Computer Systems Science & Engineering
DOI: 10.32604/csse.2023.030358
Article

Usability-Driven Mobile Application Development

Fadwa Yahya[1,2,*], Lassaad Ben Ammar[1,2] and Gasmi Karim[3]

[1]Prince Sattam Bin Abdulaziz University, Alkharj, Saudi Arabia
[2]University of Sfax, Sfax, Tunisia
[3]Department of Computer Sciences, Faculty of Sciences and Arts in TABARJAL, Jouf University, Kingdom of Saudi Arabia
*Corresponding Author: Fadwa Yahya. Email: f.yahya@psau.edu.sa
Received: 24 March 2022; Accepted: 26 April 2022

Abstract: Recently, a specific interest is being taken in the development of mobile application (app) via Model-Based User Interface Development (MBUID) approach. MBUID allows the generation of mobile apps in the target platform(s) from conceptual models. As such it simplified the development process of mobile app. However, the interest is only focused on the functional aspects of the mobile app while neglecting the non-functional aspects, such as usability. The latter is largely considered as the main factor leading to the success or failure of any software system. This paper aims at addressing non-functional aspects of mobile apps generated using MBUID approach. As such, we propose a usability-driven approach for the development of mobile apps. The main stages of the proposed approach are defined in a generic way so that they can be integrated with any MBUID method. A case study is presented, in the paper, with the aim of illustrating the feasibility of this approach.

Keywords: Mobile application; user interface; model-based user interface development; mobile usability

1 Introduction

In the last decade, mobile apps development has become one of the most concerned and rapidly developing areas [1]. In this context, the rapid increase in number and diversity of mobile platform constitutes a challenging issue for application developers since they need to develop the same applications across each target platform separately [2]. A promising solution that is widely accepted is Model-Based User Interface Development (MBUID) [3,4]. Using an MBUID approach, an application is usually developed through the definition of high-level models and their transformation into a less abstract level to the code in the target platform. MBUID has shown to be successful in simplifying the app development process, reduce complexity, increase abstraction level and maximize cost-effectiveness and productivity [2].

The aforementioned potentialities of the MBUID approach have led to an increasing number of research initiatives adopting MBUID-compliant method for the development of mobile apps. In 2016, the survey presented in [3] discusses 17 research works. In 2018, the number reached 30 in a Systematic Literature

Review (SLR) conducted in [5]. The most recent SLR conducted in this context in 2020 [4] identify 55 MBUID based development process for mobile apps development. The ultimate objective of the majority of these research works is to investigate the adoption of MBUID paradigm for the development of mobile apps. Existing research initiatives in this field are very interesting in demonstrating the applicability and effectiveness of MBUID in the mobile development context. However, these initiatives still suffer from several limitations that could undermine their adoption. For instance, existing approaches focus only on the functional aspects of the mobile apps while neglecting the non-functional ones, such as usability. The latter is largely considered the main factor that could lead to the success or failure of any software system.

Starting from this report, the present paper aims at investigating how to ensure the development of a usable mobile app, in particular user interfaces, using an MBUID-compliant process. The main idea behind our proposal is the following: how to exploit the traceability mechanism established between conceptual models and the final user interface to improve the usability of the obtained user interface. To this end, the proposed approach relies on 3 stages: the first stage abstractly represent the usability requirements/properties in the intermediate artifacts of the development process. The second stage resorts to the parameterized transformation [6] to blend the usability properties with the conceptual primitives in the development process. Finally, the third stage executes the model transformation process to generate a user interface that meets the required usability properties. Furthermore, we demonstrated the integration of our approach with an existing MBUID method for the development of mobile apps to improve the quality of the generated apps. It is worth mentioning that our proposal is defined in a generic way allowing its integration with any MBUID method. The feasibility of the proposed approach is illustrated through a case study.

The rest of the paper is structured as follows. Section 2 presents an overview of the Model-based approach for the development of Graphical User Interface (GUI) and discusses some related works. Section 3 introduces the fundamentals of our usability-driven mobile apps development approach. Section 4 shows how this approach is integrated with an MBUID method. Section 5 suggests a case study illustrating the feasibility and effectiveness of our proposal. Finally, concluding remarks and some directions for future works are drawn in Section 6.

2 Background

This section presents the fundamental concepts of model-based approach for the development of user interfaces. Then, it introduces the notion of parameterized transformation that is widely accepted as promising technique to combine functional and non-functional requirements. These two concepts constitute the building blocks of our proposal to incorporate usability requirements during the development of mobile applications. The last part of this section discusses related works in the field of model-based UI development and usability evaluation.

2.1 Model-Based User Interface Development

The development of today's user interfaces is challenging due to the heterogeneity of end-users, computing platforms and programming languages [7]. Model-Based User Interface Development (MBUID) is an approach for dealing with the aforementioned issues and reducing the required effort to develop UIs while ensuring their quality [8]. The purpose of MBUID is to build and conceptual models that describe the UI at different level of abstraction, and the design proceeds in a structured manner from abstract models to more concrete ones.

Different frameworks and guidelines were developed over the last years to best capture the important parts of a MBUID process. Consequently, a clear understanding of the abstraction layers and type of

models to be considered within a MBUID process is established. In 2003, the EU-funded CAMELEON project introduced a framework that serves as a reference for the development of multiple targets or multiple context of use UIs on the basis of the model-based approach [9]. The Cameleon Reference Framework (CRF) covers both the design-time and run-time phases. It describes different abstraction layers and the relationships among them.

Fig. 1 depicts the different abstraction layers and their relationships as introduced in the CRF. The first abstraction layer of the CRF describes the hierarchies of tasks to be carried out in a specific temporal order to achieve the users' goals. The second layer expresses the UI in terms of abstract interaction objects that are independent of any platform or modality (graphical, vocal, and haptic). The third layer expresses the UI in terms concrete interaction objects that are modality-dependent but implementation technology independent, thus platform specific. The last layer expresses the UI in terms of technology language dependent source code. It can be represented in any UI programming language (e.g., Java) or mark-up language (e.g., HTML).

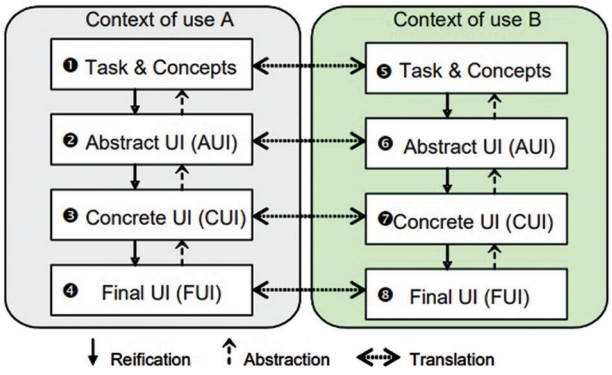

Figure 1: The Cameleon Reference Framework [9]

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

2.2 Parameterized Transformation

In model driven development, model transformation is a crucial concept that provides a mechanism for automating the manipulation of models. The aim of a model transformation is usually to (automatically) generate a model (target) from another (source), according to a *transformation definition* [10]. The latter is defined as a set of *transformation rules* that together describe how a model can be transformed into another [11]. Each transformation rule describes how one or more constructs in the source model is transformed into one or more constructs in the target model.

An interesting variant of the ordinary model transformation called *parameterized transformation* was initiated in [6] to adapt UIs to their context of use. In such a transformation, a third model is required to play the role of a parameter that could be used to improve new functionalities (values, properties, operations) or to change the application behavior (activities). The designer must specify the parameters to be inserted at the transformation as well as the elements from the application model that will receive contextual details. In [6], the *parameterized transformation* was adopted to refine PIM models by adding, deleting or updating contextual details and hence generating a Contextual PIM (CPIM). After that, a

traditional transformation is susceptible to generate a Contextual PSM (CPSM) that inherits business requirements and context from the CPIM (Fig. 2).

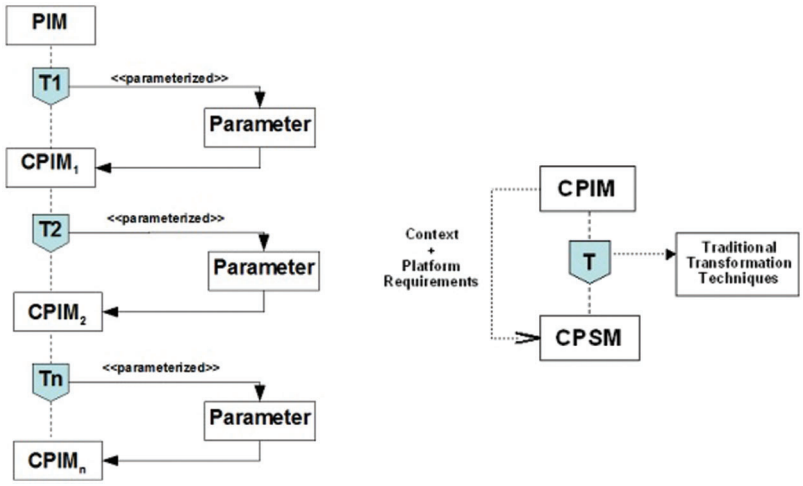

Figure 2: The parameterized transformation concepts [6]

In this paper, we opted for the parameterized transformation to blend the usability issues with the conceptual models as per Section 3.2.

2.3 Related Works

This section briefly describes relevant model-based approaches for the development of mobile apps with focus on UI development. Then, it overviews dedicated approaches for the evaluation of mobile apps' usability. Finally, it outlines the challenges that need to be addressed to cope with the gaps between both research areas (i.e., model-based development and usability evaluation).

2.3.1 Model-Based Approaches for Mobile Applications

Our literature review reveals that Model-Based User Interface Development (MBUID) has gained a special attention by the SE community. In recent years, research initiatives that adopt MBUID for the development of mobile apps have increased considerably. These initiatives have provided promising results in dealing with common challenges in the development of mobile apps. However, our literature review reveals that more efforts are still required to take the strengths of MBUID in the context of mobile UI development.

In 2016, a survey of model-based approaches for the development of mobile applications discussed only 13 research papers and 4 commercial solutions [3]. Furthermore, only about 30 research works in the field were identified in two Systematic Literature Reviews (SLR) presented in [12] and [5]. This proves the shortage of research work in the field of mobile UI development via model-based techniques.

In the most recent SLR presented in [4], the authors identified only 55 research works that focus on the adoption of model-based techniques for the development of mobile apps. This study argue that the adoption of these techniques impacts both the development lifecycle and the obtained app. On the one hand, adopting model-based techniques allows to improve the development lifecycle by guaranteeing a certain level of flexibility, efficiency, reliability, and reuse [4]. On the other hand, the adoption of these techniques could help improving the quality of the developed mobile apps [4].

With regard to the modeling techniques, our literature review reveals that existing model-based approaches commonly use UML and DSL to design models of various level of abstraction (e.g., [13–15]). Then a set of model-to-model and model-to-text transformations are defined to generate the application code from the high level model.

In terms of achieved results, existing studies highlight the potentialities of this development paradigm in increasing abstraction, productivity, flexibility, efficiency, and automation for the app development as well as supporting cross-platform, multi-platform or multi-version app development [4].

Based on the analysis of the state of the art, we can conclude that the development of mobile apps, in particular UIs, using model-based techniques still an immature research area and many more efforts are required. There is a lack of an agreed-upon language for specifying the app's functional and non-functional requirements. Thus, future works must focus on the unification of the concepts, terminologies and tools to be used by researchers. Furthermore, the usability of the obtained mobile app is usually evaluated once the app is developed. Hence, a lot of re-engineering efforts are required to improve its quality.

2.3.2 Evaluation of Mobile UIs

Evaluating and improving the usability of software systems is an important factor to ensure their success and acceptance by end users [16]. Regarding its importance, a wide range of research works have investigated the evaluation and improvement of the usability for various types of software systems. Their main objective is to find potential problems that could lead to the failure of the developed systems. They may focus also on the user-satisfaction level against the overall system by considering both objective and subjective measures.

In the context of mobile development, usability evaluation still focuses on the same objectives as for other kind of software systems. Thus, classic usability evaluation techniques are adopted with a slight tolerance to overcome mobile device limitations such as small screen size, limited connectivity, high power consumption rates, and limited input modalities. Usability evaluation techniques for mobile apps can be mainly classified into 2 main categories: *laboratory experiments* and *field studies*.

Laboratory experiment is the most adopted technique for evaluating the usability of software systems, in particular mobile apps. This technique involves potential users of the application that are asked to accomplish a set of predefined tasks in a very specific and controlled environment [17,18].

The behavior of the users, while interacting with the application, is recorded and then analyzed to identify potential usability problems. Thanks to the controlled environment and predefined tasks, this kind of technique may ensure an evaluation of overall usability aspects. However, isolating users from the environmental factors prevalent in the real world may cause differences in user experience [19]. Furthermore, carrying out experiments in a controlled environment require a large amount of resources increasing considerably their cost [20].

As for the field study, data about users' needs and product requirements are collected using observations and interviews [19]. The technique requires a trial use of the application by real users and collecting data about their opinions regarding the application. Data collection can be made either by taking notes when users are involved in some activities or asking them to fulfill a questionnaire reflecting their experience with the application. The quality of the questionnaire is considered as the main drawback of the field studies techniques [21].

In summary of our literature review for mobile usability evaluation, we should underline that the usability evaluation for mobile apps still requires a lot of efforts to overcome the following shortcomings:

Usability evaluation was typically conducted as a post-implementation step in the development process leading to a lot of re-engineering work to overcome the identified problems.

The usability measures used are independent of the development process without any way to handle them throughout this process. Consequently, there is no way for designers and developers to identify the required changes which are susceptible to improve these measures.

2.3.3 Discussion of the State of the Art

Our literature review reveals that the adoption of MBUID to support the development of cross platform mobile applications has increased over the last few years. This is due to its ability to generate various versions of the same application from a platform independent conceptual model. However, this research area still needs more effort to address current challenges. Among these challenges, we cite essentially the usability of the generated user interfaces that is of great importance to the acceptance of the developed application. In the state of the art, the usability is usually evaluated once the application is developed. Hence, the latter is improved thanks to re-engineering approaches that usually engender additional costs in terms of time and resources.

A promising solution that was adopted in the context of interactive systems consists of dealing with non-functional requirements throughout the MBUID development process. In this context, the generation of the application from the conceptual model is ensured thanks to a set of model transformations that deal with usability requirements [22]. Usability requirements are injected to the development process via the technique of parameterized transformation. Although this technique was proved appealing for the development of usable UIs of interactive systems, there are no proposals, to the best of our knowledge, for adopting it in the context of mobile apps development.

In conclusion, we highlight that the adoption of MBUID for mobile apps development is at an early stage and there is acute lack for an effective solution. Effective solution should provide for:

- The unification of the concepts, terminologies and tools to be used by researchers.
- The adoption of an MBUID approach that considers usability throughout the development process of mobile apps.
- The adoption of a generic approach that can be applied to any MBUID method.

This research work is a first step in this direction and aims to propose an approach that combines functional and non-functional requirements during the development of mobile UI.

3 Usability-Driven Mobile App Development (UMAD) Approach

This section details our approach for blending conceptual primitives with usability properties (Fig. 3).

The approach goes through 3 stages: Conceptual Primitives (CP) Identification, Usability Injection, and Execution. The CP identification stage identifies the most suitable conceptual primitive[1] for supporting each usability properties presented in the agreed-upon usability model as discussed in Section 3.1. The usability injection stage blends the conceptual primitives with corresponding usability properties thanks to the parameterized transformation technique. This stage is detailed in Section 3.2 Finally, the execution stage applies the transformations defined in the injection stage on the conceptual models to generate the final UI as detailed in Section 3.3.

3.1 CP Identification

The aim of this stage is to identify the most suitable conceptual primitive(s) that may abstractly represent a usability property. To this end, 2 identification processes are carried out. The former aims to create a repository of usability properties. While the latter focuses on the identification of the conceptual primitives that could support each usability property.

[1] A conceptual primitive is an element of the modeling language that abstractly represents a feature of the system. Classes, attributes and services are examples of conceptual primitives in a class diagram.

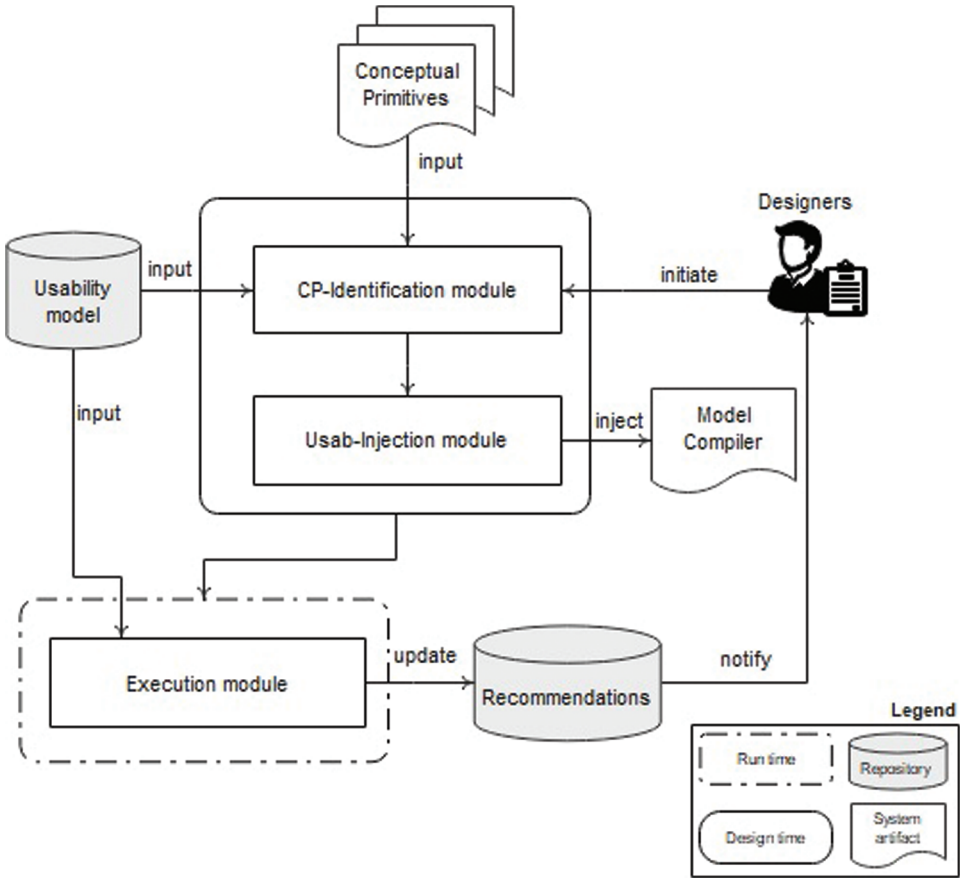

Figure 3: Approach for blending conceptual primitives with usability properties

To create a repository of usability properties, we established a depth analysis of the most known usability guidelines and surveys presented in the usability literature, in particular mobile usability. Note that, since these usability properties are intended to be tackled in the intermediate artifacts, we only consider those that can be abstractly represented in a precise notation by means of conceptual primitives. For example, effectiveness and efficiency can't be quantified until the system is implemented. Thus, they are discarded from our analysis.

Moreover, since the identification of the conceptual primitives depends on the considered MBUID-method, the first step is to select the desired method. In this paper, we opted for the one initiated in [23]. After that, we identified the conceptual primitives that can abstractly represent each usability property, from the repository. For instance, most guidelines for mobile user interfaces recommend a well-defined size for the tappable area of touch target widget (44 × 44 dp for iOS system). A possible conceptual primitive that may support this recommendation can be any attribute that holds the size of a widget. Should this primitive be absent in the MBUID method, UMAD recommends to update/enrich the conceptual models of the method by the missing primitive(s). Those UMAD's recommendations highlight one of the potentialities of our approach: discovering the weaknesses and limitations of an MBUID method with regard to its conceptual primitive's expressiveness.

3.2 Usability Injection

This stage aims to blend the conceptual primitives with the usability properties identified in the previous stage. Such a blend takes place in the model compiler, which include the transformation rules responsible for generating the UI of the mobile app from the conceptual model. These transformation rules are subject to 2 possible configurations:

- Static Configuration: This configuration is applied when there is only one alternative to blend the usability property and the conceptual primitive. For instance, most mobile usability guidelines agree that the use of an appropriate text body font size will increase the legibility of the mobile app. Thus, each conceptual primitive representing this property need to be configured with the recommended value according to the usability guideline. It is important to note that the recommended value may differ according to the platform (iOS, Android).
- Dynamic Configuration: This configuration is applied when a usability property may be fulfilled with more than one alternative. For example, if end-users are asked to put the data within an input widget, the following alternatives can be considered:
  - Use a text field with an associated label displaying supplementary information about the required data/format.
  - Use a drop-down list containing all accepted values.
  - Use a list of radio button where each item represents a value/range of values.

Even if the aforementioned alternatives achieve the same functional requirement, it is clear that each one of them favors at least one usability property (e.g., prompting for the former, error prevention for the two latter). Hence, the selection of the appropriate alternative may depend on the usability property to be fulfilled.

To perform both configurations, we opted for the following strategies: (1)

- Static Configuration, the transformation rules associated with a conceptual primitive, who is subject of a blending operation to support a usability attribute, is slightly modified by putting the recommended value. For example, the transformation rule that create a *Text Field* will be adapted to set the appropriate text body size.
- Dynamic Configuration, we resort to the parameterized transformation to inject the usability properties requiring such configuration. As such, the model transformation engine will select the appropriate alternative that meet the desired usability attribute.

As per Section 2.2, a parameterized transformation requires a third model that plays the role of a parameter. In our case this new model includes the desired usability properties that the mobile UI is expected to fulfill.

3.3 Execution

The last stage in our approach is about the generation of the mobile UI from the conceptual model with respect to the desired usability properties. To this end, the conceptual model of the mobile app must be designed according to the selected MBUID method. Furthermore, our approach implies the definition of a usability model that specifies the usability properties that the mobile app is expected to fulfill. As any transformation process implies that each model is defined with respect to a high level meta-model, UMAD proposes the meta-model depicted in Fig. 4 to define the usability model. This latter plays the role of a parameter to the model compiler. Based on the specified properties, the model compiler selects the adequate configuration of the transformation rules.

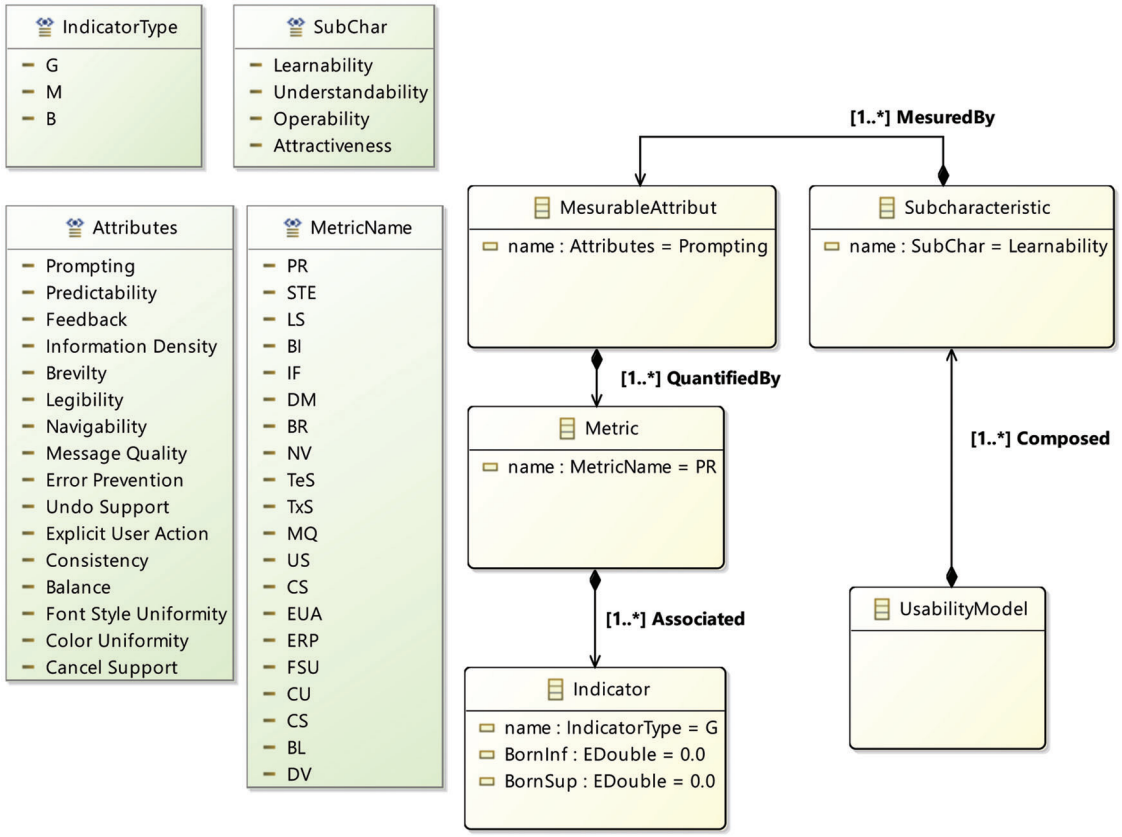

Figure 4: Usability meta-model

4 Proof of Concept

This section demonstrates the feasibility of our approach through its application to an MBUID method. To this end, it introduces the adopted method then it details the application of UMAD to generate a usable mobile UI.

4.1 MBUID Method for Mobile UI Development: Overview

The MBUID method proposed in [23] automatically generates the source code of a mobile application's UIs thanks to a model transformation process. The transformation process takes as input the conceptual models of the application that represent the application thanks to 3 abstraction levels. The first two levels are independent from the platform while the last is platform-dependent.

- Abstract User Interface (AUI) Model: specify the user interface in terms of *Abstract Interaction Object* (AIO) and *Abstract Relationship*. AIO represents an abstraction of widgets that can be found in popular GUI toolkit such as buttons and panels. They have been classified into two main categories: *Abstract containers* (AC) and *Abstract Individual Components* (AI). AC represents an entity that may gather other abstract containers or components. AIC represents an abstraction of an interaction object such as text field, button, drop down menu, etc. Abstract relationships specify relationships that can be established between abstract interaction objects of all kinds.
- Concrete User Interface (CUI) Model: specify the user interface in terms of *Concrete Interaction Object* (CIO) known as widgets. CIO can be either *Concrete Containers* (CC) such as window and panel, or *Concrete Interaction Components* (CIC) like text field, button and drop down list.

- Final User Interface (FUI) Model: specify the terminologies to render the CUI model in a specific technological platform such as Android or iOS. As the Android OS takes over the majority of the worldwide market, the interest in [23] is focused on that OS.

On top of these abstraction levels, the method proposes a transformation process that entails 2 kinds of transformation. The former automatically transforms an AUI model to a CUI model that is in turn transformed to a FUI model thanks to a model-to-model transformation. The latter performs a model-to-text transformation that generates the source code of the user interface from the obtained FUI model. In this paper, we consider this method to prove the feasibility of our UMAD approach as per the following Section.

4.2 UMAD's Application

In this section, we explain how UMAD upgrades the adopted MBUID method to support some usability properties. As an example of usability properties, we selected Tapped Element Size, Error Prevention, Text Size, Prompting and Cancel Support from the usability model proposed in [24]. We have selected these usability properties because of their widespread recommendation in most usability guidelines and surveys for mobile apps.

4.2.1 Tapped Element Size

The Material Design Guidelines for the Android system[2] recommends maintaining a minimum tappable area of 48 × 48 dp for all controls.

To allow the MBUID method presented in [23] to support such a usability property, UMAD proceeds as follow:

- CP Identification: two attributes (*layout-height* and *layout-width*) located in the Widget meta-class of the Final UI meta-model can be configured to support the usability recommendation with regard to tapped element size. These attributes are currently initialized in [23] with a default value "*wrap-content*" in any transformation rule allowing to create an object from the meta-class Widget (Fig. 5a). Such a value makes the widget size auto-adjustable to its content.
- Usability Injection: to be in compliance with the usability guidelines for the Android widgets, UMAD proposes to set the default value of *layout-height* and *layout-width* to 48 dp for any touch target widget (Fig. 5b). As such, we illustrate the Static Configuration to blend Conceptual Primitives (CP) with Usability Properties (UP). It is important to note that this configuration is only applicable for touch target widgets which are concerned by the recommendation.

4.2.2 Error Prevention

A common way that is usually recommended to improve the error prevention is to allow the user to select data from potential enumerated values instead of typing them. In what follow, we demonstrate how UMAD proceed to fulfill such recommendation.

- CP Identification: in [23], a class called Input is used in the Abstract UI meta-model to abstractly represent an input element. An object from this class inherits from its super-class called Abstract Interaction Object (AIC) an attribute that represents the action required for performing the task: UserAction. To indicate that the required action for an input element is to type/enter data by the user, the UserAction must be initialized with the "*Create*" value.
- Usability Injection: should the Error Prevention need to be favored/fulfilled, UMAD proposes that the model compiler generate an input element from the list class instead of a text field. Note that this

[2] https://material.io/design/usability/accessibility.html\#layout-and-typography

action requires that the system analyst puts, when it is possible, the list of all accepted values for an input element. To consider the Error Prevention, UMAD suggest some modification in both AUI meta-model and the model compiler presented in [23]. With regard to the former, an attribute called "*Value*" was added to the AIC super-class to keep the list of possible values. As for the model compiler, changes are made to allow a Dynamic Configuration of the transformation rules associated with an input element having the "*Create*" as a required action. Such configuration strongly depends on the existence of the Error Prevention in the usability model. Should this property exist (need to be favored), the model compiler will select the appropriate transformation rule allowing to generate a list-type widget in the CUI model for each input element from the AUI model. Otherwise, a text field will be generated. Recall that, we resort to the parameterized transformation to inject the usability properties to be fulfilled/favored in the model compiler.

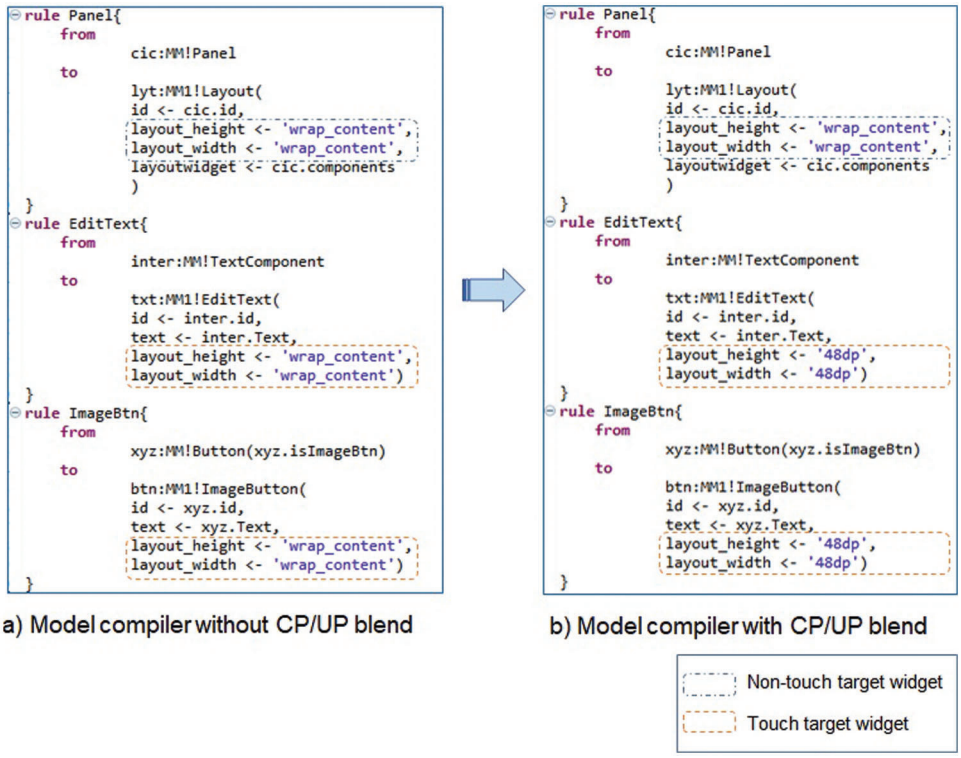

Figure 5: Example of *static configuration* for blending conceptual primitives with usability properties

4.2.3 Text Size

Apple in its Human Interface Guidelines recommends setting the minimum size for Body text to be 17 pt.

Google in Material Design guidelines recommends setting the minimum size for Body text to be 16 sp (equal to 16 pt in iOs). Such recommendations are intended to improve the readability in a mobile user interface and consequently enhance its usability. Similarly to the size of tappable area, we propose to fix the text size in various types of widgets of the FUI model to be conform to the recommendations.

- CP Identification: as the current version of the FUI's meta-model in [23] does not contain any attribute to fix the text size, UMAD proposes to enrich such meta-model with a new attribute called "*Size*" in the Widget class.

- Usability Injection: we modified the model compiler to support the new Static Configuration of the widget size (Fig. 6).

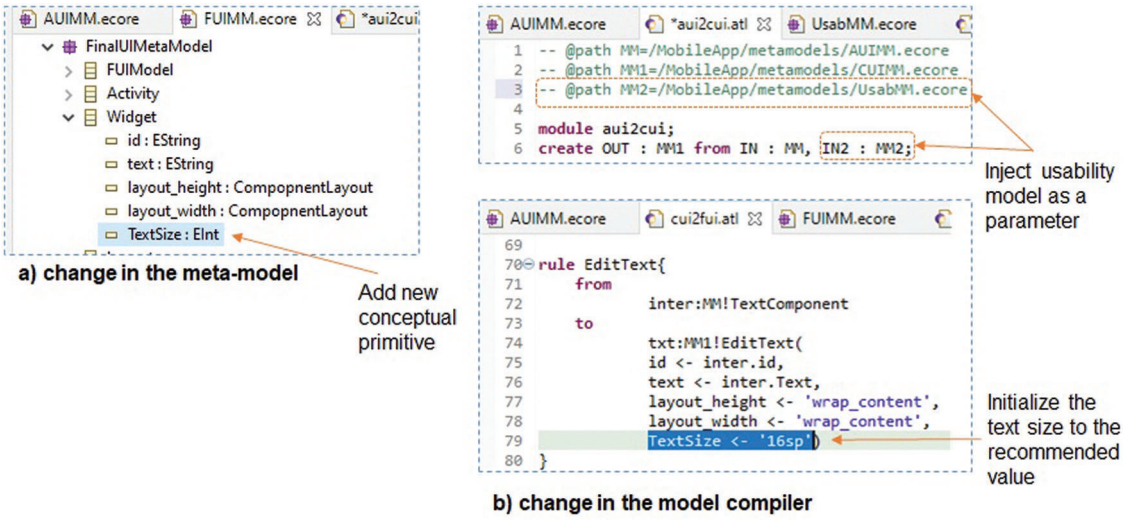

Figure 6: Implementation of the *Text Size* property

It is worth mentioning that adding the attribute Text Size to the Widget class highlights the ability of our approach to discover any lack of expressiveness in the used meta-models.

4.2.4 Prompting

A successful interaction design must provide means to advise, orient, inform and guide the users throughout their interaction with the system. Prompting is one of the most commonly recommended criteria that a software system needs to support to improve its usability. It refers to the means available to guide users through certain actions whether it is data entry or other tasks. It may be implemented via different forms starting from including additional information about the required data until the provision of online help. In this paper, UMAD opted for the implementation of prompting by providing cues on the data format, measurement unit and acceptable length of entries. Should any data entry require additional information about the accepted values or data format, UMAD proposes including such information in the field label.

- CP Identification: similarly to the *Text Size*, UMAD suggests to enrich the current version of the AUI's meta-model by adding a new attribute called "*Info*" in the Input class. Such attribute will hold the supplementary information that is likely to better guide users to enter the correct data.
- Usability Injection: with regard to the model compiler that derives the CUI model from the AUI's one, the transformation rule that deal with the Input class undergoes the required change to include in a field label additional cueing of data format or accepted values.

4.2.5 Undo/Cancel Support

Supporting user control and freedom is one of the most basic characteristics of successful interface design. Users frequently do activities by accident/mistakes, necessitating the presence of a clearly marked "emergency exit" that allows them to quit the unwanted activity without having to go through a lengthy process. To fulfill such control, UMAD recommends to automatically add a *Cancel* button that closes any window and discards any changes the user may have made within that window.

- CP Identification: the conceptual primitive that is concerned with this recommendation is the Window class in the CUI meta-model.
- Usability Injection: to implement such recommendation, a Static Configuration of the model compiler that derives the CUI model from the AUI is made. The transformation rule allowing to create a window in the CUI model is enriched to create a button called *Cancel* in the generated window.

5 Case Study

To illustrate the feasibility of our UMAD approach, this section presents a case study. The main objective here is to prove the value added of UMAD in generating mobile UIs that fulfill usability requirements, at least to some extent. In addition, this section will allow us to better identify the potentialities and the limitations of our approach.

5.1 Subject

We will consider, in this section, the same case study presented in [23] in order to illustrate the impact of UMAD with regard to the usability of the generated UI. The case study is about AlMubasher mobile app that is among the most used mobile applications in Saudi Arabia (Fig. 7).

AlMubasher facilitate customers avail various banking services (transfer, payment, accounts management, insurance, etc.) without visiting branch. Since the application is too large; we restrict our focus on the login task to illustrate our approach.

The current login screen of the app 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 current login screen is kept in mind while defining the Abstract UI model.

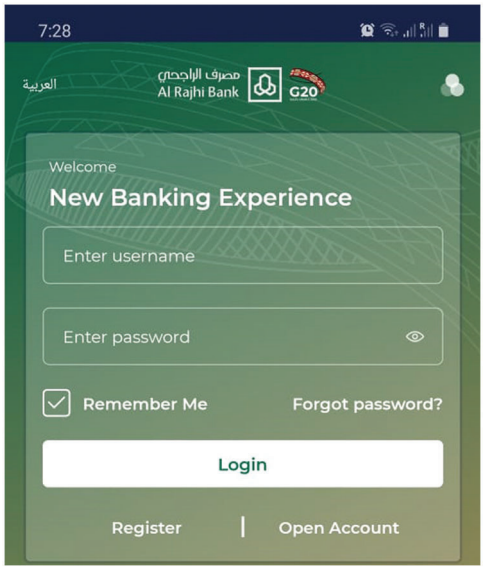

Figure 7: Sketch of the graphical UI AlMubasher

Figure 8: Examples of usability properties and their impact on the generated models

5.2 Blending Conceptual Primitives with Usability Properties

The upper part of Fig. 8 shows the resulting xmi file of the AUI model. This latter undergo a first transformation called aui2cui to generate the CUI model (middle part of Fig. 8). During this transformation, two usability properties are injected: *Prompting* and *Cancel Support*. Besides, the AUI model is enriched by additional information to guide the user to enter the required data for the username which can be either the national ID or the phone number. Such information was inserted in the Info attribute of the correspondent Input element. As per UMAD suggest, this information is putted in the label (between parenthesis) of the generated input element in the concrete UI model. Concerning the *Cancel Support*, a new button called "Cancel" is added to the generated window in the CUI model as is suggested per UMAD.

With regard to the second transformation that generate the Final UI model from the Concrete one (bottom part of Fig. 8), only Tapped Element Size is considered. As per Section 4.1, the consideration of such property is illustrated through the static configuration of the two attributes *layout-height* and *layout-width* for each touch target widget. The value of these attributes is fixed to 48 dp and can be changed according to the target platform (iOS or other).

5.3 Learned Lesson

Considering the case study, it becomes clear that UMAD has a great impact on the generated UI with regard to the usability properties, at least to some extent. The injected usability properties are fulfilled in the generated UI and consequently the usability of such a UI is improved. In addition, the case study raised new issues that need to be considered in future work. The most important one is about the evaluation of the generated UI to investigate whether it meets the required level of usability or not.

6 Conclusion

In this paper, we proposed UMAD approach that aims to enhance MBUID method dedicated for the generation of mobile apps to support usability properties. The proposed approach is generic and can be easily instantiated by any MBUID method to blend usability properties with its conceptual primitives during the development of a mobile app. UMAD goes through 3 stages: the first identifies the conceptual primitives of the MBUID method that could be concerned by usability properties. The second stage focus on the injection of the usability properties in model compiler. Last but not least, the execution stage generates the code of the application according to the specified functional and non-functional requirements. To prove the feasibility of UMAD approach, we applied it, in this paper, to the MBUID-method presented in [23]. The obtained UI proves the ability of UMAD to enhance the usability of the app.

In terms of future work, we would like to examine the evaluation of the generated UI to check whether it meets the required level of usability (user satisfaction) or not.

Funding Statement: This project was supported by the Deanship of Scientific Research at Prince Sattam bin Abdulaziz University under the research project \#2021/01/17815.

Conflicts of Interest: The authors declare that they have no conflicts of interest to report regarding the present study.

References

[1] P. Anachack and D. Dolgaya, "Evolution of mobile applications," *MATEC Web of Conferences*, vol. 155, pp. 1–7, 2018.

[2] A. Arshad, K. Li, C. Feng, S. M. Asim, A. Yousif *et al.,* "An empirical study of investigating mobile applications development challenges," *IEEE Access*, vol. 6, pp. 17711–17728, 2018.

[3] E. Umuhoza and M. Brambilla, "Model driven development approaches for mobile applications: A survey," *Mobile Web and Intelligent Information Systems*, vol. 9847, pp. 93–107, 2016.

[4] M. Shamsujjoha, J. Grundy, L. Li, H. Khalajzadeh and Q. Lu, "Developing mobile applications via model driven development: A systematic literature review," *Information and Software Technology*, vol. 140, no. 4, pp. 106693, 2021.

[5] I. Qasim, F. Azam, M. Anwar, H. Tufail, T. Qasim *et al.,* "Mobile user interface development techniques: A systematic literature review," in *2018 IEEE 9th Annual Information Technology, Electronics and Mobile Communication Conf. (IEMCON)*, Vancouver, Canada, pp. 1029–1034, 2018.

[6] S. Vale and S. Hammoudi, "Context-aware model driven development by parameterized transformation," *Proceedings of MDISIS*, pp. 121–133, 2008.

[7] G. Meixner and G. Calvary, "Introduction to model-based user interfaces," *Group Note 7 W3C*, 2014.

[8] G. Meixner, F. Paternò and J. Vanderdonckt, "Past, present, and future of model-based user interface development," *i-com*, vol. 10, pp. 2–11, 2011.

[9] G. Calvary, J. Coutaz, D. Thevenin, Q. Limbourg, L. Bouillon *et al.,* "A unifying reference framework for multi-target user interfaces," *Interacting with Computers*, vol. 15, no. 3, pp. 289–308, 2003.

[10] M. Biehl, "Literature study on model transformations," 2010. [Online]. Available: http://staffwww.dcs.shef.ac.uk/people/A.Simons/remodel/papers/BiehlModelTransformations.pdf.

[11] A. Kleppe, J. Warmer and W. Bast, "MDA Explained: The Model Driven Architecture: Practice and Promise," Addison-Wesley, 2003. [Online]. Available: https://www.amazon.com/MDA-Explained-Architecture%C2%BF-Practice-Promise/dp/032119442X.

[12] H. Tufail, F. Azam, M. W. Anwar and I. Qasim, "Model-driven development of mobile applications: A systematic literature review," in *2018 IEEE 9th Annual Information Technology, Electronics and Mobile Communication Conf. (IEMCON)*, Vancouver, Canada, pp. 1165–1171, 2018.

[13] F. Freitas and P. H. M. Maia, "Justmodeling: An mde approach to develop android business applications," in *2016 VI Brazilian Symp. on Computing Systems Engineering (SBESC)*, Paraíba, Brazil, pp. 48–55, 2016.

[14] A. Sabraoui, A. Abouzahra, K. Afdel and M. Machkour, "Mdd approach for mobile applications based on dsl," in *2019 Int. Conf. of Computer Science and Renewable Energies (ICCSRE)*, Agadir, Morocco, pp. 1–6, 2019.

[15] T. Channonthawat and Y. Limpiyakorn, "Model driven de-velopment of android application prototypes from windows navigation diagrams," in *2016 Int. Conf. on Software Networking (ICSN)*, Jeju Island, Republic of Korea, pp. 1–4, 2016.

[16] A. Seffah, M. Donyaee, R. Kline and H. Padda, "Usability measurement and metrics: A consolidated model," *Software Quality Journal*, vol. 14, no. 2, pp. 159–178, 2006.

[17] K. Moumane, A. Idri and A. Abran, "Usability evaluation of mobile applications using iso 9241 and iso 25062 standards," *SpringerPlus*, vol. 5, no. 1, pp. 1–15, 2016.

[18] A. Barros, R. Leitão and J. Ribeiro, "Design and evaluation of a mobile user interface for older adults: Navigation, interaction and visual design recommendations," *Procedia Computer Science*, vol. 27, pp. 369–378, 2014.

[19] F. Nayebi, J. M. Desharnais and A. Abran, "The state of the art of mobile application usability evaluation," in *2012 25th IEEE Canadian Conf. on Electrical and Computer Engineering (CCECE)*, Montreal, QC, Canada, pp. 1–4, 2012.

[20] D. Zhang and B. Adipat, "Challenges, methodologies, and issues in the usability testing of mobile applications," *International Journal of Human-Computer Interaction*, vol. 18, no. 3, pp. 293–308, 2005.

[21] R. Harrison, D. Flood and D. Duce, "Usability of mobile applications: Literature review and rationale for a new usability model," *Journal of Interaction Science*, vol. 1, no. 1, pp. 1–16, 2013.

[22] L. Ben Ammar, A. Trabelsi and A. Mahfoudhi, "Incorporating usability requirements into model transformation technologies," *Requirements Engineering*, vol. 20, no. 4, pp. 465–479, 2015.

[23] L. Ben Ammar, "An automated model-based approach for developing mobile user interfaces," *IEEE Access*, vol. 9, pp. 51573–51581, 2021.

[24] L. Ben Ammar, "A usability model for mobile applications generated with a model-driven approach," *International Journal of Advanced Computer Science and Applications*, vol. 10, pp. 140–146, 2019.