*	*Year*
Vaupel Approach[13]	Abstract Syntax	Native code for Android and iOs	DSL (Xtext)	Xtend	Textual	Automatic	Not supported	2016
Lachgar Approach [14, 15]	Abstract Syntax	Native code for Android, iOs and Windows phone	DSL (Xtext)	Xtend	Textual	Automatic	Partially	2016
Thanaseth Approach [16]	UML Model	Native code for Android	UML (Navigation Diagram)	-	Graphical	Automatic	Not supported	2016
Just Modeling [17]	UML Model	Java code with annotation	UML	Acceleo	Graphical	Automatic	Not supported	2016
Benouda Approach [18, 19]	UML Model	GUI and class Java files for Android platform	UML	QVT, Acceleo	Graphical	Automatic	Not supported	2016
Koji Approach [20]	UML Model	Native code	UML Diagrams, GUI Builder	-	Graphical	Automatic	Not supported	2014
Diep Approach [21]	GUI graphic Model	GUI Native Code	Graphical GUI	DOM	Graphical	Automatic	Not supported	2013
Sabraoui Approach [22]	UML Model	GUI for mobile platforms and class structure	UML	ATL, DOM, Xpand	Graphical	Automatic	Not supported	2013
MD2 [23]	Abstract Syntax	Native Code for Android and iOs	DSL (Xtext)	Xpand	Textual	Automatic	Partially	2013
Minhyuk Approach[24]	UML Model	Android Models	UML Profiles	-	Graphical	Manual	Partially	2012

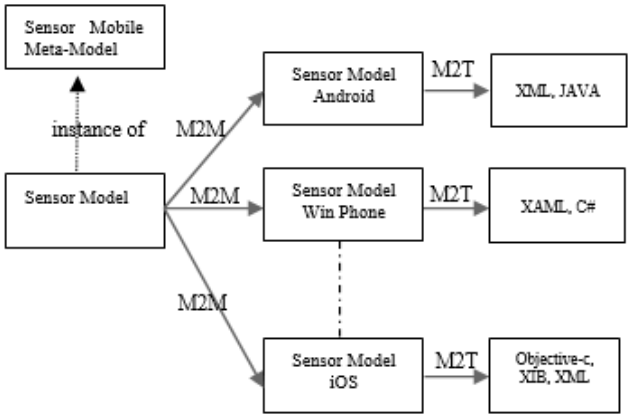

Fig. 7 Proposed architecture for code generation from a platform independent model [14]

After generating code using a set of template, the user can also add code snippets to enhance the application. Thus, the generator allows substantial savings of time and generates a code in accordance with the coding standards (*see Figure 8 for more details*).

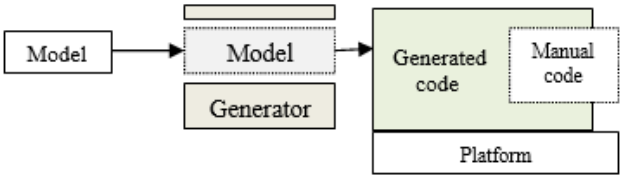

Fig. 8 Model to Text transformation (M2T)

5. Mobile Sensor MetaModel

A mobile application can use many sensors, each sensor sends data in a specific time interval. These data will be stored in collections, files, data bases or sent to web services etc. thereafter; they will be used in applications such as labyrinth, blood pressure, accelerometer analyzer, room temperature, etc.

The metamodel proposed to model a mobile application making use of embedded sensors is presented in the figure 11.

An extract from the textual description of metamodel is shown below (*see Figure 9 for more details*):

```
grammar ma.ucam.mobile.sensor.SensorDsl with
        org.eclipse.xtext.common.Terminals
generate sensorDsl
"http://www.ucam.ma/mobile/sensor/SensorDsl"
MobileApp returns MobileApp:
          {MobileApp}
          'MobileApp'
          name=STRING
          '{'
             ('version' version=STRING)?
             ('packageName' packageName=STRING)?
             ('minSdk' minSdk=INT)?
             ('sensors' '(' sensors+=[Sensor|STRING] ( ","
                sensors+=[Sensor|STRING])* ')' )?
          '}';
Sensor returns Sensor:
          Accelerometer | Gravity | Gyroscope | Light |
          Orientation | Pressure | Proximity;
Accelerometer returns Accelerometer:
          {Accelerometer}
          'Accelerometer'
          '{'
             ('output' output=INT)?
             ('accuracy' accuracy=[accuracy|STRING])?
             ('delay' delay=[Delay|STRING])?
             ('targets' '(' targets+=[Target|STRING] ( ","
                targets+=[Target|STRING])* ')' )?
          '}';
accuracy returns accuracy:
          {accuracy}
          'accuracy'
          '{'
               ('accuracy' accuracy=Statue)?
          '}';
Delay returns Delay:
          {Delay}
          'Delay'
          '{'
               ('delay' delay=Value)?
          '}';
Target returns Target:
           File | DataBase | WebService | UI | Collection;
WebService returns WebService:
          {WebService}
          'WebService'
          '{'
               ('url' url=STRING)?
               ('methodName' methodName=STRING)?
               ('soapAction' soapAction=STRING)?
               ('nameSpace' nameSpace=STRING)?
          '}';
…
```

Fig. 9 Extract from the textual description of metamodel

The graph below illustrates a part of the grammar presented in Figure 9. This graph shows one part of the rules used.

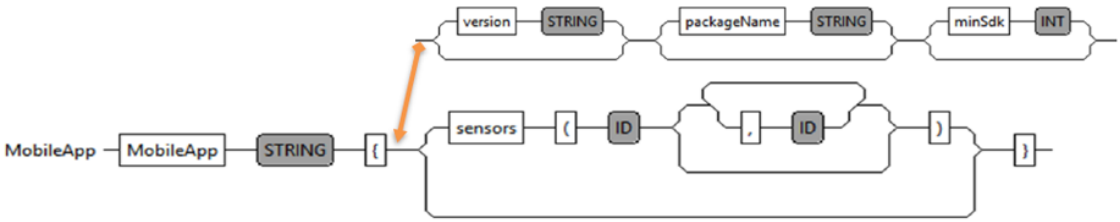

Fig. 10 Extract of graph description of metamodel

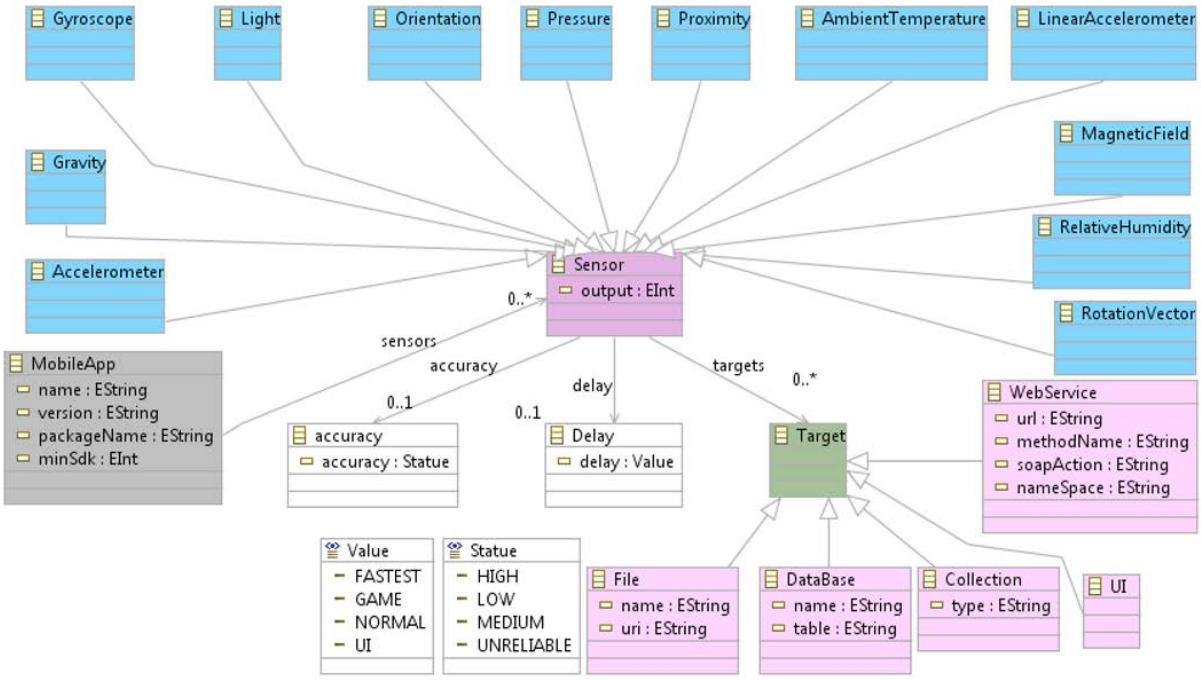

Fig. 11 Mobile phone sensor metamodel

6. Android Sensor Template

In this section, we will discuss the different template used in code generation, to build a mobile application for Android, which makes use of embedded sensors.

To do this you must follow three steps:

Step 1: Declare the sensors in the AndroidManifest configuration file. This allows Google Play filtering applications compatible with the user's device (Manifest Template).

Step 2: Collect the values through the SensorEvent class. All data is stored in an array, whose size depends on the type of sensor used (Activity Template).

Step 3: Send the recovered data to designated targets, and this in a separate thread.

In the following sub-sections, we present the different proposed Template.

6.1. Manifest Template

Each sensor must be specified in a separate tag. A snippet of the AndroidManifest.xml for the example app is shown here:

```
<uses-feature
android:name="android.hardware.sensor.light"
android:required="true" />
```

Below is the proposed Template for Manifest.xml configuration file generation to support the use of embedded sensors (*see Figure 12 for more details*).

```
def genManifest(EList<Sensor> sensors) '''
    <manifest xmlns:android="http://schemas.android.com/apk/res/android"
      package="«packageName»" android:versionCode="1"
      android:versionName="«version»">
      <uses-sdkandroid:minSdkVersion="«minSdk»" />
        «FOR s : sensors»
         <uses-featureandroid:name="android.hardware.sensor.«s.types.toString.toLowerCase»" android:required="true" />
          «IF (s.types.equals("Gravity") || s.types.equals("Linear_Accelerometer"))
                                        && !sensors.contains("Accelerometer")»
                <uses-featureandroid:name="android.hardware.sensor.accelerometer" android:required="true" />
          «ENDIF»
        «ENDFOR»
     </manifest>
'''
```

Fig. 12 Manifest file Template

6.2. Activity Class Template

As stated in Figure 3, in order to implement SensorEventListener, The Activity class provides concrete implementations for onAccuracyChanged(), and onSensorChanged(). Both methods update the display whenever a sensor reports new data or its accuracy changes.

An extract of Template used to generate the Activity class is presented below (*see Figure 13 for more details*):

```
def genActivity(EList<Sensor> sensors)'''
public class SensorActivity extends Activity implements
        SensorEventListener {
      private SensorManager sensorManager;
        «genOnCreate(sensors)»
        «genOnAccuracy(sensors)»
        «genOnSensorChanged(sensors)»
        …
}
'''
```

Fig. 13 Extract of Activity Class

The template proposed for generating the onSensorChanged method is presented below. The latter allows recovering data according to the specified sensor and the number of desired data. The sendData method allows to send the recovered data towards the target mentionned in Target Data (*see Figure 14 for more details*).

```
def genOnSensorChanged(EList<Sensor> sensors)'''
public void onSensorChanged(SensorEvent event) {
 onAccuracyChanged(event.sensor, event.accuracy);
 switch (event.sensor.getType()) {
     «FOR s : sensors»
     case Sensor.TYPE_«getClass(s).toUpperCase»:
         sendData("«getClass(s)»",
         «FOR i : 0 ..< s.output»
                  event.values[«i»]
                  «IF i != s.output - 1»,«ENDIF»
         «ENDFOR»);
     break;
     «ENDFOR»
 }
}
'''
```

Fig. 14 onSensorChanged method template

7. Illustrating Example: Magnetometer Metal Detector

Magnetometer metal detector is a tool to detect the electromagnetic field around a smartphone.

This tool allows the metal detection using the integrated magnetic sensor in the smartphone. The intensity of the magnetic field in the wild is about 49µT (490mG, 1µT = 10mG). If there is no metal in the zone, the intensity of the magnetic field must increase. It can be helpful to find pipes and electrical lines in the walls or the metal in the ground.

7.1.Analysis and model creation

The application's model is presented below (*see Figure 15 for more details*).

```
MobileApp "MobileApp"{

        version "1.0"
        packageName "ma.uca.mobile"
        minSdk 4
        sensors(
                MagneticField {
                        output 3
                        Accuracy {LOW}
                        Delay{GAME}
                        Data {UI}
        )
        Screen title "Detector" orientation "portrait"
        {
                Layout [width "match" orientation
                   "horizontal" column "1"  weight
                   "match" Type "Linear"]{
                        Label [id 1 text
                        "MAGNETOMETER METAL DETECTOR"
                        gravity "center"],
                        Label [id 2 text "xTV"],
                        Label [id 3 text "yTV"],
                        Label [id 4 text "zTV"],
                        ProgressBar [id 1 max 100],
                        Dialog [id 1 type "info" text
                        "Metal detected"]
                }
        }

}
```

Fig. 15 The application's model

7.2. The generated code

Code snippet generated for the Activity class (*see Figure 16 for more details*):

```
public void onSensorChanged(SensorEvent event) {
        onAccuracyChanged(event.sensor, event.accuracy);
        switch (event.sensor.getType()) {
            case Sensor. TYPE_MAGNETICFIELD:
                sendData("MagneticField",
                        event.values[0],
                        event.values[1],
                        event.values[2]
                );
                break;
        }
}
```

Fig. 16 Code Snippet generated for the Activity Class

The configuration file generated from our model (*see Figure 17 for more details*):

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
   package="ma.uca.mobile" android:versionCode="1"
   android:versionName="1.0">
   <uses-sdkandroid:minSdkVersion="4" />
      <uses-featureandroid:name="android.hardware.sensor.magneticfield" android:required="true" />
</manifest>
```

Fig. 17 Manifest.xml configuration file generated

7.3. The graphical interface generated for the Android platform

The figure below shows a graphical interface generated, which displays the data retrieved by the embedded sensor in the text fields. A test code on values is added to the program in order to detect the magnetic field (*see Figure 18 for more details*).

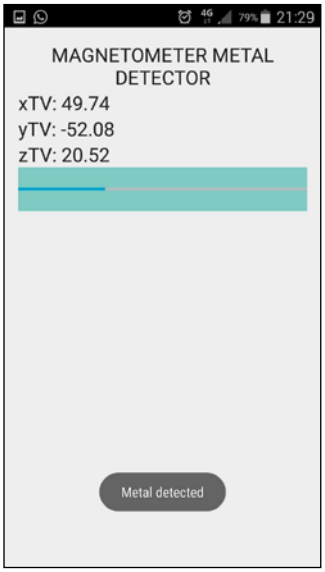

Fig. 18 GUI generated for Magnetometer metal detector

8. Conclusion

To conclude, in this paper, we have presented a study on the embedded sensors in mobile devices, and we have adopted the MDA approach for modeling and automatically generate applications that make use of the data retrieved via these sensors, based on a model consistent with the metamodel proposed.

This approach will be generalized for all mobile platforms (e.g. iOS, Windows Mobile, etc.). To implement our approach, we opted for Xtext language to realize the metamodel and Xtend for the implementation of the various transformations and Templates. The proposed matamodel comes to enhance the work already presented in [15], In fact, this latter allow to model mobile applications that make use of embedded sensors with the ability to transmit collected data to various targets, such as web services, embedded databases, files, or graphical interfaces.

For future improvements, we will study the possibility to extend our method to deal with data recovering from external sensors; we also want to design a software layer to ensure data's security access, and the integrity of recovered data.

References

[1] C. Perera, A. Zaslavsky, P. Christen, A. Salehi, & D. Georgakopoulos, "Capturing sensor data from mobile phones using global sensor network middleware". In 23rd International Symposium on Personal, Indoor and Mobile Radio Communications-(PIMRC), 2012, pp. 24-29. IEEE.

[2] S. Kent, "Model driven engineering. Integrated Formal Methods (IFM)", 2002 Turku (Finland) Springer, p. 286-298.

[3] J. Bézivin. In : search of a basic principle for model driven engineering. Novatica Journal, Special Issue, 5, pp. 21-24. 2004.

[4] Android, "Sensors Overview", https://developer.android.com/guide/topics/sensors/sensors_overview.html, 2016-05-29.

[5] Y. Zhi-An, & M. Chun-Miao, "The development and application of sensor based on android". In : 8th International Conference on Information Science and Digital Content Technology (ICIDT), 2012. IEEE, 2012. p. 231-234.

[6] M. El Hamlaoui, "Mise en correspondance et gestion de la cohérence de modèles hétérogènes évolutifs". Thèse de doctorat. Université Toulouse le Mirail-Toulouse II, 2015.

[7] OMG. 1989. Object Management Group [Online]. Available: http://www.omg.org [Accessed April 2016].

[8] B. Hailpern, & P. Tarr, "Model-driven development: The good, the bad, and the ugly", IBM systems journal, 45, p. 451-461. 2006

[9] Object Management Group, "Meta Object Facility (MOF) 2.0 Query/View/Transformation Specification", Version 1.1, January 2011, http://www.omg.org/spec/QVT/1.1/..

[10] S. Alkhir, "Understanding the model driven architecture". published in Methods & Tools, October 2003.

[11] J. A. Monte-Mor, E. O. Ferreira, H. F. Campos, A. M. da Cunha, & L. A. V. Dias, "Applying MDA Approach to Create Graphical User Interfaces", Eighth International Conference on Information Technology: New Generations, Las Vegas, NV, IEEE, 11-13 April 2011, pp. 766-771.

[12] S. Link, T.Schuster, P. Hoyer, & S. Abeck, "Focusing Graphical User Interfaces in Model-Driven Software Development", First International Conference on Advances

in Computer-Human Interaction", Sainte Luce, 10-15 Feb. 2008, pp. 3-8.
[13] S. Vaupel, G. Taentzer, R. Gerlach, & M. Guckert, "Model-driven development of mobile applications for Android and iOS supporting role-based app variability". Software & Systems Modeling, 2016, pp. 1-29.
[14] M. Lachgar, & A. Abdali, "Generating Android graphical user interfaces using an MDA approach". In: Third IEEE International Colloquium in Information Science and Technology (CIST), 2014. IEEE, 2014, pp. 80-85.
[15] M. Lachgar, & A. Abdali, "Modeling and Generating Native Code for Cross-Platform Mobile Applications Using DSL", Intelligent Automation & Soft Computing, 2016, pp. 1-14.
[16] C. Thanaseth, & L. Yachai, "Model Driven Development of Android Application Prototypes from Windows Navigation Diagrams". In : 2016 International Conference on Software Networking (ICSN). IEEE, 2016, pp. 1-4.
[17] F. Freitas, & P. Maia. "JustModeling: An MDE Approach to Develop Android Business Applications". In : Computing Systems Engineering (SBESC), 2016 VI Brazilian Symposium on. IEEE, 2016, pp. 48-55.
[18] H. Benouda, M. Azizi, R. Esbai , & M. Moussaoui, "MDA Approach to Automate Code Generat ion for Mobile Applications". In : Mobile and Wireless Technologies 2016. Springer Singapore, 2016, pp. 241-250.
[19] H. Benouda, M. Azizi, R. Esbai , & M. Moussaoui, "Code generation approach for mobile application using acceleo". International Review on Computers and Software (IRECOS), 2016, vol. 11, no 2, pp. 160-166.
[20] M. Koji, & M. Saeko, "MDD for Smartphone Application with Smartphone Feature Specific Model and GUI Builder". In: The 9th International Conference on Software Engineering Advances, 2014, pp. 64-68, ISBN: 978-1-61208-367-4.
[21] C. K. Diep, Q. N. Tran, & M. T. Tran. "Online model-driven IDE to design GUIs for cross-platform mobile applications". In Proceedings of the Fourth Symposium on Information and Communication Technology, 2013, pp. 294-300, ACM.
[22] A. Sabraoui, M. El Koutbi, & I. Khriss, "A MDA-Based Model-Driven Approach to Generate GUI for Mobile Applications". International Review on Computers and Software Journal (IRECOS), 2013, vol. 8, no 3, pp. 845-852.
[23] H. Heitkötter, T. A. Majchrzak, & H. Kuchen, "Cross-platform model-driven development of mobile applications with md 2". In Proceedings of the 28th Annual ACM Symposium on Applied Computing , 2013, pp. 526-533, ACM.
[24] K. Minhyuk, S. Yong-Jin, M. Bup-Ki, K. Seunghak, & S. K. Hyeon, "Extending UML meta-model for android application". In 11th International Conference on Computer and Information Science (ICIS), 2012 IEEE/ACIS, pp. 669-674. IEEE.
[25] The Eclipse Foundatio. Xtext. Retrieved from http://www.eclipse.org/Xtext/, 2013.
[26] L. Bettini, "Implementing domain-specific languages with Xtext and Xtend". Packt Publishing Ltd, 2016.
[27] The Eclipse Foundation. Xtend. Retrieved from http://www.eclipse.org/xtend/, 2013.

Lachgar Mohamed received his Diploma in Computer Engineering from National School of Computer Science and System Analysis (ENSIAS), University Mohammed V Souissi, Rabat, Morocco in 2009. He is currently a PhD student in Cadi Ayyad University. His research interests are in the areas of automation tools development in embedded software, software modeling & design, metamodel design, model transformation, and model verification & validation method.

Abdelmounaim Abdali, Ph.D. in Solid Mechanics and Structures in University of Amiens in 1996, France, He is a professor in Computer Science at the University Cadi Ayyad, Faculty of Sciences and Technics, Marrakech, Morocco, Member at the Laboratory of Applied Mathematics and Computer Science (LAMAI) Marrakech, Morocco. His Research interests are: computer science, DTN network, business intelligence, design and implementation in data warehouse, ubiquitous systems, modeling & design, metamodel design, model transformation, and model verification & validation method, numerical simulation, biomechanics, bone remodeling and damage.