Applied Mathematics & Information Sciences
An International Journal

http://dx.doi.org/10.18576/amis/130304

A Model-Driven Engineering Approach for Automating the Portability of User Interfaces in Native Mobile Applications

Riham Abdel Kader and Wassim El Hajj Chehade*

Department of Mathematics and Computer Science, Beirut Arab University, Beirut, Lebanon

Received: 15 Jul. 2018, Revised: 12 Feb. 2019, Accepted: 23 Feb. 2019
Published online: 1 May 2019

Abstract: In the last decade, mobile applications have been on the rise. When developing an app, and because of the wide variety of existing mobile devices and operating systems, developers need to support several target platforms. Designing, and developing a mobile app on several platforms require from the developer to invest a considerable amount of extra time and effort, which delays the introduction of the app to the market. The aim of this paper is to propose a technique that takes the user interface developed for one platform as input and generates its equivalent interface for the other platform. In simple terms, our approach translates an iOS interface into its equivalent Android interface and vice versa. Our approach is based on a model-driven solution that proposes a generic UML profile that builds the link between the different resources in Android and iOS. Using this profile, the framework can map any iOS resource to its correct counterpart in Android, and vice versa. Experiments have shown that our approach is feasible, fast, easy to use by developers, and does not require any interference from their side.

Keywords: UML, Mobile Application Development, MDD, Platform Modeling, Application Migration, Cross-Platform Development

1 Introduction

In the last several years, there has been an explosion in the number of mobile apps in the market. Google Play Store leads with 2.2 million apps and Apple App Store follows with 2 million, while Windows Store has 669 thousand apps as of June 2016 [1]. The total number of apps downloaded in 2016 reached almost 225 million and is projected to increase to 268 million in 2017. With a revenue amounting to 88.3 billion dollars in 2016 from app sales and a projected 92 billion dollars in 2018 [1]; there is an increase in competition between companies in the field, pushing the industry to focus on finding ways to develop apps in a faster manner without jeopardizing the quality of the software.

To reach a wide audience, and since the mobile device market is divided among mainly iOS and Android, developers need to build apps that support these two target platforms. Developing an app for both platforms requires double time and effort, especially that these two platforms differ greatly from each other. The aim of this research is to propose a model-driven solution that minimizes the time and effort of the programmer when developing the user interfaces of an app on both platforms.

Model-driven development (MDD) is a development paradigm that aims at offering solutions to reduce the development cost of software while increasing its quality. Perhaps the most mature formulation of this vision is the Model-Driven Architecture (MDA) initiative [2], undertaken by the Object Management Group (OMG). MDA is based on using models to represent on one hand the essence of an application and on the other hand the resources of the target platforms. Given a Platform Independent Model (PIM) or a Platform Specific Model (PSM) and a Platform Description Model (PDM), code is automatically generated using a predefined Mapping Model (MM). Such models can be described using the Unified Modeling Language (UML) [3].

Many model-driven approaches have been proposed to address the challenge of a cost effective, portable mobile applications development [4]. These approaches propose new frameworks and languages that must be used to build the app. Although, this is one way to achieve portability, the application developer must, on the first

* Corresponding author e-mail: r.abdelkader@bau.edu.lb

hand, learn a new language and how to use these new frameworks; and on the second hand, the new frameworks do not encapsulate all the richness that the native development frameworks such as Android studio and iOS Xcode offer.

In this approach, we are not going to propose to the developer to use a new development framework or language, but we offer him a solution in which he /she uses the existing native IDEs to build his user interfaces. Once the GUI is built on one platform, our approach allows the developer to port it without extra effort to a new target platform. Our approach is based on model transformations that realize the mapping of an iOS user interface to its Android counterpart, and vice versa. We propose two platform description models for each of the iOS and Android platforms, and a mapping algorithm that links these two models together. Our framework takes a user interface (say iOS) as input, and using the PDM and the mapping algorithm, generates the code for the Android interface. The solution presented in this paper targets the modeling and code generation of the user interface, as part of the whole application. In fact, the average time spent on implementing the user interface of a program is usually 50% of the whole application [5], which is a considerable amount of time.

The solution we propose is general and can be easily extended to incorporate new resources as well as target platforms beyond iOS and Android. The addition of any of the above requires minimal extra implementation efforts. It is worth noting that our approach can also be used as a basis for application migration. Moreover, we realize that every new version of iOS and Android might introduce some new attributes while some other attributes might become deprecated. Our approach can easily handle these kinds of updates, by either adding new attributes or resources for the newly-added attributes or removing the attributes and resources corresponding to the deprecated ones.

In the next section, we examine the domain model of Android and iOS and we define the Mobile Resource Modeling profile (MRM). Section 3 describes the mapping from one platform to the other. In Section 4, we present a case study to illustrate our approach. Section 5 overviews the related work and we conclude in section 6.

2 Mobile Platform Modeling

As shown in Figure 1, the mapping from one interface to the other requires the existence of mobile Platform Description Models (PDM) for both Android and iOS. In this section, we describe our approach for modeling these two platforms which is conducted in two steps. The first step aims at enumerating all the concepts required to cover the iOS and Android GUI domain. The output of this stage is called the domain model of the profile. It is considered as a specification of the domain-specific language. The second stage then consists of implementing

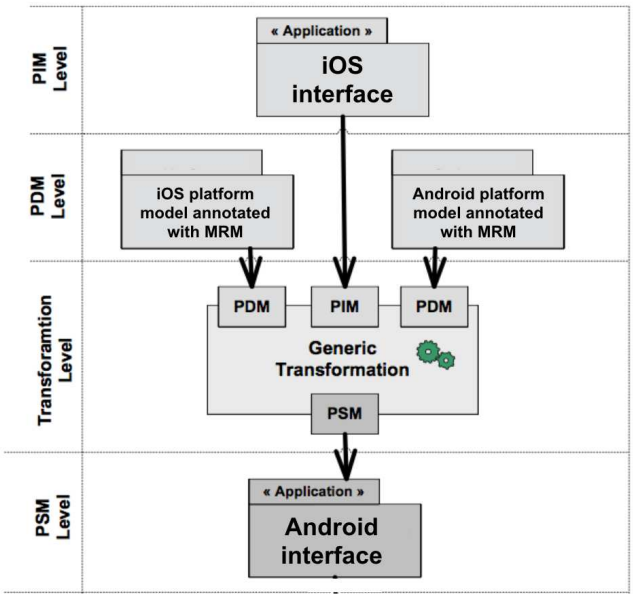

Fig. 1: Proposed framework

and modeling this specification in terms of UML extensions, i.e. defining UML stereotypes and their related properties and constraints.

2.1 The Domain Model of Android and iOS

The domain models have been built on the basis of a detailed analysis of the iOS and Android mobile application API standards. As a result of the graphical user interface analysis, we have classified the GUI elements into two groups:

- Windows represent an area on the screen that displays information, with its contents being displayed independently from the rest of the screen such as view and relative layout.
- Widgets are software components with which a mobile user interacts through direct manipulation to read or edit information about an application. Widgets can be divided into four categories:
  - display of collections of related items (various list and canvas controls),
  - initiation of actions and processes within the interface (buttons and menus),
  - navigation within the space of the app (links, tabs and scrollbars),
  - representing and manipulating data values (labels, check boxes, radio buttons, sliders, spinners, etc.).

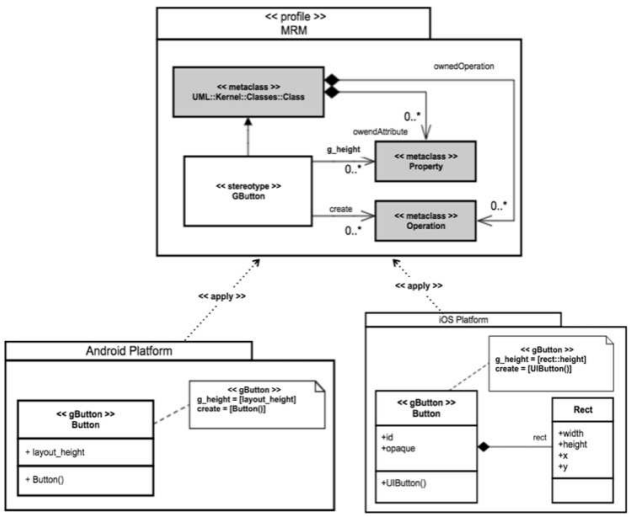

Fig. 2: Extract of the MRM Profile with the Android and iOS platform description models

2.2 MRM Profile

Using the above extracted domain model, we have implemented the domain view as a UML 2.0 profile. Profiles, such as SRM profiles, have been previously designed and used for modeling multitasking and embedded applications [6]. We call our suggested UML profile an MRM. MRM is not a new API for mobile applications such as Android or IOS, but a language to describe those APIs.

The MRM profile is based on the "Resource-Service" pattern. That pattern allows describing resources that own properties and provide services. Some properties and services play roles. Such roles are modeled as resources attributes. Each concept in the windows and widgets groups of the domain model is modeled as a resource with a stereotype. Figure 2 provides an overview of the profile architecture and platform models showing the Button resource. In the MRM profile, a button is modeled as a *gButton* stereotype. A button resource owns some attributes. Among those attributes, the *g_height* attribute plays the role of the height characteristic of the button. A button resource provides also *services*, such as the *create* service. Due to space limitations, it is out of the scope of the paper to describe in very details the MRM profile. A more thorough description of the Android and iOS platform description models is given in Section 3.3.

The main goal of MRM profile is to be used in a generative mobile application development process context. Therefore, MRM provides finely-detailed artifacts modeling capabilities which allow the description of mobile device GUI in a detailed way. It also builds a bridge between the description models of the different platforms and thus facilitates the mapping between the platforms.

3 Mapping and Code Generation

The aim of our approach is to make it easier and faster for the developer to implement the user interfaces of an app on different target platforms. Currently, if a user interface has been developed for an iOS app, the developer has to redo the whole interface development for Android, hence doubling the implementation effort, time and money. Alternatively, the developer can use a cross platform development approach, but in this case, he/she has to learn and deal with a new technology that has its advantages and drawbacks. Using our approach shown in Figure 1, the developer can feed the implemented iOS interfaces to our framework which automatically generates the code for the Android interfaces. Platform Description Models (PDM) for the iOS and Android as well as a mapping algorithm are needed to achieve this translation. The PDMs model the resources in each of the platforms while the mapping algorithm is a bijective mapping that links the components in the two platforms. In the next sections, we present our running example and then give a detailed description of the platform description model and the mapping algorithm.

3.1 Running Example

To explain and demonstrate our technique, we use a running example throughout this section. In the example, we assume the developer has created a simple iOS user interface and uses our approach to automatically generate the equivalent android interface. The interface consists of a textfield and a button. Figure 3 shows an extract of the user interface that codes the button resource in iOS. The same interface developed for the android OS is shown in Figure 4. The objective of this research is to develop an automatic mapping that can generate the content of Figure 4 when the content of Figure 3 is given as input. It is worth noting that our running example is kept short for ease of presentation and explanation; however, our approach covers other aspects in user interfaces, like textfields, labels, checkboxes, etc. We show a more thorough example in Section 4.

3.2 Preprocessing Step

Before delving into the details of our approach, we describe our starting point. First, the user needs to create an interface on either Android or iOS. This can be done in two ways; either by writing directly the XML code or for an easier option, by using a drag and drop tool. In fact, both of the Android Studio or Apple Xcode IDEs provide

```
<button opaque="NO" contentMode="scaleToFill" fixedFrame="YES"
        contentHorizontalAlignment="center"
        contentVerticalAlignment="center"
        buttonType="roundedRect"
        lineBreakMode="middleTruncation"
        translatesAutoresizingMaskIntoConstraints="NO"
        id="OwD-7p-N5n">
    <rect key="frame" x="144" y="163" width="86" height="30"/>
    <autoresizingMask key="autoresizingMask" flexibleMaxX="YES"
                      flexibleMaxY="YES"/>
    <state key="normal" title="Button"/>
</button>
```

Fig. 3: An extract of the iOS interface coding a button resource

```
<Button
    android:layout_width="86"
    android:layout_height="30"
    android:text="New Button"
    android:id="@+id/button"
    android:layout_below="@+id/editText"
    android:layout_centerHorizontal="true" />
```

Fig. 4: An extract of the Android interface coding a button resource

the possibility to build an interface by drag and drop. If the interface is built using the easier drag and drop option, the IDE generates simultaneously the XML code for the interface.

Given the XML code of the iOS interface, our approach starts by parsing the fragments into the corresponding XML tree. Then we traverse through the tree to generate the Android interface. To do so, we need a way to map between the resources in iOS and the resources in Android. For that, we have designed platform description models for both iOS and android. The common description of these models using the MRM profile allows the creation of a mapping between the operating systems.

The next section describes the platform description models we have designed and then we describe how the mapping takes place.

3.3 Platform Description Model

A Platform Description Model (PDM) displays all the resources that a given platform offers. For example, Android offers, among others, a Button and a Textview resources. In iOS, these same resources are called Button and TextField. A PDM also lists the attributes each resource has. Figure 2 shows the Android and iOS modeling of the Button resource. A complete PDM also includes a model of the other resources, e.g. TextField, TextBox, Label .... Extensive PDMs for the iOS resources and the Android resources are shown in Figures 5 and 6.

Every resource in a PDM contains its corresponding list of attributes. We define these attributes by examining and analyzing the XML API of the Android and iOS interfaces. Figure 4 shows that a node Button in the Android XML contains several attributes called *layout_width*, *layout_height*, *text*, *id*, *layout_below*, *layout_centerHorizontal*, each one with its own String value. We map each of the XML attribute names to attributes in the Android Button resource model as shown in Figure 6. Other Android resources and their attributes are depicted in Figure 6. However, the button node in the iOS XML specification contains not only attributes but also children nodes like *rect*, *autoresizingMask*, and *state*. Each of the children node is modeled as a separate resource that is associated with the Button resource. Figure 5 shows the *Rect* child node modelled as a separate resource linked to the Button resource (in this case it is associated to the Widget class and by inheritance to the Button resource).

As can be seen in Figures 5 and 6, the two models differ in many aspects, the number of attributes, the name of the attributes, as well as the structure of the attributes. For example, the height size of a button in Android is called layout_height while in iOS the height is encoded within a Rect class that is associated with the Button resource. These differences between the two platforms pose a challenge for automatic mapping. To overcome this difficulty, we use the UML *stereotypes* defined in MRM as a way to link together resources or attributes that represent the same thing while having different names and structures. A stereotype allows extendibility in UML by permitting designers to create new model elements derived from existing ones. These elements are assigned a specific property that renders them suitable for a specific domain or usage. A stereotype is denoted as a name enclosed by guillemets (<<>>). The MRM profile illustrated in Figure 2 defines the stereotype <<GButton>> to represent the button resource. The same stereotype is then used in the Android and iOS PDM to annotate the button resources.

The MRM profile shown in Figure 2 specifies that the GButton stereotype has a g_height property. Therefore, the Button resource in the two PDMs needs to implement the g_height property. That is another major difference between the two PDMs: the g_height property in Android corresponds to the layout_height attribute as part of the Button resource, while in iOS it is modeled as a height attribute in another resource Rect which is associated to the Button resource. To facilitate the mapping between the two different structures, we annotate the PDMs with extra information. As shown in Figures 5 and 6, the attributes of the Button resource in the iOS and Android PDM are referenced by the attributes of the GButton stereotype. For example, in the Android PDM, the layout_height attribute in the class Button is referenced by

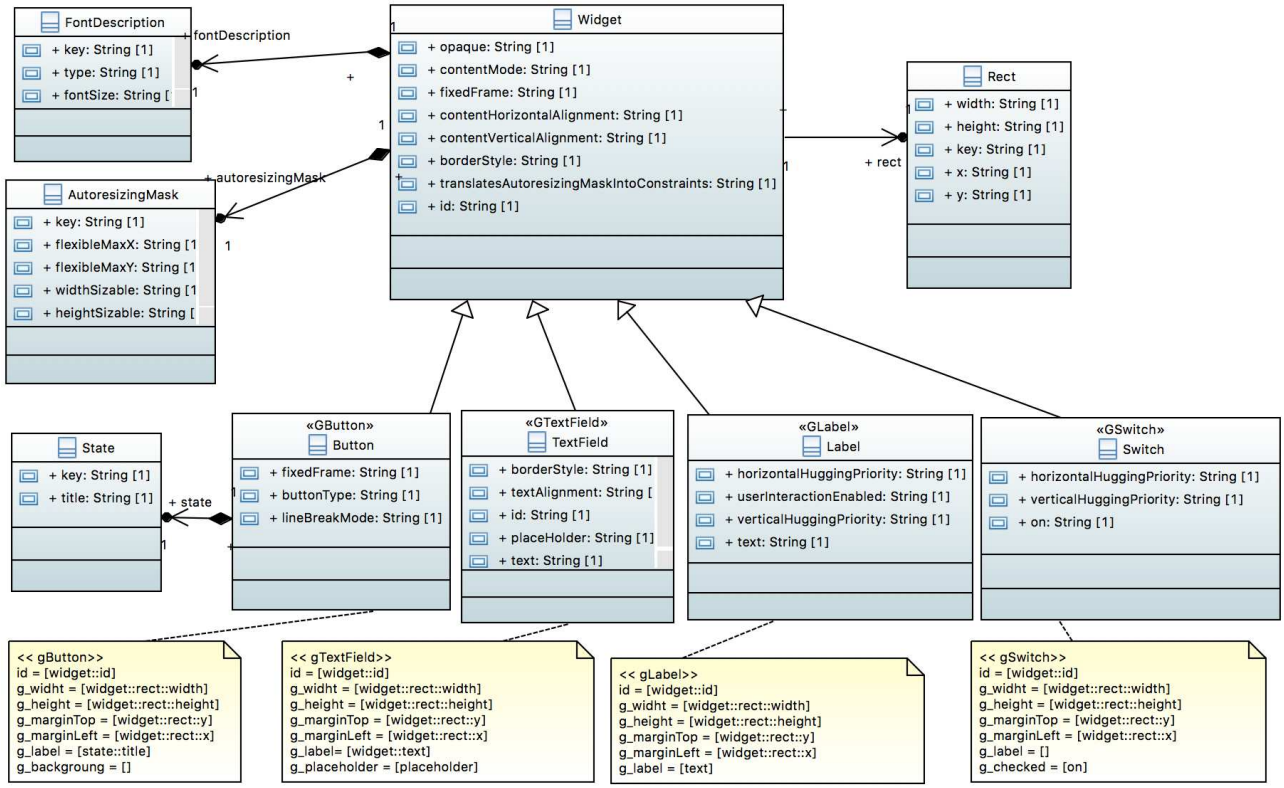

Fig. 5: Extract of the iOS platform description model

the g_height attribute of the Gbutton stereotype ("g_height = [layout_height]"). While the attribute height in the iOS PDM, which is inside the Rect class, is referenced to the g_height attribute of the GButton stereotype ("g_height = [rect:height]"). This common description of the resources as well as their attributes using the stereotypes in the MRM profile help us perform a generic mapping between iOS and Android platforms.

3.4 The Mapping

Our approach takes the XML parse tree of an interface for a given platform (say iOS) as input and generates the equivalent interface for the other platform (Android) making use of the PDMs we have designed. In this section, we describe how the mapping takes place.

Algorithm 1: The mapping algorithm is shown in Algorithm 1. The algorithm MAPINTERFACE takes the XML tree T as input, the name of the source platform for which the interface is designed (in our case it is iOS), and the name of the target platform for which we wish to generate the new interface (in our running example it is Android).

The algorithm starts by extracting the PDMs of the source and target platforms (lines 2-3). It then traverses the XML tree in a depth-first fashion. For every

Algorithm 1 - Map an interface from one platform to another.

```
1:  procedure MAPINTERFACE(Tree T, String sourcePlatform,
    String targetPlatform)
2:      sourcePDM ← getPDM(sourcePlatform);
3:      targetPDM ← getPDM(targetPlatform);
4:      for all nodes n in T do
5:          st ← getStereotype(n, sourcePDM);
6:          if st != null then
7:              resource ← getResource(targetPDM, st);
8:              output("<" + resource + " ");
9:          for all attribute attr in n do
10:             st_property ← getStereotypeProperty(attr,
                              sourcePDM);
11:             newAttr ← getAttribute(targetPDM, resource,
                              st_property);
12:             if newAttr != null then
13:                 OUTPUTATTRIBUTE(newAttr, attr, resource,
                              SourcePDM, TargetPDM);
14:         for all Attributes attr in resource do
15:             if attr is not output yet then
16:                 output(attr + " = \"" + defaultValue() + "\"");
17:         output(">");
```

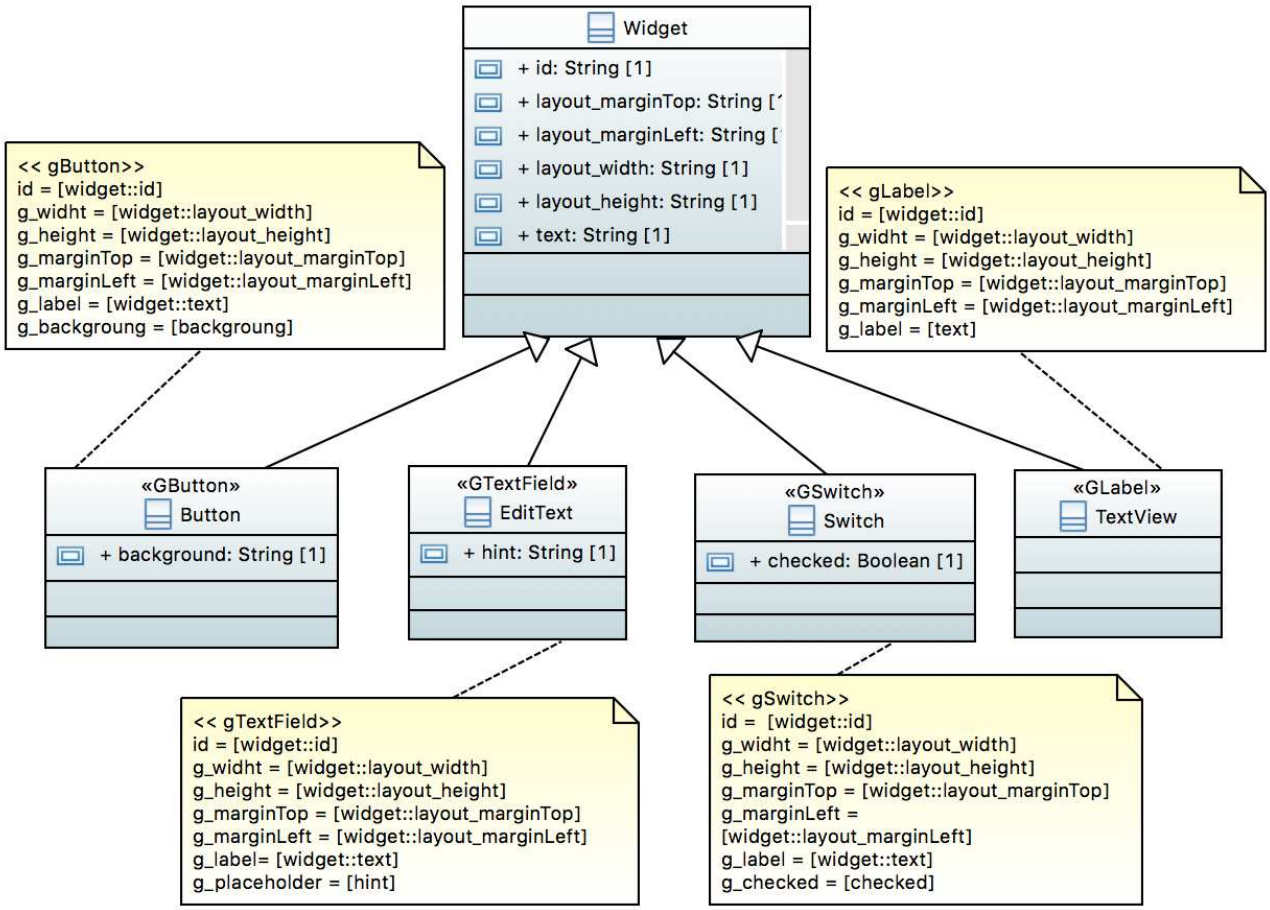

Fig. 6: Extract of the Android platform description model

encountered node, the algorithm examines the source PDM and gets the stereotype corresponding to that resource node (line 5). If the node has an associated stereotype, the algorithm searches for the resource that has the same stereotype in the target PDM (lines 6-7). Then the target resource is output inside an XML tag "<" (line 8). If the node of the tree has no associated stereotype, this means that this node is not shared among the different platforms. For example, the rect node is not assigned any stereotype since its corresponding rect resource exists in iOS but not in Android. When translating an iOS interface to Android, if a rect node is encountered in the tree, it will be ignored and omitted from the XML output.

After mapping a node in the tree, the algorithm maps its attributes (lines 9-13). Even if the node has no stereotype and is not output into the XML code, its attributes will still be mapped. Taking again the rect node example, although the node is not mapped, some of its attributes (*e.g.*, width, height) must be mapped. For each attribute, the algorithm extracts its property from the source PDM and finds the corresponding attribute in the target PDM (lines 9-11). If the target attribute exists, it will be output by calling Algorithm 2 (lines 12-13).

Algorithm 2: Algorithm 2 outputs the attributes of a resource as XML code. There are several attribute types whose output should be handled differently.

- If the target attribute is of type id, then the value of the attribute is generated using the function generateID() that generates a unique random id for the resource (lines 2-3).
- If the target attribute is of the form class:attribute then we need to output a new XML node of type class and within it the target attribute and its value which is the same value specified in the source attribute (lines 4-5). To demonstrate this case, we consider the example of mapping the layout_height attribute of a button from Android to iOS. The property of the layout_height attribute is g_height which corresponds to the attribute rect:height in the iOS PDM, which means that the property g_height is mapped to the attribute height in the class rect. In this case, the Android attribute layout_height does not map to a single attribute but to the height attribute within the

rect class, and therefore "<rect height = someValue" becomes an output in the XML code. We also need to output all attributes that belong to the class. This is accomplished by calling Algorithm 3 which will be explained shortly (line 6). Finally at line 7, the algorithm closes the tag for the XML node class.

- Attributes might also be of type position, meaning that this attribute specifies the position of the widget on the GUI screen. Example of position attributes are: layout_below, layout_centerHorizontal, x, y. In this case (lines 8-10), the algorithm extracts the absolute position of the source attribute and then outputs the target attribute with the value of the extracted position. We need to do this because positioning of widgets differs greatly between Android and iOS. In iOS, positions are absolute numbers relative to the screen dimensions, while in Android positions are specified relative to other widgets placed on the screen. The absolute position is extracted and computed in the preprocessing phase when building the XML parse tree of the source GUI code (Section 3.2). We omit the details of the absolute position extraction mechanism as it is beyond the scope of this paper. However, we would like to note that android has recently launched the "ConstraintLayout" attribute that allows to create large and complex layouts without the need to use nested view groups. It is similar to "RelativeLayout", however; it is more flexible and easier to use. The introduction of constraintLayout makes it easier for our tool to map positions between iOS and Android. Using the absolute position allows our tool to handle different screen sizes and resolutions, increasing the portability of our approach. Moreover, the iOS XML code uses the unit "dp" for the position of its widgets, which is a density-independent pixel, making the position independent of the resolution of the screen.
- Finally, for all other types of attributes, the attribute is output with a value equal to the value specified in the source attribute (lines 11-12).

Algorithm 3: Algorithm 3 loops over all attributes in a class (line 2), and if the attribute is not the same as the attribute that is already output in Algorithm 2 (line 3), it will retrieve the property of the attribute from the target PDM (line 4). In line 5, the algorithm searches for the corresponding attribute in the source PDM. If the source attribute exists, Algorithm 2 is called to output the target attribute attr with the value in the sourceAttr (lines 6-7). Otherwise the target attribute is output with its default value (lines 8-9). Once an attribute of a resource is output as XML code, it will be flagged so it does not get output again.

So far Algorithm 1 has mapped every attribute in one platform to its corresponding attribute in the other platform. However, if we compare the XML code in Figures 3 and 4 and the two PDMs in Figures 5 and 6, we realize that the iOS Button has more attributes than that of Android. For example, the iOS attributes *opaque*, *contentMode*, *FixedFrame* do not have counterpart Android attributes. Therefore, if our starting point is an Android interface and we want to generate its corresponding iOS interface, the algorithm omits generating those extra iOS attributes. Lines 14-16 of Algorithm 1 handle this issue by looping over all the attributes of the resource of the target platform and if any of these attributes has not been output yet as XML code, it will be output with a default value. The default value of an attribute is taken as the value that gets assigned to the attribute when the resource is dragged and dropped in the IDE. For example, when a button is dragged and dropped in iOS the default value of the attributes *opaque*, *contentMode*, *FixedFrame* is respectively "NO", "scaleToFill", "YES". When all attributes are mapped, the algorithm outputs a closing tag for the XML node (line 17).

Algorithm 2 - The Algorithm that handles the output of attributes.

```
1:  procedure OUTPUTATTRIBUTE(String newAttr, String attr,
    String resource, PDM SourcePDM, PDM TargetPDM)
2:      if newAttr == "id" then
3:          output(newAttr + " = \"" + generateID() + "\"");

4:      else if newAttr hasForm class:attribute then
5:          output("<" + class + " " + newAttr + " = \"" +
                attr.value + "\"");
6:          OUTPUTALLATTRIBUTES(class, resource, newAttr,
                SourcePDM, TargetPDM);
7:          output(">");

8:      else if newAttr is a position attribute then
9:          int pos = extractAbsolutePosition(attr);
10:         output(newAttr + " = \"" + pos + "\"");

11:     else
12:         output(newAttr + " = \"" + attr.value + "\"");
```

Algorithm 3 - Output all attributes in a class.

```
1:  procedure OUTPUTALLATTRIBUTES(String class, String
    resource, String attribute, PDM SourcePDM, PDM
    TargetPDM)
2:      for all attributes attr in class do
3:          if attr != attribute then
4:              property ← getProperty(attr, TargetPDM);
5:              sourceAttr ← getAttribute(SourcePDM,
                    resource, property);
6:              if sourceAttr != null then
7:                  OUTPUTATTRIBUTE(attr, sourceAttr,
                        resource);
8:              else
9:                  output(attr + " = \"" + defaultValue() + "\"");
```

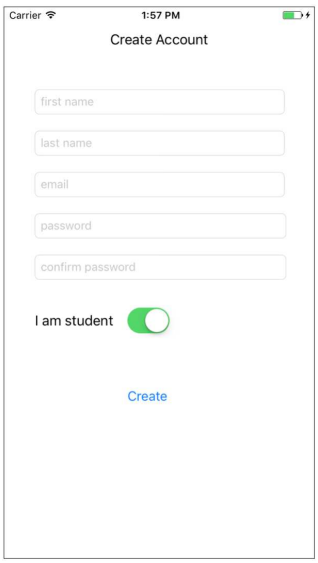

Fig. 7: The iOS page that is ported to Android

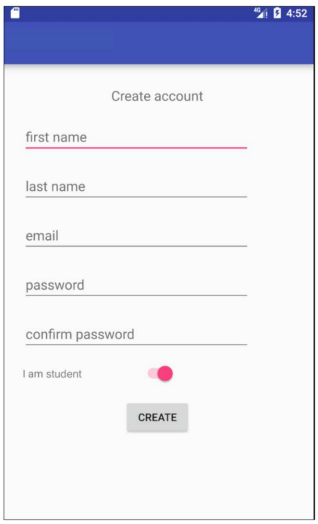

Fig. 8: The equivalent Android page generated by our framework

Algorithm complexity: Our proposed algorithms are efficient. Algorithm 3 contains one for loop that iterates over the attributes in a given class, so its complexity is proportional to the number n of attributes in a given class. Based on the Android and iOS specification documents, the number n is small and limited. As a result, we can say Algorithm 3 has a constant complexity of O(1). Algorithm 2 contains no iterations but calls Algorithm 3, therefore its complexity is equal to that of Algorithm 3.

Algorithm 1 contains a nested for loop with two inner loops. The outer loop (line 4) iterates over the nodes in the XML tree while the first inner loop (line 9) iterates over the attributes of each source node and the second inner loop (line 14) iterates over the attributes of the target node. Therefore the complexity of Algorithm 1 is O(mn) where n is the number of nodes in the XML tree and m is the sum of the number of attributes in the source and target nodes. The number n varies with the size of the tree which depends on the complexity of the screen we are mapping. The number m is small as it represents the number of attributes a certain widget can have which, based on the Android and iOS specification documents, is limited.

4 Case Study

To demonstrate the applicability of our approach, we have applied it on a case study that consists of developing the pages of a mobile application in iOS using Xcode and trying to port these pages to Android using our tool. The case study is about a school management system that can be used by teachers, students and parents of the students to interact with each other. It was developed in iOS, then using our tool the pages of this application are mapped to android pages. This application consists of 31 pages, and took around 100 working hours to develop. Each page contains a different set of widgets. To develop these pages on Android, it requires other 100 hours of development. Using our tool, we automatically generate the Android pages with zero cost and effort.

Figure 7 shows one of the 31 pages we have developed. The page consists of the “Create Account” page and includes two labels, five text fields, one switch and one button. All the widgets of the page are included in a view. Figure 9 depicts the XML source code corresponding to the create account page shown in Figure 7. The source code shows some of the widgets’ properties. For example, we can see that all the text fields have values for the placeholder attributes. We can see also that all the widgets are positioned in specific places defined by the attribute x and y in the rect node. The idea is to show that these values are preserved when we use our tool to port the page.

Figures 8 and 10 show respectively the generated user interface of the create account page on the Android platform and its corresponding XML source code. We can see clearly that our approach has preserved the position and the properties of the widgets as they have been specified in the iOS version (i.e. placeholders, widget size, position). For example, the text field ‘first name’ in iOS has an absolute position at x= 37 and y=100 as shown in Figure 9. This position is conserved after porting the page on Android using our tool. We can see in Figure 10 that the edit text ‘first name’ has the layout_marginLeft property set to 37dp and layout_marginTop property set to 100dp.

We suppose now that the application developer has started by creating the Android version of the “create

```
<view key="view" contentMode="scaleToFill" id="8bC-Xf-vdC">
    <rect key="frame" x="0.0" y="0.0" width="375" height="667"/>
    <autoresizingMask key="autoresizingMask" widthSizable="YES" heightSizable="YES"/>
    <subviews>
        <textField opaque="NO" clipsSubviews="YES" contentMode="scaleToFill" fixedFrame="YES" contentHorizontalAlignment="left" contentVerticalAlignment="center"
            borderStyle="roundedRect" placeholder="last name" textAlignment="natural" minimumFontSize="17" translatesAutoresizingMaskIntoConstraints="NO" id="NLh-ik-w8P">
            <rect key="frame" x="37" y="150" width="302" height="30"/>
        </textField>
        <button opaque="NO" contentMode="scaleToFill" fixedFrame="YES" contentHorizontalAlignment="center" contentVerticalAlignment="center" buttonType="roundedRect"
            lineBreakMode="middleTruncation" translatesAutoresizingMaskIntoConstraints="NO" id="6UH-6N-qRl">
            <rect key="frame" x="21" y="456" width="304" height="30"/>
            <state key="normal" title="Create account"/>
        </button>
        <textField opaque="NO" clipsSubviews="YES" contentMode="scaleToFill" fixedFrame="YES" contentHorizontalAlignment="left" contentVerticalAlignment="center"
            borderStyle="roundedRect" placeholder="first name" textAlignment="natural" minimumFontSize="17" translatesAutoresizingMaskIntoConstraints="NO" id="1Mz-ch-Rx2">
            <rect key="frame" x="37" y="100" width="302" height="30"/>
            <autoresizingMask key="autoresizingMask" flexibleMaxX="YES" flexibleMaxY="YES"/>
        </textField>
        <label opaque="NO" userInteractionEnabled="NO" contentMode="left" horizontalHuggingPriority="251" verticalHuggingPriority="251" fixedFrame="YES" text="Create Account"
            textAlignment="natural" lineBreakMode="tailTruncation" baselineAdjustment="alignBaselines" adjustsFontSizeToFit="NO" translatesAutoresizingMaskIntoConstraints="NO" id="p07-
            s6-PTR">
            <rect key="frame" x="128" y="29" width="119" height="21"/>
        </label>
        <textField opaque="NO" clipsSubviews="YES" contentMode="scaleToFill" fixedFrame="YES" contentHorizontalAlignment="left" contentVerticalAlignment="center"
            borderStyle="roundedRect" placeholder="email" textAlignment="natural" minimumFontSize="17" translatesAutoresizingMaskIntoConstraints="NO" id="eWp-9U-F62">
            <rect key="frame" x="37" y="200" width="302" height="30"/>
        </textField>
        <textField opaque="NO" clipsSubviews="YES" contentMode="scaleToFill" fixedFrame="YES" contentHorizontalAlignment="left" contentVerticalAlignment="center"
            borderStyle="roundedRect" placeholder="password" textAlignment="natural" minimumFontSize="17" translatesAutoresizingMaskIntoConstraints="NO" id="03x-5u-Nlv">
            <rect key="frame" x="37" y="249" width="304" height="30"/>
        </textField>
        <textField opaque="NO" clipsSubviews="YES" contentMode="scaleToFill" fixedFrame="YES" contentHorizontalAlignment="left" contentVerticalAlignment="center"
            borderStyle="roundedRect" placeholder="confirm password" textAlignment="natural" minimumFontSize="17" translatesAutoresizingMaskIntoConstraints="NO" id="D1Q-1P-ETI">
            <rect key="frame" x="36" y="297" width="304" height="30"/>
            <textInputTraits key="textInputTraits" secureTextEntry="YES"/>
        </textField>
        <switch opaque="NO" contentMode="scaleToFill" horizontalHuggingPriority="750" verticalHuggingPriority="750" fixedFrame="YES" contentHorizontalAlignment="center"
            contentVerticalAlignment="center" on="YES" translatesAutoresizingMaskIntoConstraints="NO" id="bi3-wu-tDQ">
            <rect key="frame" x="149" y="364" width="51" height="31"/>
        </switch>
        <label opaque="NO" userInteractionEnabled="NO" contentMode="left" horizontalHuggingPriority="251" verticalHuggingPriority="251" fixedFrame="YES" text="I am male"
            textAlignment="natural" lineBreakMode="tailTruncation" baselineAdjustment="alignBaselines" adjustsFontSizeToFit="NO" translatesAutoresizingMaskIntoConstraints="NO" id="zBM-
            j8-Oei">
            <rect key="frame" x="37" y="369" width="84" height="21"/>
        </label>
    </subviews>
    <color key="backgroundColor" red="1" green="1" blue="1" alpha="1" colorSpace="custom" customColorSpace="sRGB"/>
</view>
```

Fig. 9: The XML code of the iOS GUI

```
<?xml version="1.0" encoding="UTF-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android" xmlns:tools="http://schemas.android.com/tools" android:
    layout_width="match_parent" android:layout_height="match_parent">
    <TextView android:id="@+id/textView2" android:layout_width="wrap_content" android:layout_height="wrap_content"
        android:layout_marginLeft="128dp" android:layout_marginTop="29dp" android:text="Create Account" android:textSize="17sp" />
    <EditText android:id="@+id/editText" android:layout_width="302dp" android:layout_height="wrap_content" android:layout_marginLeft="37dp"
        android:layout_marginTop="100dp" android:ems="10" android:hint="first name" android:inputType="textPersonName" />
    <EditText android:id="@+id/editText2" android:layout_width="302dp" android:layout_height="wrap_content" android:layout_marginLeft="37dp"
        android:layout_marginTop="150dp" android:ems="10" android:hint="last name" android:inputType="textPersonName" />
    <EditText android:id="@+id/editText3" android:layout_width="302dp" android:layout_height="wrap_content" android:layout_marginLeft="37dp"
        android:layout_marginTop="200dp" android:ems="10" android:hint="email" android:inputType="textPersonName" />
    <EditText android:id="@+id/editText4" android:layout_width="302dp" android:layout_height="wrap_content" android:layout_marginLeft="37dp"
        android:layout_marginTop="250dp" android:ems="10" android:hint="password" android:inputType="textPassword" />
    <EditText android:id="@+id/editText5" android:layout_width="302dp" android:layout_height="wrap_content" android:layout_marginLeft="37dp"
        android:layout_marginTop="300dp" android:ems="10" android:hint="confirm password" android:inputType="textPassword" />
    <TextView android:id="@+id/textView" android:layout_width="302dp" android:layout_height="wrap_content" android:layout_marginLeft="37dp"
        android:layout_marginTop="369dp" android:text="I am male" />
    <Switch android:id="@+id/switch1" android:layout_width="wrap_content" android:layout_height="wrap_content"
        android:layout_marginLeft="149dp" android:layout_marginTop="364dp" tools:checked="true" />
    <Button android:id="@+id/button" android:layout_width="304dp" android:layout_height="34dp" android:layout_marginLeft="21dp" android:
        layout_marginTop="456dp" android:text="create" />
</RelativeLayout>
```

Fig. 10: The XML code of the Android GUI generated by our framework

account" page in Figure 8. The source code of this page is shown in Figure 11. When, we have developed the Android version of the page, we have not imposed any constraints on the developer on how to build the page. We have allowed the developer to build the page freely to have a native Android page. We can see, in this figure, that the widgets of the page are positioned relative to each other. For example, the editText "first name" with id="@+id/editText" is below the textView (layout_below="@+id/textView"). Furthermore, it has a *marginTop* and its left edge matches the left edge of the "last name" edit text.

When using our approach to generate the iOS version of the page, we get the same page presented in Figure 7. This is possible due to the "compute absolute positions" phase of our approach that is presented in Section 4, Algorithm 2 and that translates the relative positions of the widgets into absolute positions with specific x and y values.

It is worth to mention here that the generation of the XML files for android and iOS using our tool conforms the XSD android and iOS formats respectively. Therefore, in order to run or to view the generated user interfaces we need simply to import the generated xml files into the *layout* folder of an android project if we are porting an application from iOS to android and run the project. If we are targeting iOS from android, one xml file called *Main.storyboard* will be generated that aggregates all the

```
<?xml version="1.0" encoding="UTF-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android" android:layout_width="match_parent" android:layout_height="match_parent">
    <TextView android:id="@+id/textView" android:layout_width="wrap_content" android:layout_height="wrap_content"
        android:layout_alignParentTop="true" android:layout_centerHorizontal="true" android:layout_marginTop="29dp" android:text="Create account"
        android:textSize="18sp" />
    <EditText android:id="@+id/editText" android:layout_width="300dp" android:layout_height="wrap_content" android:layout_marginTop="20dp" android:
        ems="10" android:hint="first name" android:inputType="textPersonName" android:layout_below="@+id/textView" android:layout_alignLeft="@+id/
        editText2" android:layout_alignStart="@+id/editText2" />
    <EditText android:id="@+id/editText2" android:layout_width="300dp" android:layout_height="wrap_content" android:layout_marginTop="20dp" android:
        ems="10" android:hint="last name" android:inputType="textPersonName" android:layout_below="@+id/editText" android:layout_alignParentLeft=
        "true" android:layout_alignParentStart="true" android:layout_marginLeft="28dp" android:layout_marginStart="28dp" />
    <EditText android:id="@+id/editText3" android:layout_width="300dp" android:layout_height="wrap_content" android:layout_alignRight="@+id/
        editText2" android:layout_below="@+id/editText2" android:layout_marginTop="20dp" android:ems="10" android:hint="email" android:inputType=
        "textPersonName" android:layout_alignLeft="@+id/editText2" />
    <EditText android:id="@+id/editText4" android:layout_width="300dp" android:layout_height="wrap_content" android:layout_alignEnd="@+id/editText3"
        android:layout_alignRight="@+id/editText3" android:layout_below="@+id/editText3" android:layout_marginTop="20dp" android:ems="10" android:
        hint="password" android:inputType="textPersonName" />
    <EditText android:id="@+id/editText5" android:layout_width="300dp" android:layout_height="wrap_content" android:layout_alignEnd="@+id/editText4"
        android:layout_alignRight="@+id/editText4" android:layout_below="@+id/editText4" android:layout_marginTop="20dp" android:ems="10" android:
        hint="confirm password" android:inputType="textPersonName" />
    <TextView android:id="@+id/textView2" android:layout_width="wrap_content" android:layout_height="wrap_content" android:layout_alignLeft="@+id/
        editText5" android:layout_alignStart="@+id/editText5" android:layout_below="@+id/editText5" android:layout_marginTop="20dp" android:text="I
        am male" />
    <Switch android:id="@+id/switch1" android:layout_width="wrap_content" android:layout_height="wrap_content" android:layout_alignBaseline="@+id/
        textView2" android:layout_alignBottom="@+id/textView2" android:layout_centerHorizontal="true" />
    <Button android:id="@+id/button" android:layout_width="wrap_content" android:layout_height="wrap_content" android:layout_below="@+id/switch1"
        android:layout_centerHorizontal="true" android:layout_marginTop="20dp" android:text="Create" />
</RelativeLayout>
```

Fig. 11: The XML code of the Android GUI created by the developer

generated user interfaces. This file is then imported into an iOS project to be run.

5 Related Work

No related work has been identified in the literature review that uses model driven engineering to automatically and directly port native mobile applications user interfaces from one platform to another. However, all the existing approaches require the use of a third party tool (instead of the native android and iOS platforms) to model and generate the user interface. These approaches can be classified into cross platform approaches and model-driven development approaches [7].

Cross Platform Approaches

Cross-platform approaches fall into four different categories [8]: mobile Web app, hybrid apps, interpreted environments, or generative approaches. Web apps are built with Web technologies (HTML, CSS, and JavaScript) and are accessed via mobile browsers. They lack access to device-specific features, except for a limited set that is made available with HTML5 [9]. Hybrid apps, as the ones created with Apache Cordova [10] (formerly known as PhoneGap), package a Web site with a native component that provides access to device features. Both Web apps and hybrid apps look and behave like Web sites, because the browser engine is responsible for rendering the UI.

Interpreted apps that are built using approaches such as Titanium [11] uses a separate runtime environment. They are, in principle, able to build up the UI with native components. On each target platform, an interpreter at runtime interprets the source code of the app written in a scripting language.

Generative approaches create completely native apps out of a common code base. They are based on the implementation of mobile applications by using the common programming languages. The developers can write the source code in a common programming language and the cross compiler compiles the source code into a particular native code by converting the source code into native binaries. Examples of tools based on this approach include Xamarin [12] that uses C# as the common programming language.

The cross platform approaches can be used to target multiple platforms. The user interface in these approaches is built using programming languages such as Javascript or C#. They offer new development environments instead of using the existing native environments such as Android studio, Xcode or Visual studio. Proposing new environments means, on one hand, additional cost in terms of maintainability and portability for the cross platform framework providers and, on the other hand, the application developers are forced to learn new technologies. While, in our proposed approach, the application designer builds the user interface using the native environment that he / she is familiar with (i.e., Android studio, Xcode or Visual studio). Then using our tool, and without any extra effort from the developer's side, the user interface is generated for the specified target platform.

Model-Driven Development Approaches

Many model-driven frameworks have been proposed for cross-platform application development. Previous work in [13,14,15,16,17,18] focuses on mobile application development, while other previous work [19,20,21] focuses on the development of real-time embedded systems.

Heitkotter et al. presented a framework [13] called MD2 which allows to describe the application in a platform-independent model using a textual domain specific language. The language allows also the

description of the data model, the views and the behavior of the mobile application. A code generator specific for each target platform is used to transform the application model into source code. The MD2 framework provides in addition a code generator which creates a server-backend based on the data model of the application.

Usman et al. presented a model-driven development approach [14] to generate mobile apps for multiple platforms based on UML. The application requirements are modeled through use-case diagrams, UML class diagrams are used to model the structure of the app while UML state machine diagrams are used for behavioral modeling. In this approach, they have focused on the business logic code of the app and they did not deal with the user interface development.

JUSE4Android [15] is a model-driven tool that allows the automatic generation of Business Information Systems (BIS) for Android. The apps are specified through annotated UML class diagrams from which the running code is generated.

The model-driven *applause* [16,17] provides a DSL for creating cross platform applications. The framework *applause* is based on a DSL to describe mobile apps and a set of code generators for iOS, Android, Windows Phone 7 and Google App Engine.

AXIOM [18] is a model-driven approach for the development of cross platform mobile apps. In AXIOM, the requirements of the application are first described in platform-independent intent models (interaction and domain perspective) using AXIOM's DSL. Those intent models are then enriched with structural decisions and refined with platform-specific elements during a multi-phase transformation process to produce the source code for native apps: from requirements models to Platform Independent Model (PIM), from PIM to Platform Specific Models (PSMs), and finally from PSMs to running code.

Generic Application Migration Approaches

A generic approach for migrating real-time embedded applications was proposed [19] and it was extended in [20] and [21] to support code generation and verification. It is based on the SRM profile [6] which allows the description of platform resources in a common way. As a result generic transformations are proposed to migrate real-time embedded applications from one platform to another. Although this approach provides a generic and effective way to migrate real-time embedded applications, it could not be applied to migrate mobile applications as the resources of the mobile application operating systems differ from real-time operating systems. In fact, mobile application platform resources cannot be modeled using the SRM profile.

All the presented MDD approaches start by describing the mobile application model in a platform independent way (PIM). Each approach relies on its own DSL, either defined from scratch or by using UML profiles. Then, in order to generate executable code, they use code generators specific for each target platform. The user interface of the app is also described using DSLs. So, in order to build a user interface, the application developer must learn how to use this new DSL. In addition, the DSL must be rich enough to describe all the user interface properties. And this is usually not the case; the proposed DSLs lack the richness of the native development environments (*i.e.*, Android studio and Xcode). Moreover, as these approaches use specific code generators, this means they are not portable. So, if a new platform is introduced, new code generators must be developed which require from the tool-chain provider dual skills in the new platform and in model driven engineering.

Our approach tackles all the above concerns. First, the developer does not need to learn new modeling techniques to model the user interfaces. In fact, the developer does not need to model the user interfaces but he/she will build them using the native environment that he/she is familiar with. Second, our mapping technique is not based on generators specialized to a specific platform, but on a generic profile that links resources of different platforms together. Therefore, any introduction of a new platform does not require the development of a new generator, but the addition of the Platform Description Model (PDM) that we recommend to be designed by the platform provider given their high knowledge in their platform. We note, however; that our current tool deals only with porting user interfaces from one platform to the other. It does not yet support the transition between user interfaces which are one of our future work.

6 Perspective

In this paper, we have described a new model-driven framework for the development of mobile applications. Our goal is to offer a development process that incorporates systematic strategies in order to achieve portability. We have sought to achieve portability by relying on the native development processes such as Android Studio and iOS XCode. Hence, the application developer is not forced to learn a new technology as it is the case with the approaches that have been previously suggested to achieve portability.

Our proposed approach is based on a common description of the mobile application platforms using the UML profile MRM that we have introduced. Then, based on this common description, we have proposed a generic transformation framework that takes the mobile application user interface implemented on a specific platform and generates the equivalent user interface on the target platform. Our solution is general and can be easily extended to incorporate new resources as well as target platforms beyond iOS and Android.

We have demonstrated our approach using a case study where we have successfully ported a mobile application

user interface from iOS to Android and vice versa with relatively no cost.

Future research will focus on extending our framework to support porting complex user interfaces then porting complete mobile applications. This will be done by extending the platform description model that was presented in this paper to model behavioral features of the mobile resources.

Acknowledgement

The authors are grateful to the anonymous referees for a careful checking of the details and for helpful comments that improved this paper.

References

[1] Mobile App Usage. Report by Statista.
[2] Object Management Group, MDA Guide Revision 2.0, June 2014, OMG document number: ormsc/14-06-01.
[3] Object Management Group, OMG Unified Modeling Language (OMG UML), Superstructure, V2.5, March 2015, OMG document number: formal/15-03-01.
[4] E. Umuhoza and M. Brambilla, Model driven development approaches for mobile applications: A survey, International Conference on Mobile Web and Information Systems. pp. 93 - 107, (2016).
[5] B. Myers and M. Rosson, Survey on user interface programming, Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, pp. 195-202 (1992).
[6] Object Management Group, A UML profile for MARTE (V1.1), June 2011, OMG document number: formal/2011-06-02.
[7] A. Charland and B. Leroux, Mobile application development: web vs. native, Communications of the ACM, Vol. 54, No. 5, pp. 49 - 53 (2011).
[8] H. Heitkotter, S. Hanschke and T. A. Majchrzak, Evaluating cross-platform development approaches for mobile applications, International Conference on Web Information Systems and Technologies, pp. 120 - 138, (2012).
[9] A. Boehm, Z. Ruvalcaba, Murach's HTML5 and CSS3, Mike Murach & Associates, 4th edition.
[10] Apache Cordova, 2015. Apache Cordova.
[11] Appcelerator Titanium, 2017. Appcelerator Titanium.
[12] Xamarin, 2019. Xamarin.
[13] H. Heitkotter, T. A. Majchrzak and H. Kuchen, Cross-platform model-driven development of mobile applications with md2, Proceedings of the 28th Annual ACM Symposium on Applied Computing, pp. 526 - 533 (2013).
[14] M. Usman, Z. Iqbal and M. Khan, A model-driven approach to generate mobile applications for multiple platforms, Proceedings of the 2014 21st Asia-Pacific Software Engineering Conference, Vol. 2, pp. 111 - 118 (2014).
[15] L. P. Da Silva and F. Brito e Abreu, Model-driven GUI generation and navigation for android BIS apps, 2nd International Conference on Model-Driven Engineering and Software Development, pp. 400-407 (2014).
[16] H. Behrens, MDSD for the iPhone: developing a domain-specific language and IDE tooling to produce real world applications for mobile devices, Proceedings of the ACM international conference companion on Object oriented programming systems languages and applications companion, pp. 123 - 128 (2010).
[17] R. Fielding and R. Taylor, Principled design of the modern web architecture, ACM Transactions on Internet Technology, Vol. 2, No. 2, pp. 115-150 (2002).
[18] X. Jia and C. Jones, AXIOM: A model- driven approach to cross-platform application development, ICSOFT Communications in Computer and Information, Vol. 1, pp. 24 - 33 (2012).
[19] F. Thomas, J. Delatour, F. Terrier and S. Gerard, Towards a framework for explicit platform-based transformations, 11th IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing (ISORC), pp. 211-218 (2008).
[20] W. E. H. Chehade, A. Radermacher, F. Terrier, B. Selic and S. Gerard, A model-driven framework for the development of portable real-time embedded systems, 16th IEEE International Conference on Engineering of Complex Computer Systems, pp. 45-54 (2011).
[21] W. E. H. Chehade and R. Abdel Kader, A Model-Driven Approach for the Validation of RTOS Constraints in Real-time Application Models, International Journal of Applied Engineering Research, Vol. 12, No. 5, pp. 622-631 (2017).

Riham Abdel Kader received her Bachelor and Masters degrees in Computer Science from the American University of Beirut in Lebanon in respectively 2003 and 2006, and her PhD degree in the optimization of XQueries in the context of relational database systems from the University of Twente in The Netherlands in 2010. She then worked as a Senior Software Engineer at ASML, a leading international company in the lithography industry. Since 2014, she is an assistant professor at the Faculty of Science at Beirut Arab University. Her research interests are in model driven engineering, image processing, and database management systems.

Wassim El Hajj Chehade received his Bachelor degree in Computer and Communication Engineering from the Lebanese University and his PhD degree in Computer Science from Paris XI University in 2011 where he worked at the Laboratory of Model Driven Engineering for Embedded Systems at CEA LIST. After obtaining his PhD, he worked at PodBox, a start-up company based in Brittany, France. Currently, he is an assistant professor at the Faculty of Science at Beirut Arab University since 2014. His research interests are in image processing, model driven engineering, real-time embedded systems and mobile application development.