IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received February 27, 2021, accepted March 10, 2021, date of publication March 15, 2021, date of current version April 8, 2021.

Digital Object Identifier 10.1109/ACCESS.2021.3066007

An Automated Model-Based Approach for Developing Mobile User Interfaces

LASSAAD BEN AMMAR
College of Sciences and Humanities, Prince Sattam bin Abdulaziz University, Al-Kharj 11942, Saudi Arabia
Department of Computer Science, University of Sfax, Sfax 3029, Tunisia

e-mail: l.benammar@psau.edu.sa

ABSTRACT The ever-increasing number of mobile platforms constitutes a challenge for application developers, who must develop efficient mobile applications for multiple platforms. Recently, a specific interest is being taken in the Model Based User Interface Development (MBUID) by Software Engineering Community. In such paradigm, an application's user interface is obtained by defining models and transformations of those models. This paper aim at adopting MBUID paradigm to address the problem of mobile application development. As such, we propose a new approach and its support system for the automatic generation of mobile user interfaces. The approach and the system are based on a set of standards and relevant technologies such as EMF, GMF, ATL, and Xpand. A case study is presented, in the paper, with the aim of verifying the usefulness of this approach.

INDEX TERMS Mobile application, user interface, model based user interface development, model transformation.

I. INTRODUCTION

Nowadays, the ubiquity and widespread of mobiles devices lead to an intensive use of mobile applications that run on different mobile platforms (Android, iOS, Symbian, ...). Such a diversity in mobile platforms makes mobile application development a difficult task for different reasons. On the one hand, an important factor that could lead to the success of a mobile application is its ability to meet different platforms' requirements. On the other hand, developing the same application for different platform causes a wast of time and resources. The real challenge that envisage software engineers is cross platform mobile development. A possible solution for this issue could take the advantages of Model-driven Engineering (MDE) [1].

In recent years, some research focuses on the adoption of Model-driven Engineering (MDE) [1] in the development of cross platform mobile application [2]. Indeed, this field of software engineering could benefit from MDE that is known for its ability to automate the generation of source codes, in any programming language, from conceptual models. Thanks to MDE, the development process starts by a set of conceptual models that represent the system independently of the target platform. Such an independence from the platform will thereby maximise the reuse of the conceptual models to generate applications that meet various platforms' requirements. During an MDE-compliant development process, relevant features of the implementation and/or the platform could be introduced using Model-to-Model (M2M) and Model-to-text (M2T) transformations.

In the context of MDE, Model-Based User Interface Development (MBUID) [3] addresses the development of Graphical User Interfaces (GUI) based on conceptual models and transformations between these models. The interest of MBUID is focused on GUI since it is a time-consuming task that could impact the development-cost features. In addition, GUI is usually the main reason that may lead to the success/failure of the software application.

In the literature, several researches focus on the adoption of MBUID to address GUI development for cross platform application ([4]–[7]). The objective is take advantage of MBUID in terms of model reuse and automatic generation of application source codes that could meet the requirements of several platforms.

The present paper aims to delineate with MBUID for mobile User Interfaces (UI) development. The approach proposes 3 meta-models called Abstract User Interface (AUI), Concrete User Interface (CUI) and Final User Interface (FUI). On top of these meta-models, the approach proposes a set of transformations allowing the automatic generation

The associate editor coordinating the review of this manuscript and approving it for publication was Francisco J. Garcia-Penalvo.

of GUI for a mobile application starting from a model that is conform to the AUI. It is worth mentioning that the proposed approach enable the generation of GUI for cross platform mobile application. In terms of technical details, the proposed approach is supported by a system that is developed under Eclipse IDE for Java Developers[1] and takes advantage of relevant technologies in the field of MBDUI such as Eclipse Modeling Framework (EMF)[2] and Graphical Modeling Framework (GMF).[3] In terms of programming languages, the system is developed using ATLAS Transformation Language (ATL)[4] as an M2M transformation language and Xpand[5] as an M2T transformation language.

The main advantages of our approach with regard to the state of the art can be summarized in the following:

1) A development process aligned with the principles of MBUID;
2) A uniform meta-model for each intermediate artifact reducing thereby the problem of variety of concepts and terminologies presented by others research works;
3) A full-automated transformation process implemented using relevant technologies in the field of MBUID such as MOF, EMF, ATL, and Xpand.

The rest of this paper is organised as follows. Section II presents an overview of MBUID and discusses some related works. Section III details the proposed approach for mobile user interface generation. Section IV illustrates the proposed approach through a case study. Finally, Section V concludes the paper and gives some directions for future works.

II. FOUNDATIONS AND RELATED WORKS

This section presents a brief overview of MBUID concepts as well as the main contributions in the field of mobile user interface development with model-based approach.

A. MODEL-BASED USER INTERFACE DEVELOPMENT: MBUID

In the context of MBUID, declarative models can be constructed and related, as part of the user interface design process [8]. The aim was to permit developers to design and implement user interfaces in a systematic and professional way. A lot of efforts have been established to capture conceptually the most important parts of the development process. These efforts achieved a level of maturity leading to the development of a reference framework that structures the development process in four levels of abstraction [3]:

- At the first level of abstraction, the Task & Concepts model of the application is usually proposed to describe the various user tasks to be carried out and the domain-oriented concepts required by these tasks.
- In the second level, the AUI model is specified. At this level, an abstract definition of the content and the structure of the user interface is provided.
- In the third level, the CUI model is specified. At this level, the user interface is modeled using concrete interactors. It defines how the user interface is perceived and can be manipulated by end users using widgets and interface navigation. The CIU is modality-dependent, but implementation technology independent.
- In the last level, the FUI model is specified. It represents the source code of the user interface in any implementation technology (programming language like java, markup language like HTML, etc.)

On top of these abstraction levels, 3 main relationships are defined. 1) Reification covers the inference process from high-level abstract descriptions to runtime code; 2) Abstraction maps a user interface representation from one level of abstraction to a higher level of abstraction; 3) Translation transforms a description intended for a particular target into a description at the same abstraction level but aimed at a different target.

B. MODEL-BASED APPROACHES FOR MOBILE APPLICATIONS

This section examines model-driven approaches for the development of mobile applications with an aim at providing the potentialities and limitations of the current state of the art. We consider only those approaches that apply model-driven techniques to specify mobile user interfaces. However, approaches that address non functional requirements are out of the scope of this paper.

Our literature review reveals that the adoption of model driven approaches for the development of mobile application has gained a special attention by the SE community. Indeed, in recent years, there are some initiatives focusing on this research area and providing promising results. However, more efforts are still required to take the advantages of Model-driven Engineering in the context of mobile application development.

The recent Systematic Literature Reviews (SLR) presented in [9] and [10] prove the shortage of research works in this field. In [2], a survey of model-driven approaches for the development of mobile applications discusses only 13 research works and 4 commercial solutions. These 3 references are interesting and helpful to identify the most important related works in the context of model-driven mobile development. The works in this context can be divided into two different clusters. On the one hand, we consider researches that apply model driven techniques for the purpose of modelling native app (mobile apps for a specific operating system). On the other hand, we encounter some researches dealing with cross-platform app (mobile apps that work across multiple platforms). In what follow, we present some of the most referenced researches in the context of model-based development for mobile applications.

[1] www.eclipse.org/downloads/packages.
[2] https://www.eclipse.org/modeling/emf/
[3] https://www.eclipse.org/gmf-tooling/
[4] https://www.eclipse.org/atl/
[5] https://www.eclipse.org/modeling/m2t/?project=xpand

In [11], a model-driven framework is presented for the development of a cross-platform business applications. The application is presented independently from the platform through a textual Domain Specific Language (DSL). Then, a code generator transforms this representation into source code for a specific platform. Finally, the developers need to compile the generated source code using the corresponding development environment such as the Android Developer Tools or Xcode.

In [12], the authors propose a model-driven approach for the development of native mobile applications with a focus on the data layer. The development process starts by defining meta-models and Architecture-Specific Model (ASM) profiles. Then, a series of transformation are applied to generate an Android and/or Windows Phone applications allowing the applications' functionalities to be tested.

In [13], a model driven approach to automatically generate mobile applications for various platforms is presented. In such an approach, UML diagrams are used to model the applications: use case for the requirements modeling, class diagram for the structural modeling and state machine for the behavioral modeling. A UML profile is used to include the mobile domain specific concepts. The code generator allows developers to generate the business logic code while the GUI is developed separately.

In [14], the authors propose an Agile method to transform an abstract model tree into final code. Applications requirements are described in platform independent models using a textual DSL. Then, they are augmented with structural decisions and refined with other platform-specific elements during a multi-phase transformation process to produce source code for native applications.

Another interesting initiative in this context is presented in [15] where the application's business logic is modeled as a UML class diagram using a graphical tool called JBModel. Platform specific details may be added via annotation before executing a model-to-text transformation to generate the code of each class. This later will be the input for the JustBusiness framework allowing the generation of the user interfaces, the persistence code and the application resources. Developers have to manually implement the method's body.

Based on the analysis of the aforementioned research works, it becomes clear that the development of mobile applications using model-based approach still an immature area and many more efforts are needed. In fact, most of researches are based on their own DSL rather than use an agreed one. In addition, the variety of terminologies and tools used by these researches make it difficult to select one of them to adopt/adapt it. We argue that the main objectives of the future works in this domain should be the unification of the concepts, terminologies and tools to be used by researchers. In addition, the conformity to the MDE standard is required to facilitate the adoption of the approach. This work falls in this orientation.

III. PROPOSED APPROACH FOR MOBILE USER INTERFACE DEVELOPMENT

This section presents an overview of the proposed approach for model-based mobile user interface generation prior to detailing its different stages.

A. OVERVIEW

As per Figure 1, our approach for model-based mobile user interface generation is made up of 2 major stages:

1) Meta-models definition: during this stages the AUI, CUI, and FUI meta-models are defined. These meta-models cover the different levels of abstraction required to adopt an MBUID-based approach. Section III-B discusses these meta-models.
2) Transformation process: during this stage, the transformations from AUI to CUI, CUI to FUI, and FUI to source code are conducted. These transformations are aligned with the MBUID paradigm. They allow to generate a mobile user interface from any model that is conform to the already defined AUI meta-model. The different transformations are discussed in Section III-C.

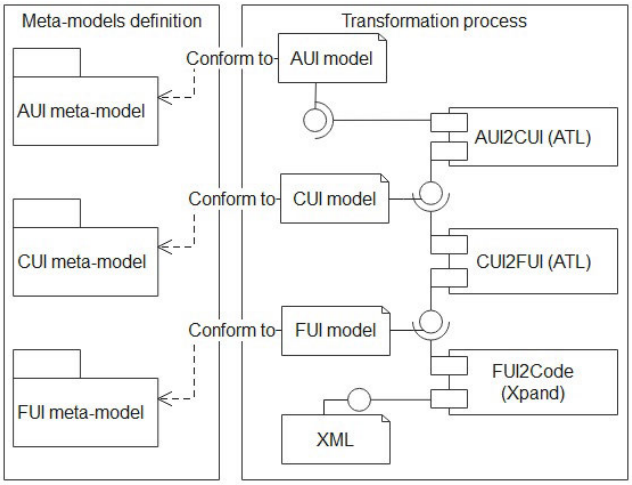

FIGURE 1. Overview of our proposal.

B. META-MODELS DEFINITION

The first stage of our approach is the definition of the required meta-model. In this context, we define 3 meta-models that constitute the basis of our approach.

1) ABSTRACT USER INTERFACE META-MODEL

In a previous work [16], we proposed a uniform AUI meta-model that represents the basic concepts to be used while specifying a user interface in an abstract way. According to this meta-model, a user interface is usually populated by *Abstract Interaction Object (AIO)* and *Abstract Relationship* as per Figure 2.

AIO represents an abstraction of widgets that can be found in popular GUI toolkit (e.g. windows, buttons, panels, etc.). AIO have been classified into two main types:

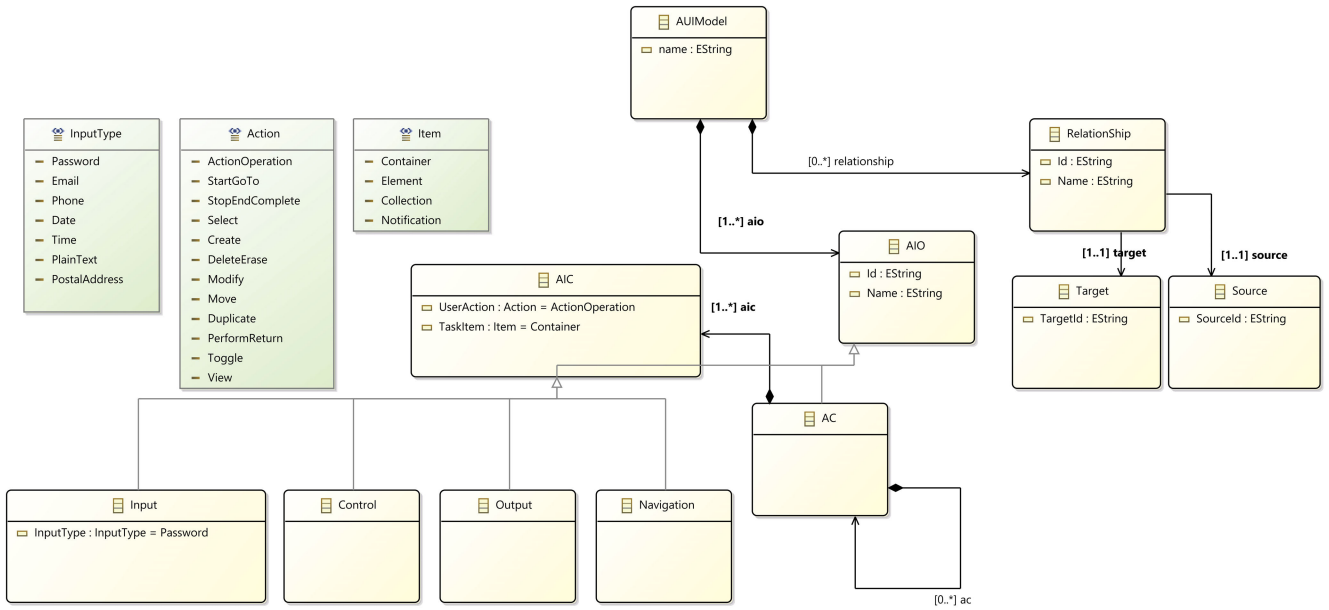

FIGURE 2. Abstract user interface (AUI) meta-model.

- *Abstract containers (AC)*: called sometimes presentation unit and represent an entity that may gather other abstract containers or components. Such interaction object is intended to support the representation of a logically/semantically connected tasks. Window, dialog box and panel are, among others, example of the rendering of such entity in the concrete level.
- *Abstract individual components (AIC)*: represent an abstraction of an interaction object that usually populate an abstract container. This component can be transformed into the concrete level as text field, button, drop down menu, etc.

Abstract relationships are relationships that can be established between abstract interaction objects of all kinds. They are couples of Source and Target. Abstract Adjacency and Spatio-temporal are among the most common type of abstract relationships presented in the literature. Adjacency specifies an adjacency constraint between 2 AIOs. As for Spatio-temporal relationship, 2 basic relations are considered in our approach: *sequential* and *simultaneous*.

2) CONCRETE USER INTERFACE META-MODEL

The CUI meta-model presents the basic concepts and terminologies allowing a specification of an appearance and behavior of a user interface with elements that can be perceived by users. A syntactical uniform process was carried out in order to omit redundant terminologies/vocabularies and select those that are largely considered in user interface modelling literature. Figure 3 illustrates our CUI meta-model. A CUI model is usually populated by *Concrete Interaction Objects (CIO)* that can be either *Concrete Containers (CC)* or *Concrete Interaction Components (CIC)*.

Concrete Containers are generally window, panel, menu bar, tabbed dialog box. *Concrete Interaction Component*, called sometimes interactors, are usually classified into three main categories: control, data and multimedia. The control category is represented by the *Button* class which gather divers type of button (button, switch, toggle button, etc.). The data category is represented by the *TextComponent* class which may be plain text, password text, email text, etc. The multimedia category may represent an image view, an audio or video component. In the context of mobile application, a fourth type of interactor may be useful: *ComposedComponent*. This class of interactor group more than one facet. A facet is defined as the function that the component may endorse in the physical world. Date picker and color picker are example among others of such component.

3) FINAL USER INTERFACE META-MODEL

The FUI meta-model specifies the main concepts and terminologies needed to render a CUI in a specific technological platform.

In this paper, the interest is focused on the Android OS. This choice can be justified by the great adoption of such OS in the mobile market share worldwide in the last few years; around 75% [17].

Figure 4 illustrates the proposed meta-model for a GUI for Android application. An android application is usually composed of one or more screens defined in the application as Activities. Each Activity is populated by one or more containers and one or more widgets. The main container in a screen is called *Layout*. Three main type of layout can be identified in a mobile application: *FrameLayout*, *LinearLayout* and *TableLayout*. Each type of layout is described using

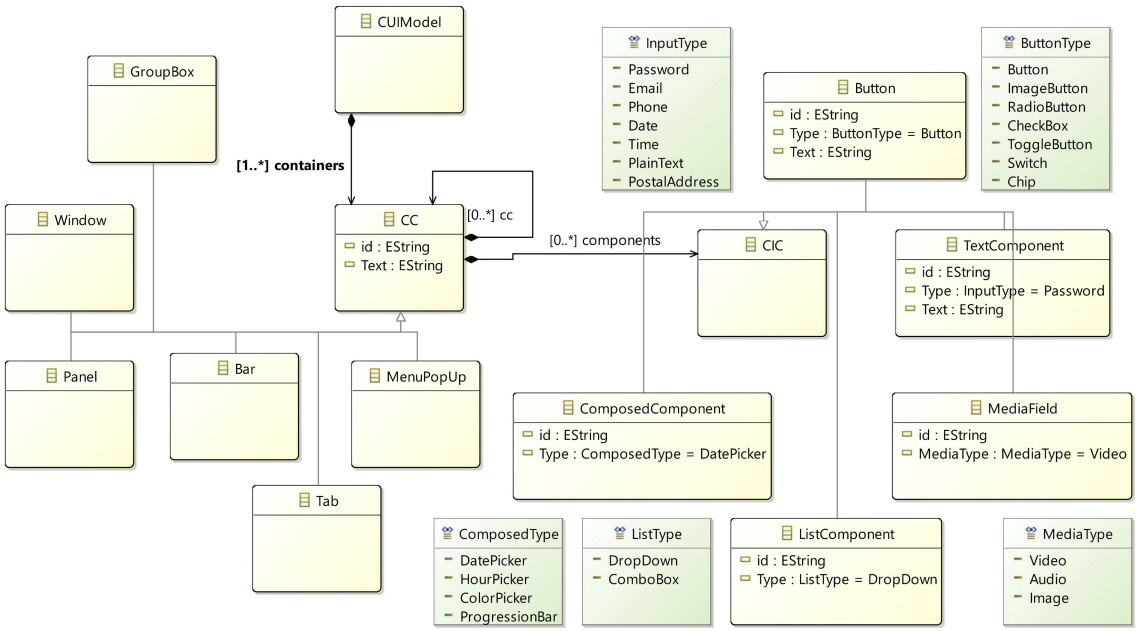

FIGURE 3. Concrete user interface (CUI) meta-model.

three main attribute: *id*, *layout_width* and *layout_height*. The *LinearLayout* may require and additional attribute describing the orientation (vertical or horizontal). The Widget class represent all interaction objects that can be presented in a mobile application such as button, edit text, image view, progress bar, etc. Each interaction object is described at least by three main attributes: *id*, *layout_width* and *layout_height*. Some widgets may need others additional attributes. For example, the Text class, which represent all possible type of text field required to input data, use another attribute called *InputType* to distinguish the type of data to be entered: password, phone, email, address, etc.

C. TRANSFORMATION PROCESSES

The second stage of our approach is about a set of transformations that permit the automatic generation of executable user interfaces. As has been previously explained, the ATL and Xpand languages have been used for the implementation of the transformation processes. ATL was used to implement Model-to-Model (M2M) transformation: AUI2CUI and CUI2FUI. The former generates the CUI model from the abstract model. The latter considers the obtained CUI model to generate the FUI model. While Xpand, it is used to implement the Model-to-Text (M2T) transformation that generates the source code of the FUI.

1) STEP 1. CUI'S AUTOMATIC GENERATOR

In this step, a CUI model is generated via an M2M transformation from the AUI. This step consists of defining the set of transformation rules allowing to establish the correspondences between elements from the AUI meta-model (source) and those from the CUI meta-model (target).

An AUI model is usually structured as a tree. The root and the intermediate nodes represent abstract containers while the leaves refer to abstract components. To transform an abstract model, we opted for a top-down process to traverse its tree and transform each node: (i) the root node will be rendered in the target model as a Window; (ii) an intermediate node will be rendered as a Panel; (iii) a leaf will be rendered as a Concrete Component among those presented in the concrete meta-model. As there are various types of concrete component, we propose 3 main criteria to guide the selection of the appropriate one:

1) The **Facet**: a function that the component may endorse in the physical world. It may be Input, Output, Control or Navigation.
2) The **UserAction** attribute: indicates the user action that is required for performing the task. For examples, start and select respectively indicate that an action is triggered and a selection between multiples items is required.
3) The **TaskItem** attribute: refers to the type of the object on which the action is operated. For example, element specifies that the item has a single characteristic, container specifies that the item is an aggregation of elements.

All the aforementioned transformations are ensured thanks to a set of transformation rules that we defined based on the AUI and CUI meta-models. Some of them are presented in what follow:

Rule 1: the root node, which is usually an abstract container, will be transformed into a Window.

Rule 2: each intermediate node, which is also an abstract container, will be transformed into a Panel.

Rule 3: each abstract component with the Input facet will be transformed into a TextComponent. The selection of the appropriate text field's type (password, email, phone, etc.)

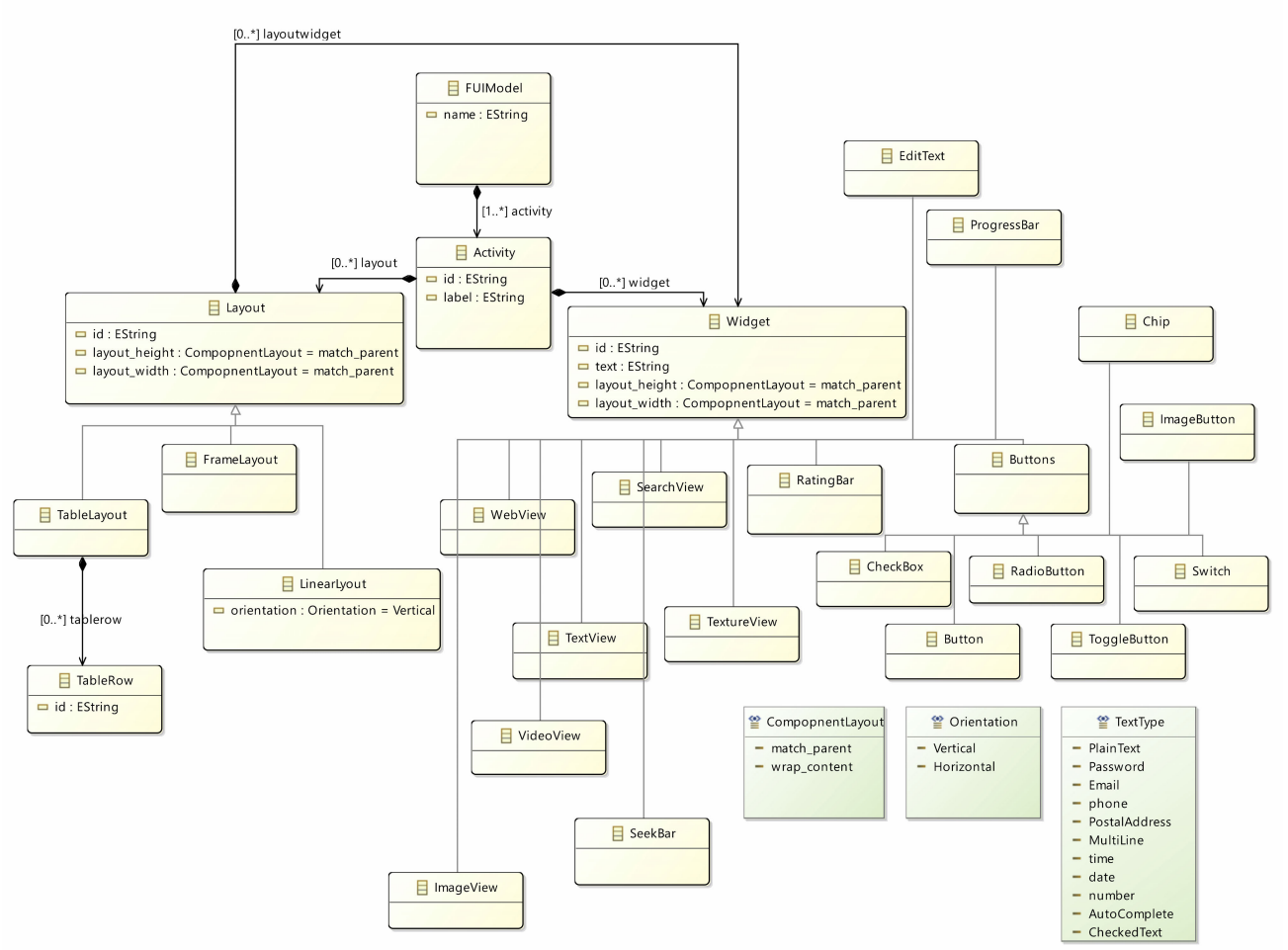

FIGURE 4. Final user interface (FUI) meta-model.

depends on the InputType attribute of the abstract component.

Rule 4: each abstract component with the Control facet will be transformed into a Button. The selection of the appropriate button's type (image button, toggle button, radio button, etc.) depends on the UserAction and TaskItem attributes of the abstract component.

To better clear up our proposal regarding the selection of the concrete component, we present afterhere an algorithm that illustrates the selection of the appropriate concrete component for an abstract element form the Input facet.

In terms of technical details, we have implemented the transformation rules, established during this step, using ATL language.

Figure 5 shows the ATL code implementing the first 3 rules.

2) STEP 2. FUI'S AUTOMATIC GENERATOR

The second step in the transformation processes takes the CUI model obtained in Section III-C1 and generates, via an M2M transformation, the FUI model. This latter is platform-specific as per MBUID's levels of abstraction. Recall that, herein the interest is only focused on the Android OS.

Algorithm 1 Pseudo-Algorithm for Specifying the Input Abstract Component

```
forall the Input(I) do
    // Checks whether I aims to select an
       item from a set
    if isSelectN(I) then
        // Checks whether I's elements are
           > 4
        if element(I).count() > 4 then
            // Create a ComboBox for I
            CreateComboBox(I)
        else
            forall the E: element(I) do
                // Create a CheckBox for E
                CreateButton( E, "CheckBox")
// rest of code
```

The generation of the FUI model implies that each component from the CUI model is mapped into its correspondent one according to the FUI meta-model terminologies. Figure 6 depicts 2 examples of transformation rules allowing

```
helper context MM!AC def : isMainWindow : Boolean =
    if
        self.refImmediateComposite().refImmediateComposite()= MM!AC then
        false
    else
        true
    endif;
rule MainWindow{
    from
            ac:MM!AC(ac.isMainWindow)
    to
            cc :MM1!Window(                                    Rule 1
            id <- ac.Id,
            Text <- ac.Name,
            components <- ac.containsAC,
            cc <- ac.contains
        )
}
rule CrateContainer{
    from
            ac:MM!AC(ac.NotMainWindow)
    to
            cc :MM1!Panel(                                     Rule 2
            id <- ac.Id,
            Text <- ac.Name,
            cc <- ac.contains,
            components <- ac.aic
        )
}
rule SelectElement{
    from
            aic:MM!Input(aic.isSelectElement)
    to
            cic: MM1!Button (                                  Rule 3
            id <- aic.Id,
            Text <- aic.Name,
            Type <- 'CheckBox' )
}
```

FIGURE 5. Excerpt of ATL code.

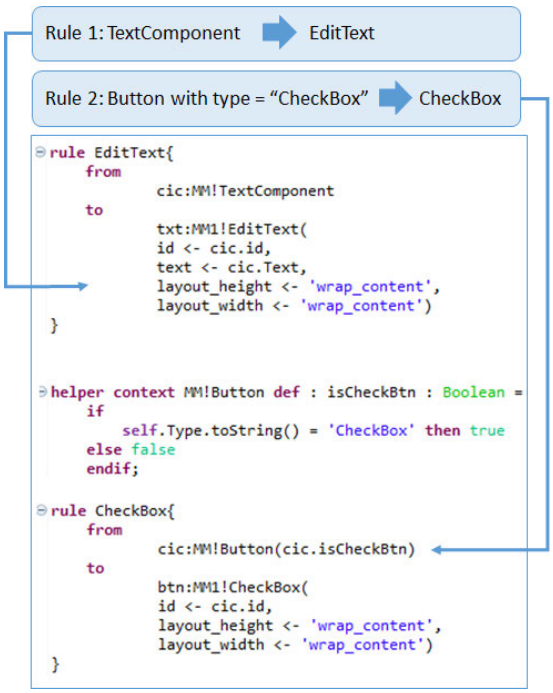

FIGURE 6. Some examples of the rules that allow selection of the concrete component and the ATL code that implements them.

to associate some concrete components with their corresponding ones in the target model. It also shows the ATL code that implements these transformation rules. For instance, each Button having the type CheckBox in the CUI model is transformed to a CheckBox in the FUI model.

3) STEP 3. GUI'S SOURCE CODE AUTOMATIC GENERATION

The last step of our approach is the generation of the mobile UI's source code, in XML language, from the FUI model.

Template.xpt | AUIModel.xmi | CUIModel.xmi | FUIModel.xmi

```
1 «IMPORT FinalUIMetaModel»
2 «DEFINE main FOR FUIModel»
3 «EXPAND form FOREACH activity»
4 «ENDDEFINE»
5 «DEFINE form FOR Activity»
6 «FILE id+".xml"»
7 <?xml version="1.0" encoding="utf-8"?>
8 <androidx.constraintlayout.widget.ConstraintLayout
9 xmlns:android="http://schemas.android.com/apk/res/android"
10     xmlns:app="http://schemas.android.com/apk/res-auto"
11     xmlns:tools="http://schemas.android.com/tools"
12     android:layout_width="match_parent"
13     android:layout_height="match_parent"
14     tools:context=".«this.id»">
15     «EXPAND widget FOREACH widget»
16     «EXPAND layout FOREACH layout»
17 </androidx.constraintlayout.widget.ConstraintLayout>
18 «ENDFILE»
19 «ENDDEFINE»
20 «DEFINE widget FOR Widget»
21 <Widget android:layout_width="«this.layout_width»"
22 android:layout_height="«this.layout_height»"
23 android:text="«this.text»" android:id="«this.id»" />
24 «ENDDEFINE»
25 «DEFINE layout FOR LinearLyout»
26 <LinearLyout android:layout_width="«this.layout_width»"
27 android:layout_height="«this.layout_height»"
28 android:orientation="«this.orientation»" />
29 «ENDDEFINE»
```

FIGURE 7. Excerpt of the Xpand templates.

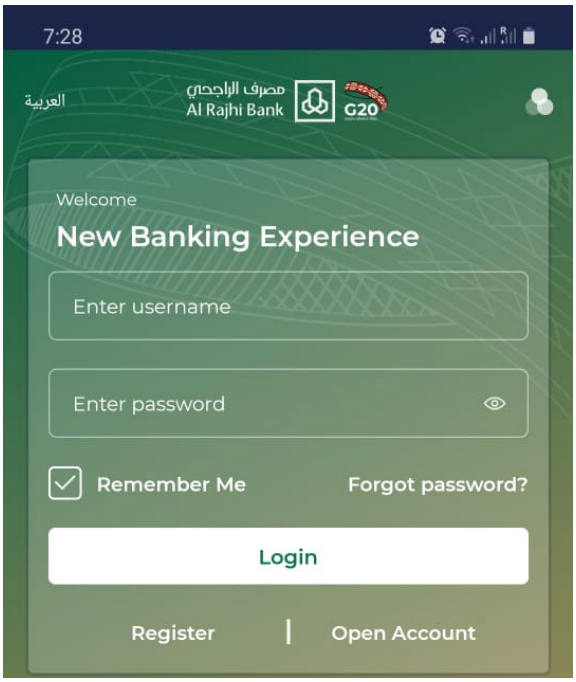

FIGURE 8. Sketch of the graphical user interface of AlMubasher.

The applicability of this step is demonstrated, in this paper, using the Android OS as platform. We argue that a slight modification in the generator code will allow to generate the final code for any others platform such as iOS or BlackBerry. A set of transformation rules and an-house developed code generator constitute the main building blocks of this step. The transformation rules assign the adequate XML tag to each element in the FUI meta-model. As for the code generator, it is developed under Eclipse and using Xpand model-to-text transformation language. Xpand defines

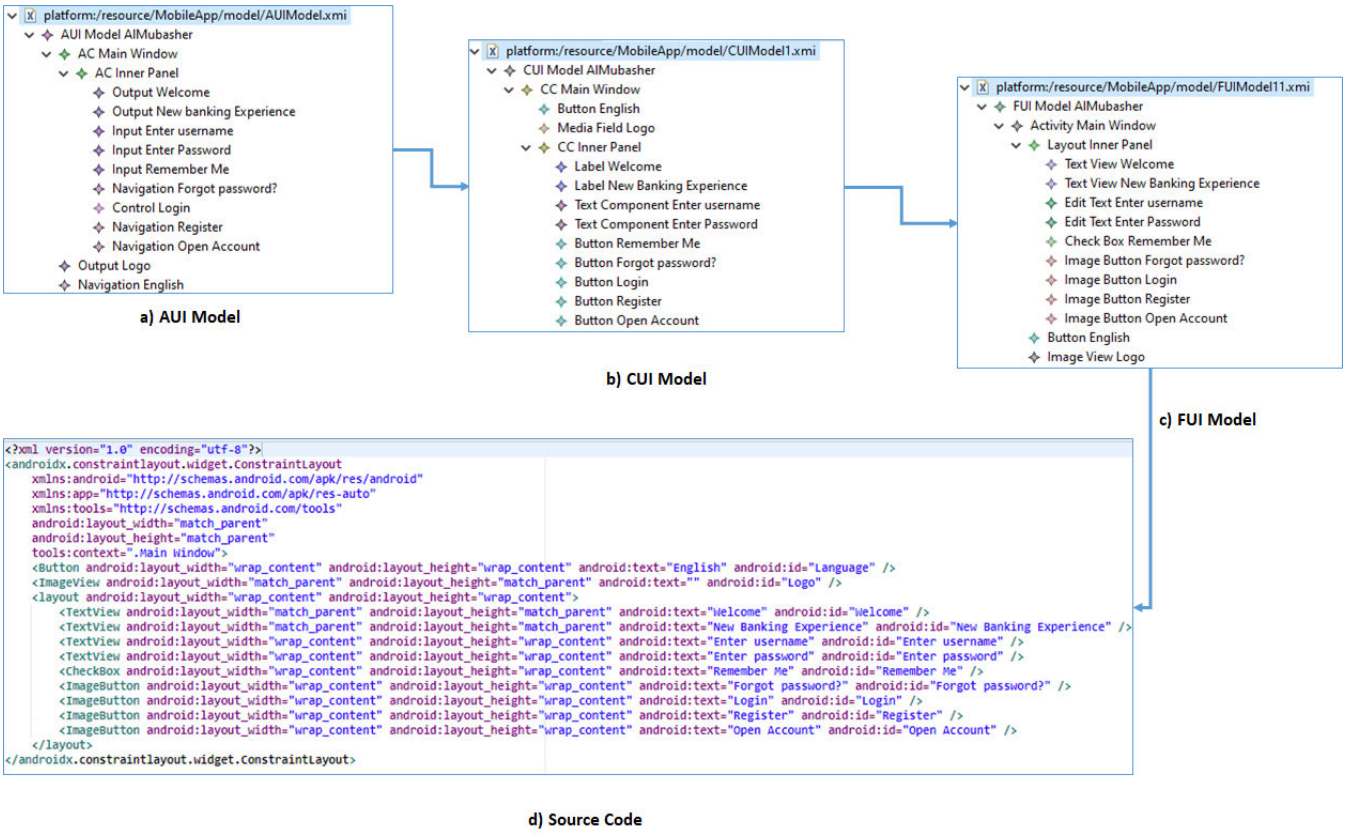

FIGURE 9. Code generation through model transformation.

templates implementing the transformation of each element in the FUI model to the adequate XML tags. The resulted XML file defines the source code of the mobile UI. Figure 7 details how activities, widgets, and layout are transformed to the corresponding XML tags.

IV. CASE STUDY

This section presents a real case study that illustrates the feasibility and usefulness of our proposed approach. The case study is about AlMubasher that is among the most used mobile applications in Saudi Arabia. AlMubasher allows AlRajhi bank's customers to access a variety of banking services such as transfer, payment, accounts management, insurance, etc. Since the application is too large; we only consider the login task to illustrate our approach.

Figure 8 presents the current login page of AlMubasher application. The login page is populated by 2 widgets showing the logo of the bank and available languages as well as a container. This latter regroups 9 widgets: 2 labels showing welcome messages, 2 text fields allowing the user to input his/her name and pass word, and 5 buttons allowing the user to login to the system, ask the system to remember him/her, ask for help, register, and open an account.

The main objective of the case study is to apply our approach in order to create a mobile user interface that is similar to the user interface presented in Figure 8. Thus, the latter is considered as a basis during the specification of our AUI model. As depicted in Figure 9-a, our AUI model includes 3 input, 1 control, 4 navigation and 3 output. These elements meet the requirement of the considered mobile user interface in terms of widgets and containers.

Once the AUI is modeled, it will be injected as input for the first M2M transformation allowing to obtain the CUI model. Figure 9-b also shows the generated CUI model. First of all, a concrete container from the class Window is associated with the root node from the AUI model. Secondly, an other container from the class Panel is associated with the intermediate node. Finally, a set of components are created and associated with their correspondent in the input model.

The obtained CUI model is also subject of a M2M transformation allowing the generation of the FUI model. As per the previous M2M transformation, each component/container in the CUI model is transformed into the corresponding component/container according to the platform considered during the definition of the FUI. In our case, we considered Android iOS, thus, the component included in the FUI model are compliant with the Android specification. For instance, each TextComponent is transformed into an Edit Text component. Figure 9-c illustrates the obtained FUI model.

Last but not least, the XML code of the mobile user interface is automatically generated from our FUI model. During this transformation, each component of the FUI model is

transformed into an XML tag according to the syntax of Android's XML vocabulary. [6] Figure 9-d depicts the obtained XML code of the mobile user interface.

V. CONCLUSION AND FUTURE WORKS

The development of mobile applications has gained a specific interest in recent year. This is due to the worldwide massive use of various types of mobile devices. However, the diversity of mobile devices and their operating systems come-up with a new challenge that is the development of cross-platform mobile applications. Indeed, the success of a mobile application is closely related to its availability for various platforms. Hence, mobile application developers need to develop several versions of the same application that meet the requirements of the different platforms.

To deal with the aforementioned challenges, the paper proposes an MBUID approach for developing graphical user interfaces of mobile applications. The approach is based on 3 meta-models that fit the different MBUID' levels of abstraction. As per the MBUID paradigm, the development process starts by an abstract specification of the user interface (AUI model) that is transformed into a concrete specification (CUI model). The later is also transformed into an operational user interface (FUI model) in a technological space (e.g., computing platform, programming or markup language). These 2 transformations are granted thanks to a set of M2M transformation rules that are implemented using ATL language and under eclipse platform. Last but not least, the FUI is transformed to the source code of a mobile application. This transformation is ensured thanks to a M2T transformation rules implemented using Xpand language and under eclipse platform too. To assess the feasibility and usefulness of the proposed approach, the paper introduces a real case study about AlMubasher mobile application.

In terms of future works, we plan to develop a graphical editor that supports users during the definition of the abstract model that constitutes the principal input of the approach. In addition, we will consider other mobile platforms like iOS through the definition of adequate FUI meta-models and transformation rules.

REFERENCES

[1] M. Brambilla, J. Cabot, and M. Wimmer, "Model-driven software engineering in practice: Second edition," *Synth. Lectures Softw. Eng.*, vol. 3, no. 1, pp. 1–207, Mar. 2017.

[2] E. Umuhoza and M. Brambilla, "Model driven development approaches for mobile applications: A survey," in *Mobile Web and Intelligent Information Systems*. Springer, 2016.

[3] G. Meixner and G. Calvary. (Jan. 2014). *Introduction to Model-Based User Interfaces*. [Online]. Available: https://www.w3.org/TR/2014/NOTE-mbui-intro-20140107/

[4] J. Vanderdonckt, "A MDA-compliant environment for developing user interfaces of information systems," in *Advanced Information Systems Engineering*, O. Pastor and J. F. E. Cunha, Eds. Berlin, Germany: Springer, 2005, pp. 16–31.

[5] G. Meixner, F. Paternò, and J. Vanderdonckt, "Past, present, and future of model-based user interface development," *I-Com*, vol. 10, no. 3, pp. 2–11, Nov. 2011.

[6] L. Zouhaier, Y. B. Hlaoui, and L. J. B. Ayed, "Generating accessible multimodal user interfaces using MDA-based adaptation approach," in *Proc. IEEE 38th Annu. Comput. Softw. Appl. Conf.*, Jul. 2014, pp. 535–540.

[7] F. Bacha, K. Oliveira, and M. Abed, "A model driven architecture approach for user interface generation focused on content personalization," in *Proc. 5th Int. Conf. Res. Challenges Inf. Sci.*, May 2011, pp. 1–6.

[8] P. Pinheiro, "User interface declarative models and development environments: A survey," in *Proc. 7th Int. Conf. Design, Specification, Verification Interact. Syst.*, 2001, pp. 207–226.

[9] H. Tufail, F. Azam, M. W. Anwar, and I. Qasim, "Model-driven development of mobile applications: A systematic literature review," in *Proc. IEEE 9th Annu. Inf. Technol., Electron. Mobile Commun. Conf. (IEMCON)*, Nov. 2018, pp. 1165–1171.

[10] I. Qasim, F. Azam, M. W. Anwar, H. Tufail, and T. Qasim, "Mobile user interface development techniques: A systematic literature review," in *Proc. IEEE 9th Annu. Inf. Technol., Electron. Mobile Commun. Conf. (IEMCON)*, Nov. 2018, pp. 1029–1034.

[11] H. Heitkötter, S. Hanschke, and T. A. Majchrzak, "Comparing cross-platform development approaches for mobile applications," in *Proc. WEBIST*, 2012, pp. 1–13.

[12] M. Nuñez, D. Bonhaure, M. González, and L. Cernuzzi, "A model-driven approach for the development of native mobile applications focusing on the data layer," *J. Syst. Softw.*, vol. 161, Mar. 2020, Art. no. 110489.

[13] M. Usman, M. Z. Iqbal, and M. U. Khan, "A model-driven approach to generate mobile applications for multiple platforms," in *Proc. 21st Asia–Pacific Softw. Eng. Conf.*, vol. 1, Dec. 2014, pp. 111–118.

[14] X. Jia and C. Jones, "Axiom: A model-driven approach to cross-platform application development," in *Proc. 7th Int. Conf. Softw. Paradigm Trends*, vol. 1, 2012, pp. 24–33.

[15] F. Freitas and P. H. M. Maia, "JustModeling: An MDE approach to develop Android business applications," in *Proc. 6th Brazilian Symp. Comput. Syst. Eng. (SBESC)*, Nov. 2016, pp. 48–55.

[16] L. B. Ammar, "Towards a uniform model transformation process for abstract user interfaces generation," in *Proc. 14th Int. Conf. Eval. Novel Approaches Softw. Eng.*, 2019, pp. 533–538.

[17] *Mobile Operating System Market Share Worldwide*, Statcounter, Dublin, Ireland, 2021.

[6] https://developer.android.com/guide/topics/ui/declaring-layout

LASSAAD BEN AMMAR received the Ph.D. degree in computer science from the University of Sfax, Tunisia, in 2015. He is currently an Assistant Professor with Prince Sattam bin Abdulaziz University, Saudi Arabia. He is the author or the coauthor of several papers in international conferences and journals. His current research interests include model driven engineering for mobile applications and usability engineering.