MOBILE PHONE SENSORS META-MODEL

Mohamed LACHGAR[1,*], Khalid LAMHADDAB[2], Abdelmounaim ABDALI[1]

[1, 2]Cadi Ayyad University Marrakesh, Morocco
[1]LAMAI Laboratory, FSTG
[2]TIM Laboratory, ENSA

ABSTRACT

In the last decade, the use of wireless electronic communication technology, such as mobile phones, is fundamental to the private and professional lives of most citizens. In fact, it has become an inseparable part of their daily lives. Nowadays, most cell phones are provided with different implanted sensors, which measure motion, orientation, and environmental conditions such as ambient light or temperature. Therefore, several functionalities in mobile applications need to use these sensors, as in the case of logistics applications, social network applications or travel information applications. Hence, the primary contribution of this work is to establish a generic meta-model, in order to show the different embedded sensors in the smartphones, and then generate mobile applications that use various features offered by these sensors for the case of Android OS. So as to achieve this, our approach is based on a model driven architecture (MDA) suggested by Object Management Group (OMG), which is a variant of Model-Driven Engineering (MDE). The MDA approach can contribute in the insurance of the sustainability of expertise, as well as the improvement of the gain in productivity while dealing with the challenges of mobile platform fragmentation.

KEYWORDS

Model Driven Architecture, Embedded Sensors, Domain Specific Language, Mobile development, Android

1. INTRODUCTION

Recently, the industry of mobile application development is increasing due to the strong use of smartphones that have become nowadays more accessible. Most of these applications run on mobile operating systems such as Android, iOS and Windows. 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.) (see Fig. 1 for more details) [1].

Applications use these sensors to support their new features, like Spirit Level in some applications related to the camera. These sensors can measured and collected various data. The most common measured data are temperature, humidity, pressure, acceleration (vibration), light, infrared magnetic fields, sound, radiation, location (GPS), mechanical stress and chemical composition. As a consequence of extensive possibilities, the envisaged applications for the embedded sensors are multiple. The most typical areas of application that use these sensors are: traffic control, security, military, health care, industrial sensing, home automation and environmental monitoring.

Figure 1. Sensors in Mobile Phones [1]

The type of collected data and the nature of processing depend on the application. Despite the variety of applications and domains, four essential tasks that are independent of application domain can be listed as follows:

- **Event Detection:** Detection of the occurrence of events of interest and their parameters.
- **Object classification**: Identification of an object or event. Broadly, this task involves the combination of data from many sources and a collaborative processing to obtain the result.
- **Object tracking**: Tracking the movements and position of a mobile object within the coverage area of the network.
- **Monitoring:** Detection of the value of a parameter in a given location or the coverage area of the network. Classically, the task is completed using periodic measurements.

However, the development of such applications is constrained by several concerns, such as: code efficiency, interaction with devices, and enhanced competitiveness at the application stores.

Due to the large variety of mobile technologies (e.g. Android, iOS, Windows Phone, etc.), developing the same application for these different platforms become an exhausting task. The model-driven engineering (MDE), a term proposed by Kent [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 put the model status contemplative than productive.

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, jointly with a comparative study. The adopted approach 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 the proposed approach through an illustrating example. The

sixth section concludes the paper. The last section sheds the light on some perspectives and future work.

2. BACKGROUND

This background section is divided into four parts: (1) the sensors in Android mobile devices, (2) the architecture of the Android sensor Framework, and (3) the 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 below for more details).

Table 1. Sensor types provided by the Android platform [5]

Sensor	**Type**	**Description**	**Common Uses**	**Unit of measure**
ACCELEROMETER	Hardware	Get the acceleration force that is applied to a device on all three physical axes (x, y, and z), including the force of gravity.	Motion detection (tilt, shake, etc.).	m/s^2
AMBIENT TEMPERATURE	Hardware	Get the ambient room temperature.	Monitoring air temperatures.	°C
GRAVITY	Hardware or Software	Get the force of gravity that is applied to a device on all three physical axes (x, y, and z).	Motion detection (shake, tilt, etc.).	m/s^2
GYROSCOPE	Hardware	Get a device's rate of rotation around each of the three physical axes (x, y, and z).	Rotation detection (turn, spin, etc.).	rad/s
LIGHT	Hardware	Measures the light illuminance.	Controlling screen brightness.	lx
LINEAR ACCELERATION	Hardware or Software	Get the acceleration force that is applied to a device on all three physical axes (x, y, and z), excluding the force of gravity.	Observing acceleration along a single axis.	m/s^2
MAGNETIC FIELD	Hardware	Get the ambient geomagnetic field for all three physical axes (x, y, and z).	Constructing a compass.	μT
ORIENTATION	Software	Get degrees of rotation that a device makes around all three physical axes (x, y, and z).	Determining device position.	Degrees
PRESSURE	Hardware	Get the ambient air pressure.	Observing air pressure changes.	hPa or mbar
PROXIMITY	Hardware	Get the proximity of an object in cm relative to the view screen of a device. This sensor is typically used to determine whether a handset is being held up to a person's ear.	Phone location during a call.	cm
RELATIVE HUMIDITY	Hardware	Get the relative ambient air humidity.	Observing dew-point, absolute, and relative humidity.	%
ROTATION VECTOR	Hardware or Software	Get the orientation of a device by providing the three elements of the device's rotation vector.	Motion detection and rotation detection.	Unitless
TEMPERATURE	Hardware	Get the temperature of the device.	Observing temperatures.	°C

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.

2.2 Architecture of the Android Sensor Framework

The main blocks of the Android sensor framework architecture are:

- **SDK**: Applications access sensors via the Sensors SDK (Software Development Kit) API. The SDK provides functions so as to list available sensors and to register to a sensor.

- **Framework:** The framework allows linking the several applications to the Hardware Abstraction Layer (HAL). The HAL itself is single-client. Without this multiplexing happening at the framework level, only a single application could access each sensor at any given time.

- **HAL**: The Sensors Hardware Abstraction Layer (HAL) API is the interface between the hardware drivers and the Android framework. It consists of one HAL interface *sensors.h* and one HAL implementation we refer to as *sensors.cpp*.

- **Kernel driver**: The sensor drivers interrelate with the physical devices. Sometimes, the HAL implementation and the drivers constitute one software entity. In some cases, the hardware integrator needs sensor chip manufacturers in order to produce the drivers. However, these latter's are the HAL implementers. In all cases, HAL implementation and kernel drivers are the responsibility of the hardware manufacturers, and Android does not provide preferred approaches to write them.

- **Sensor hub**: Sensor hub is useful to perform some low-level computation at low power while the SoC can be in a suspend mode. Some sensor hubs contain a microcontroller for generic computation, and hardware accelerators to enable very low power computation for low power sensors.

- **Sensors:** Those are the physical MEMs chips making the measurements. In many cases, several physical sensors are present on the same chip. For example, some chips include an accelerometer, a gyroscope and a magnetometer. (Such chips are often called 9-axis chips, as each sensor provides data over 3 axes.). A few of those chips also contain some logic to achieve common computations such as step detection, motion detection and 9-axis sensor fusion.

The Android Sensor Framework architecture is shown in Fig. 2 below.

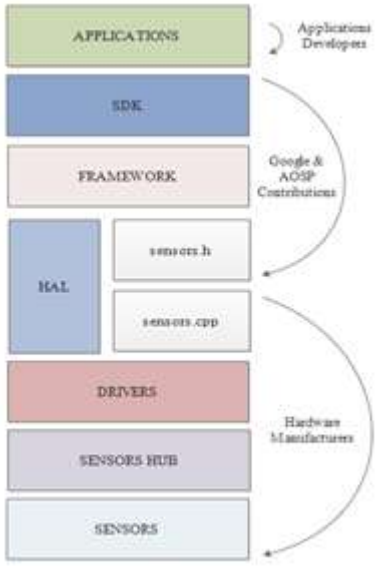

Figure 2. Layers of the Android sensor stack and their respective owners [6]

2.3 Model-driven engineering

Recently, the results collected have shown the benefits of MDE compared to the traditional development approach in terms of quality and productivity [8]:

- **Quality**: An overall reduction from 1, 2 to 4 times of the anomalies/bugs number, therefore, a significant gain during the application maintenance phase.

- **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 [9] since 2001. This is a particular view of the Model Driven Development (MDD) [10]. 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 Fig. 3 and Fig. 4.

- A model, or terminal model, (M1) is a representation of a real object (in M0) conforming to a meta-model (M2),

- A meta-model (M2) is a representation of a set of modeling elements (in M1) conforming to a meta-meta-model (M3),

- A meta-meta-model (M3) is a set of modeling elements used to define meta-models (M2 & M1) conforming to itself.

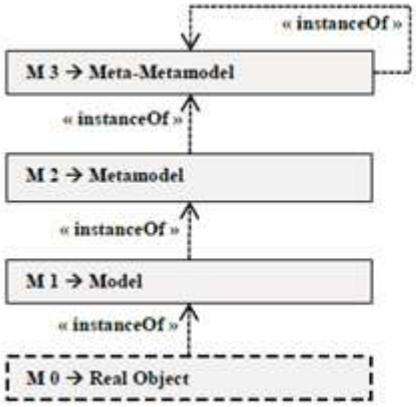

Figure 3. Four-level Meta model Architecture

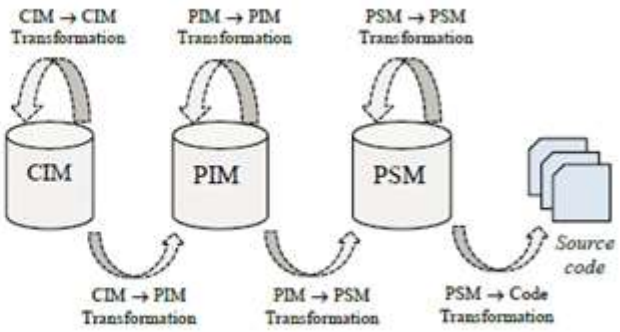

Figure 4. Fundamental models in MDA

MDA supports the development of the following three types of models:

- **Computation Independent Model (CIM)**: this model represents the highest level of abstraction. It describes the system requirements and the environment in which it will operate, while the details of the software structure and realization are hidden or not yet determined.

- **Platform Independent Model (PIM)**: this model describes the details of the system, but does not show details of the use of its platform or of a particular technology.

- **Platform Specific Model (PSM)**: this model describes the details and features absent from the PIM. It must be adapted to specify the implementation of the system in a single technology platform.

As these different types of models represent different levels of abstraction of the same system, MDA recommends the use of transformation mechanisms allowing transformations and refining models to other models (CIM to PIM, PIM to PSM, etc.).

Starting from a model of business concepts and transforming this model into other models gradually refining to finally arrive at a model of source code (see Fig.4 for more details) [11].

Transformations between the various models are conceived with tools that are compatible with the OMG standard called QVT (Query / View / Transformation) [12].

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

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

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

Fig. 5 shows the application concepts of how one generally applies the MDA.

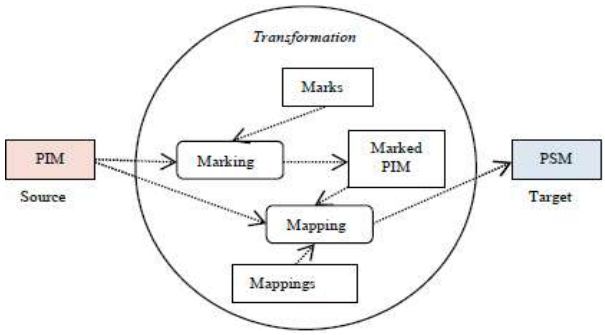

Figure 5.Transformation concepts of the MDA [11]

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 [13].

- 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 [13].

3. RELATED WORKS

Few years ago, the variety of the numerous mobile applications has increased due to the popularity of smartphones and app-stores. Despite this growth, there are still a limited number of applications, which use embedded sensing devices that are available on different mobile platforms. The MDA approach aims 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 (see table 2 and table 3 below for more details).

Table 2. Approaches for modeling and code generation of mobile applications

Work	***Title***	***Year***	***Description***
W 1	JustModeling: An MDE Approach to Develop Android Business Applications [14]	2016	JustModeling, an MDE approach formed by JBModel, that is a graphical modeling tool by which the user models the application business classes using the UML class diagram and affords a set of model transformations to generate the code for the JustBusiness framework, which automatically generates all required resources of the mobile application.
W 2	Model-driven development of mobile applications for Android and iOS supporting role-based app variability [15]	2016	A modeling language and an infrastructure for the model driven development of native apps in iOS and Android. This approach allows a flexible app development on diverse abstraction levels: compact modeling of standard app elements, such as standard data management and increasingly detailed modeling of individual elements to cover specific behavior. Moreover, a kind of variability modeling is supported, such that mobile apps with variants can be developed.

W 3	Modeling and generating native code for cross-platform mobile applications using DSL [16]	2016	Approach for modeling and generation of multiplatform mobile applications (e.g. Android, iOS, Windows Phone). Proposing a platform-independent meta-model to monetize a mobile application in textual format, and defining a set of transformation rules and templates for native code generation.
W 4	Model Driven Development of Android Application Prototypes from Windows Navigation Diagrams [17]	2016	Approach allows automating the generation of android application prototyping from windows navigation diagrams.
W 5	Code Generation Approach for Mobile Application Using Acceleo [18]	2016	Methodology based on the MDA approach to produce mobile applications according to the principal 'Develop Once, Run Everywhere'. This approach exploits UML class diagram modeling and use the Acceleo tool to generate a specific code in order to accelerate and facilitate the development of mobile apps.
W 6	Modelling and generating the user interface of mobile devices and web development with DSL [19]	2015	Approach to developing the user interface for mobile applications, applied to Android and Java Server Faces Framework. A language for the development of graphical interfaces, DSL Technology neutral, with the intention of generating native code for several defined platforms.
W 7	Model-Driven Development Based on OMG's IFML with WebRatio Web and Mobile Platform [20]	2015	WebRatio proposal platform for the development of CDEM web and mobile applications based on IFML.
W 8	A GUI Modeling Language for Mobile Applications [21]	2015	Method for modeling mobile interfaces, as part of a future project development for mobile applications MDD. The language is presented Developed: MIM (Mobile Interface Modeling), and the feasibility of implementing the proposal is evaluated
W 9	Model-Driven Cross-Platform Apps Towards Business Practicability [22]	2015	MD2 as a solution to meet the typical needs of enterprise applications.
W 10	Generating Android graphical user interfaces using an MDA approach [23]	2014	Approach to developing the user interface for mobile applications, applied to Android. A language for the development of graphical interfaces, DSL Technology neutral, with the intention of generating native code for several defined platforms.
W 11	A Model-Driven Approach to Generate Mobile Applications for Multiple Platforms [24]	2014	UML profile for the mobile development platform. The goal is to generate code and business logic for different platforms. Prototype development tool to generate code called MAG (mobile application generator).
W 12	A MDA-Based Model-Driven Approach to Generate GUI for Mobile Applications [25]	2013	A model-driven approach to modeling the graphical interface of a mobile application. It presents a design of forms using UML

However, most of these methods allow the modeling and generation of graphical user interfaces and certain portions of code, without providing a mechanism for exploiting the native capabilities of a smartphone such as cameras, embedded sensors, etc.

Main highlighted points:

- Undoubtedly, embedded sensors are a very little taken into account, as we have been analyzing previously. There are few works that come to consider the embedded sensors in the modeling.

- Another interesting point is the fact that Android and iOS are the most chosen target platforms for case studies and evaluations. As a third option Windows Phone remains the most commonly used.

- The generated applications are largely native and are data-oriented type.

- In addition, almost all the proposals generate the application taking into account the structure of the project according to the IDE corresponding to the chosen platform; By means of this one can realize a compilation with the respective SDK, thus get the executable application.

- Finally, the type of evaluation most viewed is to make an illustration or case study of the proposal submitted.

The following table 3 displays a comparative study between the various approaches allowing to develop cross-platform mobile applications based on MDA approach.

Table 3. Comparative table of approaches for modeling and code generation of mobile applications

Work	***Input Model***	***Output Platforms***	***Based Modeling***	***Design Type***	***Mapping***	***Support embedded sensors***
W 1	UML	Java code with annotation	UML	Graphical	Acceleo	Not Supported
W 2	Abstract Syntax	Native code for Android and iOs	DSL (Xtext)	Textual	Xtend	Not Supported
W 3	Abstract Syntax	Native code for Android, iOS and Windows Phone	DSL (Xtext)	Textual	Xtend	Partially
W 4	UML	GUI for Android Platform	Windows Navigation Diagram	Graphical	Not specified	Not Supported
W 5	UML	GUI and Java Class files for Android	UML	Graphical	QVT and Acceleo	Not Supported
W 6	Abstract Syntax	Native code for Android and JSF	DSL (Xtext)	Textual	Xtend	Not Supported
W 7	IFML	GUI and business logic for Android and iOS	IFML mobile	Graphical	WebRatio	Not Supported
W 8	Abstract Syntax	GUI for mobile apps	MIM DSL	Graphical	Not specified	Not Supported
W 9	Abstract Syntax	Native code for Android and iOS	DSL (Xtext)	Textual	Xpand	Not Supported
W 10	Abstract Syntax	Native code for Android	DSL (Xtext)	Textual	Xtend	Not Supported
W 11	UML	GUI and business logic for Android and Windows Phone	UML Profile	Graphical	MAG	Partially
W 12	UML	GUI for mobile platform and class structure	Object Diagram	Graphical	ATL, DOM and Xpand	Not Supported

In the following section, a meta-model is proposed for modeling applications, providing the access to data from the embedded sensors and transmit them to various targets such as embedded databases, files, web services, etc. This proposal outperforms the previously presented works, by allowing them to also support the development of applications based on embedded sensors

4. ANDROID SENSOR CODE GENERETOR

In this approach, a meta-model for designing a mobile application is provided, 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 [26] to implement the meta-model and Xtend language [26] to perform different transformations.

Fig. 6 shows the different stages that characterize our approach.

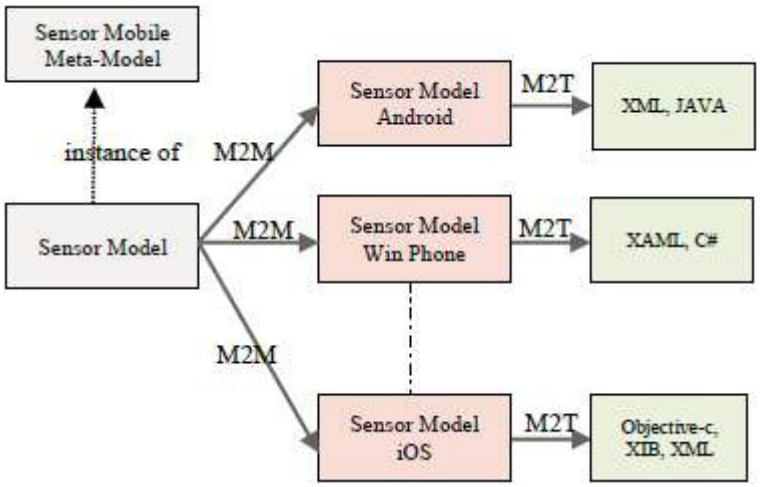

Figure 6. Architecture for code generation from a platform independent model [23]

After generating the code by using a set of templates, 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 Fig. 7 for more details).

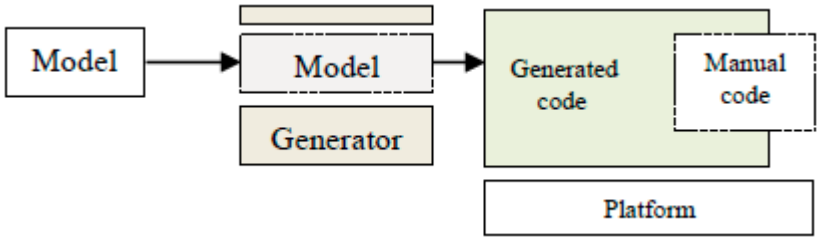

Figure 7. Model to Text transformation (M2T)

5. MOBILE SENSOR META-MODEL

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 meta-model proposed to model a mobile application making use of embedded sensors is presented in the Fig. 8.

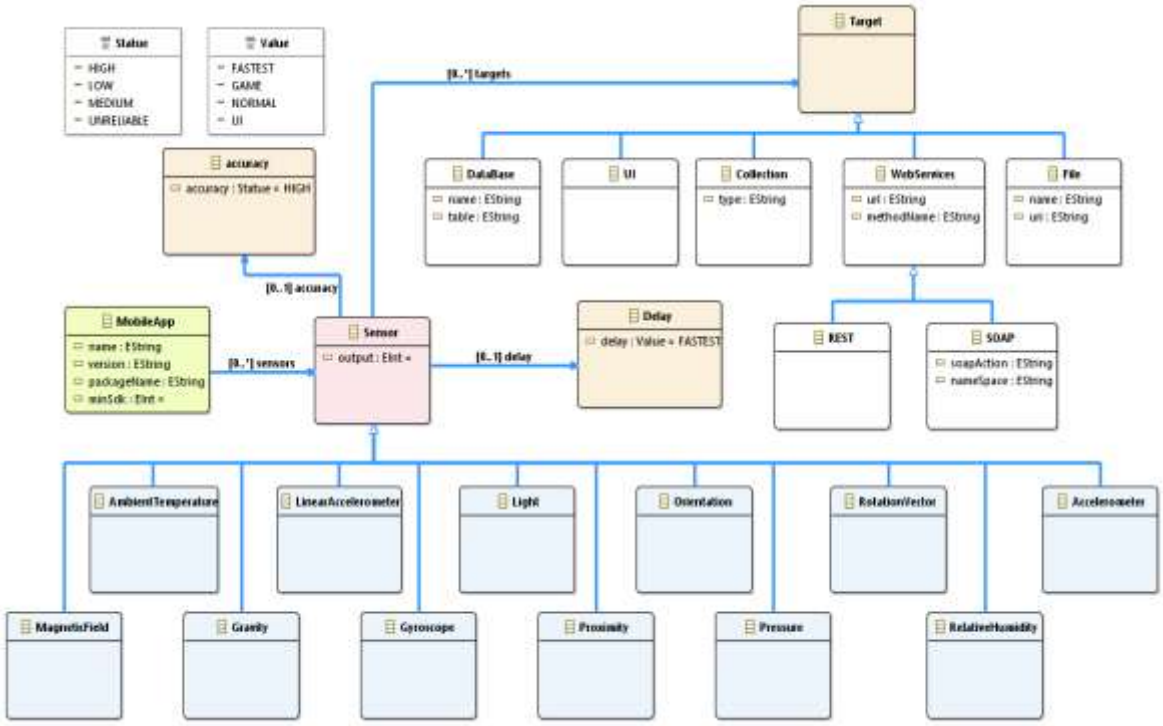

Figure 8. Mobile phone sensor meta-model

6. ANDROID SENSOR TEMPLATES

In this section, different templates used in code generation are discussed, in order to build a mobile application for Android, which makes use of embedded sensors.

To get this done, the next three steps should be followed:

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 the designated targets in a separate thread.

6.1 Activity Class Template

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 Fig. 9 for more details):

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

Figure 9. Extract of Activity Class

The proposed template for generating the *onSensorChanged()* method is presented below. The latter allows recovering data according to the specified sensor mentioned in target-data (see Fig. 10 for more details).

In the following sub-sections, the different proposed templates are presented.

```
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»
 }
}
'''
```

Figure 10. onSensorChanged method template

6.2 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.accelerometer"
android:required="true" />

This is an example manifest entry that filters apps that do not have an accelerometer.
A Template for Manifest.xml configuration file generation to support the use of embedded sensors is introduced below (see Fig. 11 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-feature
android: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-feature android:name="android.hardware.sensor.accelerometer"
                       android:required="true" />
                                «ENDIF»
                               «ENDFOR»
                              </manifest>
                                  '''
```

Figure 11. Manifest file Template

7. ILLUSTRATING EXAMPLE: TEMPERATURE AND HUMIDITY SMART SENSOR

Temperature and Humidity Smart Sensor detector is a tool allowing the measurement of the ambient humidity and the temperature of the environment in real time, by using the embedded sensors in the smartphone. (The table 4 below shows the environment sensors that are used in this application).

Table 4.The environment sensors that are used in this example

Sensor	Sensor event data	Units of measure	Data description
TYPE_AMBIENT_TEMPERATURE	event.values[0]	°C	Ambient air temperature.
TYPE_RELATIVE_HUMIDITY	event.values[0]	%	Ambient relative humidity.

7.1 Analysis and Model Creation

The application's model is presented below (see Fig. 12 for more details).

```
MobileApp "SmartSensor"{
	version "1.0"
	packageName "ma.uca.mobile"
	minSdk 4
	sensors(
		AmbientTemperature {
			output 1
			accuracy {LOW}
			delay{GAME}
			targets {UI}
		},
		RelativeHumidity{
			output 1
			accuracy {LOW}
			delay{GAME}
			targets {UI}
		},
	)
	Screen title "SMART SENSOR" orientation "portrait" {
		Layout [id 1 width "match" orientation "horizontal"
		 column "1"  weight "match" type "Linear"]{
			Label [id 1 text "MAGNETOMETER METAL DETECTOR"
			  gravity "center" paddingBottom "50"
			  paddingTop "20"],
			Label [id 2 text "TEMPERATURE" paddingBottom
			  "30"],
			Label [id 3 gravity "center" paddingBottom
			  "50"],
			Label [id 4 text "HUMIDITY" paddingBottom
			  "30"],
			Label [id 3 gravity "center"]
		}
	}
}
```

Figure 12. The application's model

7.2 The Generated Code

The configuration file generated from the application's model (see Fig. 13 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-sdk android:minSdkVersion="4" />
      <uses-feature
            android:name="android.hardware.sensor.ambient
      _temperature"
            android:required="true" />
      <uses-feature
            android:name="android.hardware.sensor.relative_hum
      idity"
            android:required="true" />
</manifest>
```

Figure 13. Manifest.xml configuration file generated

Code snippet generated for the Activity class (see Fig. 14 for more details):

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

Figure 14. Code Snippet generated for the Activity Class

7.3 The Graphical Interface Generated for the Android Platform

The figure below shows a generated graphical interface, 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 Fig. 15 for more details).

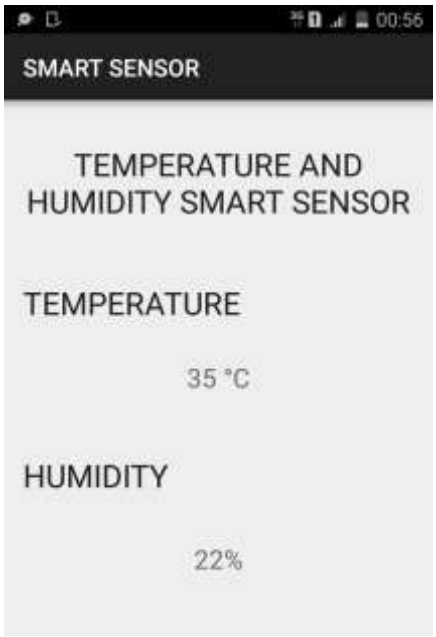

Figure 15. GUI generated for Temperature and Humidity Smart Sensor

8. FUTURE WORK

As a further perspective, this approach can be brought to a broader scheme, i.e. for all mobile platforms (e.g. iOS, Windows Mobile, etc.), by implementing the suggested approach. The Xtext language is used to realize the meta-model and Xtend for the implementation of the various transformations and Templates. The proposed meta-model comes to enhance the previously established work presented in [16], In fact, this latter allow to model mobile applications that make use of embedded sensors, plus the ability to transmit collected data to various targets, such as web services, embedded databases, files, or graphical interfaces.

For future improvements, the possibility to extend our method to deal with data recovering from external sensors might be considered; as well as the design a software layer to ensure data security access, and the integrity of recovered data.

9. CONCLUSION

By and large, thanks to the growth of the various mobile technologies, that develops the same application for these different platforms, this latter has become a daunting task. Taking into account the fact that each platform uses different tools (e.g. programming languages and user interface declarations, etc.), these heterogeneity development tools and languages make the burden heavier to develop multiplatform applications. Thus, developers are asked to select an option on the platform, while guaranteeing a large enough diffusion.

This work focuses on mobile applications which use embedded sensors, in order to generate native code without having to redevelop all files and lines of code, which are sometimes redundant and complicated to set up. Therefore, the generation of code for the embedded sensors in mobiles apps was not taken into account in the most recent work, as pointed out in the comparative study.

The potential benefits of the MDA are: the reduction of cost, in terms of maintaining only one code to write, and the time reduction, in terms of targeting multiple devices and platforms by writing only one code, and also by having the ability to research cross-platform applications development with one's effort and findings.

By and large, this paper established a study on the embedded sensors in mobile devices, adopting an 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 meta-model proposed.

REFERENCES

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

[2] Kent S (2002) Model driven engineering. In Integrated formal methods (pp. 286-298). Springer Berlin/Heidelberg.

[3] Bézivin J (2004) In search of a basic principle for model driven engineering. Novatica Journal, Special Issue, 5(2): 21-24.

[4] Android (2016) Sensors Overview. https://developer.android.com/guide/topics/sensors/sensors_overview.html. Accessed 12 June 2017.

[5] Zhi-An Y, Chun-Miao M (2012) The development and application of sensor based on android. In : 8th International Conference on Information Science and Digital Content Technology (ICIDT), (Vol. 1, pp. 231-234). IEEE.

[6] Android (2017) Sensor stack. https://source.android.com/devices/sensors/sensor-stack. Accessed 25 July 2017.

[7] Xing L, Jiqiang L, •Wei W, Yongzhong H, Xiangliang Z (2017) Discovering and understanding android sensor usage behaviors with data flow analysis, World Wide Web (pp. 1-22).

[8] El Hamlaoui M (2015) Mise en correspondance et gestion de la cohérence de modèles hétérogènes évolutifs (Doctoral dissertation, Université Toulouse le Mirail-Toulouse II).

[9] OMG (1989) Object Management Group. http://www.omg.org Accessed April 2016.

[10] Hailpern B, Tarr P (2006) Model-driven development: The good, the bad, and the ugly. IBM systems journal, 45(3):451-461.

[11] Maroukian K, Apostolopoulos C, Tsaramirsis G (2017) Extending model driven engineering aspects to business engineering domain: a model driven business engineering approach. International Journal of Information Technology, 9(1):49-57.

[12] OMG (2008). Meta object facility (MOF) 2.0 query/view/transformation specification. Final Adopted Specification (November 2005).

[13] Alkhir SS (2003). Understanding the model driven architecture. Published in Methods & Tools.

[14] Freitas F, Maia PHM (2016) JustModeling: An MDE Approach to Develop Android Business Applications. In : VI Brazilian Symposium on Computing Systems Engineering (SBESC) (pp. 48-55). IEEE.

[15] Vaupel S, Taentzer G, Gerlach R, Guckert M (2016) Model-driven development of mobile applications for Android and iOS supporting role-based app variability. Software & Systems Modeling (pp. 1-29).

[16] Lachgar M, Abdali A (2016) Modeling and generating native code for cross-platform mobile applications using DSL. Intelligent Automation & Soft Computing (pp. 1-14).

[17] Channonthawat T, Limpiyakorn Y (2016) Model Driven Development of Android Application Prototypes from Windows Navigation Diagrams. In : International Conference on Software Networking (ICSN) (pp. 1-4). IEEE.

[18] Benouda H, Azizi M, Esbai R, Moussaoui M (2016) Code generation approach for mobile application using acceleo. In : International Review on Computers and Software (IRECOS), 11(2), :160-166.

[19] Lachgar M, Abdali A (2015) Modeling and generating the user interface of mobile devices and web development with dsl. Journal of Theoretical & Applied Information Technology, 72(1).

[20] Acerbis R, Bongio A, Brambilla M, Butti S (2015) Model-driven development based on omg's IFML with webratio web and mobile platform. In International Conference on Web Engineering (pp. 605-608). Springer, Cham.

[21] Geiger-Prat S, Marín B, España S, Giachetti G (2015) A GUI modeling language for mobile applications. In : 9th International Conference on Research Challenges in Information Science (RCIS), (pp. 76-87). IEEE.

[22] Majchrzak TA, Ernsting J, Kuchen H (2015) Model-Driven cross-platform apps: Towards business practicability.

[23] Lachgar M, Abdali A (2014) Generating Android graphical user interfaces using an MDA approach. In : Third IEEE International Colloquium in Information Science and Technology (CIST), (pp. 80-85). IEEE.

[24] Usman M, Iqbal MZ, Khan MU (2014) A model-driven approach to generate mobile applications for multiple platforms. In : Software Engineering Conference (APSEC), 21st Asia-Pacific (Vol. 1) (pp. 111-118). IEEE.

[25] Sabraoui A., El Koutbi M., Khriss I (2013) A MDA-based model-driven approach to generate GUI for mobile applications. International Review on Computers and Software Journal (IRECOS), 8(3) : 845-852.

[26] Bettini L (2016) Implementing domain-specific languages with Xtext and Xtend. Packt Publishing Ltd.