frontiers | Frontiers in Computer Science

TYPE Original Research
PUBLISHED 31 May 2024
DOI 10.3389/fcomp.2024.1397805

OPEN ACCESS

EDITED BY
Lorenzo Bettini,
Universitá di Firenze, Italy

REVIEWED BY
Maria Teresa Rossi,
University of Milano-Bicocca, Italy
Tomasz Górski,
University of Gdansk, Poland

*CORRESPONDENCE
Atif Mashkoor
✉ atif.mashkoor@jku.at

RECEIVED 08 March 2024
ACCEPTED 02 May 2024
PUBLISHED 31 May 2024

CITATION
Zafar H, Ur Rehman Khan S, Mashkoor A and Nisa HU (2024) MOBICAT: a model-driven engineering approach for automatic GUI code generation for Android applications. *Front. Comput. Sci.* 6:1397805. doi: 10.3389/fcomp.2024.1397805

MOBICAT: a model-driven engineering approach for automatic GUI code generation for Android applications

Haroon Zafar[1], Saif Ur Rehman Khan[2], Atif Mashkoor[3]* and Habib Un Nisa[2]

[1]Department of Computer Science, National University of Computer and Emerging Sciences (NUCES), Peshawar, Pakistan, [2]Department of Computing, Shifa Tameer-e-Millat University (STMU), Islamabad, Pakistan, [3]Institute for Software Systems Engineering, Johannes Kepler University, Linz, Austria

Introduction: Mobile applications have become indispensable in our daily lives. However, mobile application development faces several challenges, including limited resources, budget, and time to market. The current state of the practice intends to develop the Graphical User Interface (GUI), business logic, and the controller class separately, which is a time-consuming and error-prone process. The generation of GUI is a significant concern in the development of mobile applications.

Methods: This work presents a model-driven engineering approach for automatic GUI code generation for Android applications, which intends to address the above-mentioned challenges in mobile app development. The proposed approach involves modeling domain-specific features of mobile applications and capturing requirements using UML diagrams that lead to automated GUI generation and controller class creation. We develop a Model-Based GUI Code Generator (MOBICAT) tool to provide automation support to the proposed approach.

Results: The efficacy of the MOBICAT tool is evaluated by comparing it with the baseline techniques using three open-source applications. The results indicate that the MOBICAT tool significantly outperforms the baseline techniques by attaining improved execution progress, effectively reducing development cost and effort.

Discussion: The MOBICAT tool, offers a promising solution to challenges in mobile app development. By automating GUI generation and controller class creation, it streamlines development processes and enhances productivity.

KEYWORDS

graphical user interface (GUI), mobile applications, model-driven engineering, GUI profile, code generation

1 Introduction

Nowadays, mobile applications play a vital role in our daily lives. Inspired by this, mobile application development has attained emerging growth in the software industry and is one of the most focused areas of software development (Jha and Mahmoud, 2019). According to Park (2018), ∼5.7 million mobile devices are registered, while ∼40,000 mobile applications are added to the Play Store per month (Pham et al., 2016). According to recent statistics, mobile application development is rapidly increasing and has become a billion-dollar industry (Acosta-Vargas et al., 2019). With the wide range of mobile applications, the expectations of mobile application users are also increasing. Compared with traditional software systems, mobile application development has limited

time, resources, screen size, input methods, memory, and processing speed (Vegendla et al., 2018). The above-mentioned constraints cannot be ignored during the development and testing of mobile applications. Apart from the business logic, the graphical user interface (GUI) of mobile applications in terms of resource consumption and time is also a significant requirement for the mobile user.

Mobile applications are event-centric and have rich GUIs. The GUI is vital in interacting with the mobile application and the system. Generating a GUI remains a core part of the development phase of a mobile application (Sabraoui et al., 2019). GUI code implementation is a time-consuming task and prevents the mobile application developers from giving the maximum implementation time required for features and methods of the application under development (Beltramelli, 2018). Moreover, manually developing the GUI of a mobile application is regarded as a tedious and repetitive task, especially when a change occurs in the required feature of an activity. During the development phase, the GUI should provide a functional interface for user interaction (UI) and build an intuitive nature and pleasant user experience. At the same time, mobile applications are crucial for success in a competitive market (Nudelman, 2013; Taba et al., 2014). GUI development of an application includes two separate activities: (i) UI designing and (ii) UI implementation. UI designing requires a proper mechanism for user interactions, architecture information, and visuals of the UI. In contrast, UI implementation involves properly developing layouts and widgets of the GUI framework (Allamanis et al., 2016).

Traditionally, mobile application development has involved generating GUI, controller class, and business logic (Allamanis et al., 2016). However, this approach is deemed tedious and resource-intensive, particularly as the complexity of maintaining changing activity requirements escalates over time (Joorabchi et al., 2013; Heitkötter et al., 2015). Additionally, GUI development is inherently tied to the mobile device and platform, with a significant aesthetic component that hinders automation (Usman et al., 2017). In the literature, several approaches based on model-driven development (MDD) have been reported, including IFML (OMG, 2016) and LIZARD (Botturi et al., 2013; Sabraoui et al., 2013; Marin et al., 2015); however, to the best of our knowledge, none of the reported approaches handles all of the features related to the GUI and may be leveraged. This inspires us to automatically generate the Controller class for interaction between the business logic and user interface tiers, which existing approaches lack (Usman et al., 2017).

Consequently, an opportunity exists to innovate by automating the generation of the Controller class, a feature lacking in existing approaches (Usman et al., 2017). We propose a model-driven engineering approach for automatically generating GUI code for Android applications to tackle these challenges. Our approach leverages unified modeling language (UML)—based GUI profiling[1] to allow application designers to model domain-specific GUI concepts during mobile application modeling. UML's status as an industry standard for object-oriented software system modeling makes it an ideal choice for specifying requirements, navigation flow, and lifecycle events of the application under development (Allamanis et al., 2016). We adopt a minimalist approach with UML subset diagrams, including a UML use case diagram for requirements gathering and a UML sequence diagram for modeling mobile application behavior.

Furthermore, we introduce MOBICAT, a mobile application GUI generation tool, to automate our proposed approach for Android applications. We validate our approach by applying it to three open-source Android applications across different categories and empirically assess its practical applicability. The major Research Contributions (RCs) of this work are as follows:

- **RC1:** Provides an extensive overview of the current state-of-the-art approaches and identifies various challenges of generating GUI in model-driven engineering.
- **RC2:** Develops a UML-based GUI modeling profile for mobile applications to define domain-specific GUI characteristics.
- **RC3:** Implements a MOBICAT tool for automatically generating GUI source code for Android applications.
- **RC4:** Empirically assess the practical applicability of the proposed approach by applying it to three different categories of open-source applications.
- **RC5:** Performs a comparative analysis between the existing model-driven GUI development approaches and the proposed approach.

The remaining part of the article is organized as follows: Section 2 presents the related work, while Section 3 describes the example application. The proposed approach is discussed in Section 4. Section 5 describes the developed MOBICAT tool that implements the proposed approach. Section 6 evaluates the proposed approach by employing different types of applications. Section 7 discusses the results and discussion of the obtained results, while Section 8 mentions the threats to validity. The research implications are described in Section 9. Finally, Section 10 provides the conclusion and future work.

2 Related work

This section presents existing studies on developing graphical user interface (GUI) approaches in Section 2.1 and model-driven development approaches in Section 2.2.

2.1 Graphical user interface generation approaches for mobile application

This section discusses GUI generation approaches for mobile applications, an active research area in mobile application development (Akiki et al., 2014; Núñez et al., 2020). These approaches can automatically generate GUI code for mobile applications and fall into different categories: modeling, static, and dynamic analysis approaches.

Model-based approaches widely used in the industry utilize UML diagrams and UML modeling profiles (e.g., IFML; Sabraoui et al., 2012; Bernaschina et al., 2018 and LIZARD; Botturi et al., 2013; Sabraoui et al., 2013; Marin et al., 2015; Planas et al., 2021) to generate native GUIs for mobile applications. Bernaschina et al. (2018) proposed using UML sequence and class diagrams to design

1 https://github.com/xpoiledbrat/GUIModelingProfile

the GUI. They recommend building the application from scratch to consider the complete lifecycle.

da Silva and Brito e Abreu (2014) defined a model-driven development (MDD) approach to generate GUIs for Android business data systems. They used a UML class diagram for structural modeling and integrated GUI navigation through textual annotations. Object Constraint Language (OCL) was used to define business rules. However, this approach is limited to the Android platform and focuses only on creating the structural elements of the app.

Static and dynamic analysis approaches are also used for GUI generation (Ruiz et al., 2019). Yang et al. (2013) proposed a technique to extract a mobile app model under development. It statically extracts the supported events from the app's GUI using static analysis and exercises the model's events through dynamic crawling.

Min et al. (2011) proposed an approach to build Windows mobile applications using a UML meta-model and modeling profiles for Windows mobile platform-specific concepts. They employ the model-view-controller (MVC) design pattern for GUI and hardware interaction. However, their approach lacks behavioral aspects, supports only a specific mobile platform, and does not handle feature-based variability.

Other approaches involve deep learning, artificial intelligence, and machine learning techniques. Chen et al. (2019) presented an approach to generate cross-platform GUIs for different mobile app platforms. Their framework inputs UI pages and outputs complete GUI code for iOS or Android platforms using deep learning and image processing classification. The framework requires no input besides UI pages and enables large-scale code generation without platform limitations. However, the proposed framework may face challenges in accurately handling complex UI designs that require specific platform-dependent features.

Franzago et al. (2014) presented a cooperative framework for designing data-intensive mobile applications. Their approach includes four models: data model, navigation model, user interaction model, and business model. These models collaborate to produce the mobile app code for multiple platforms. However, the detailed functioning mechanism of their framework and support for feature-based variability are not extensively explained.

2.2 Model-driven development approaches for mobile application

Model-driven development (MDD) approaches in mobile application development aim to ensure quick delivery, deployment, and time to market (Usman et al., 2014). These approaches can be categorized into three main categories: UML-based, feature-based, and others. Various modeling tools like Papyrus, RSA, and MagicDraw are available to model the app's user interface design (Safdar et al., 2015).

UML-based approaches utilize a subset of UML and UML profiles to capture GUI concepts. For example, PELLET is a UML-based profile that facilitates performance testing (Usman et al., 2020). Son et al. (2013) proposed an approach for code generation using Meta Object Facility (MOF) and UML message sequence diagram (MSD). Ko et al. (2012) introduced a domain-specific language (DSL) for generating mobile applications specific to business applications based on the MVC design pattern.

Usman et al. (2008) presented UJECTOR, a tool that generates Java source code using UML models such as class, activity, and sequence diagrams. It integrates behavioral and structural aspects of object-oriented applications. Feature-based approaches are widely used in software product line engineering applications (Safdar et al., 2020). Tools like MOPPET (Usman et al., 2017), FMP (Czarnecki et al., 2005), and FeatureID (Thüm et al., 2014) support these approaches using basic or multi-objective feature models.

Other approaches employ new notations and guidelines. AMOGA (Salihu et al., 2019) uses a crawling approach to investigate application performance based on event lists. The study does not extensively discuss potential scalability issues when applying the AMOGA strategy to larger, more complex mobile applications. Abbors et al. (2012) proposed a model-based performance testing approach using automata to describe user interactions and measure web application and service performance.

Jia and Jones (2012) introduced AXIOM, a cross-platform MDD approach for mobile app development. It uses the Abstract Model Tree (AMT) for model transformations and source code generation, starting with portraying requirements using AXIOM DSL and enhancing the models with platform-specific components. However, adequate tool support was lacking in creating, maintaining, and understanding complex models derived from UML standards.

On the other hand, Qasim et al. (2020) introduced the Model-driven Mobile HMI Framework (MMHF) to enhance human machine interfaces (HMIs) for industrial control systems, integrating a UML profile diagram for modeling mobile HMI systems within industrial settings. This UML Profile includes multiple stereotypes introducing domain-specific concepts for mobile HMI in industrial control systems. The authors detailed the implementation of the proposed model and validated its effectiveness through benchmark case studies. Górski (2021) introduced a solution for continuously delivering business applications within distributed ledger technology (DLT) networks, filling a research gap in this area. It implements two Jenkins-based continuous delivery pipelines: one for preparing the application and another for generating node deployment packages. The Smart Contract Design Pattern is used for application development, while UML and UML Profile for Distributed Ledger Deployment are employed for modeling the blockchain network installation.

2.3 Analysis of current state-of-the-art in GUI and MDD approaches

After conducting an extensive literature review on GUI and model-driven development (MDD) approaches, it was observed that none of the existing studies fully support all the features of GUI and user interaction events (UIEvents). Most approaches have limited scope, focusing on specific features in their selected examples. While some approaches target both GUI generation and UIEvents, they still lack in covering all aspects (Usman et al., 2017).

In our proposed approach, we aim to combine GUI components and model-driven engineering to leverage the benefits

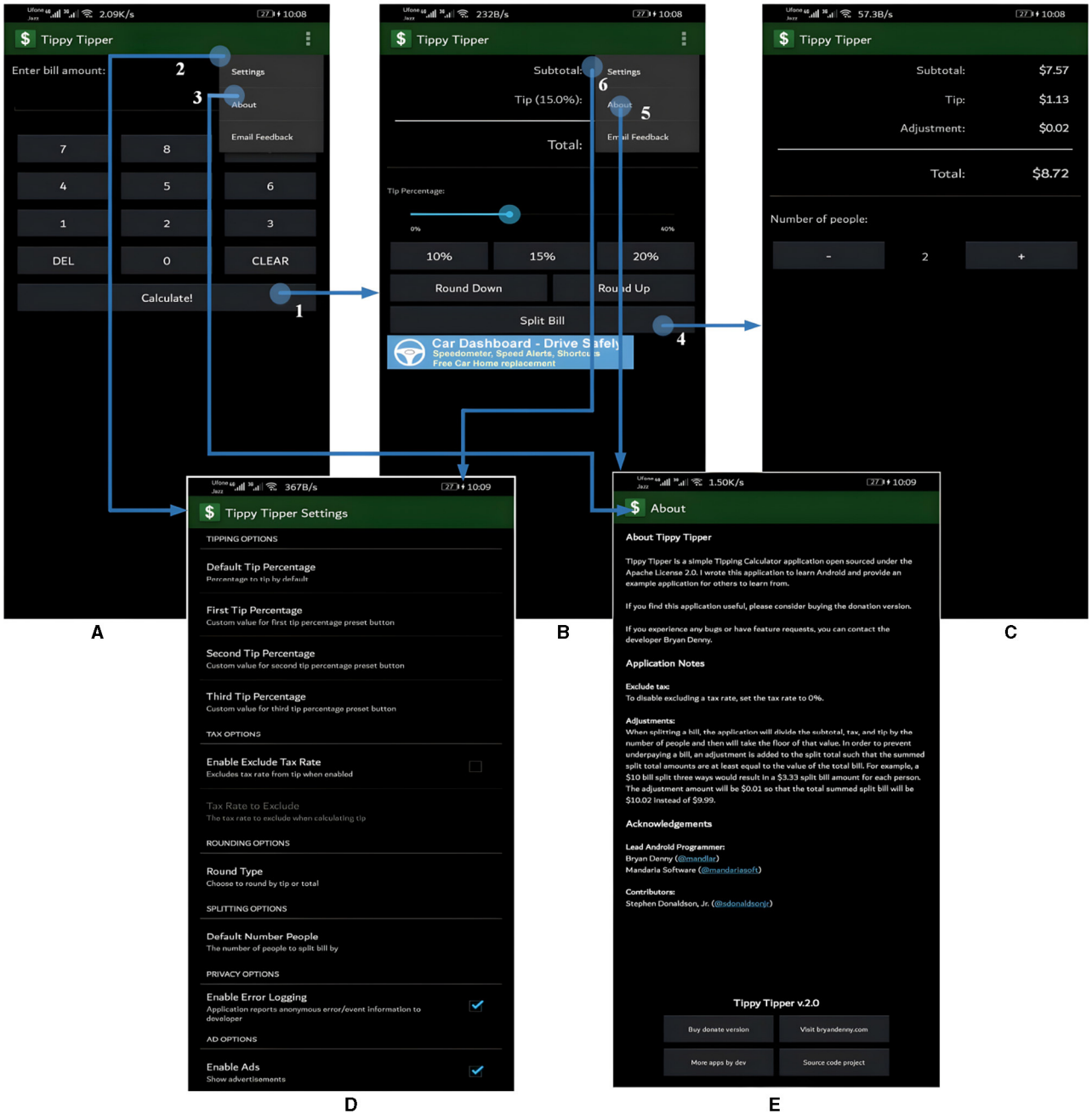

FIGURE 1
TippyTipper case study screens. **(A)** Entering the amount. **(B)** Calculated amount. **(C)** SplitBill layout. **(D)** About info. **(E)** Settings.

of both paradigms. Model-driven engineering (MDE) allows us to capture application-specific information and model the application flow along with UIEvents. Our approach supports GUI generation and UIEvents, specifically for the Android mobile application platform.

3 TippyTipper application

This section describes the example Android application on which the proposed approach is illustrated. The Android application is named "TippyTipper" and is an open-source simplified version of the Android application (Google, 2020a). The application is used to calculate the tip amount for a meal. The Android application is comprised of five screens. Figure 1 shows the complete flow of the working application.

When the application is open, the first screen Figure 1A visible to the user is about entering the amount of the meal through a numeric keyboard available on the (Input) screen (Figure 1). On the input screen, two more buttons exist, i.e., *CLEAR* and *DEL*. Both buttons have UI events, e.g., onLongClick. The *CLEAR* button on the long press erases all of the given data, while the DEL button only

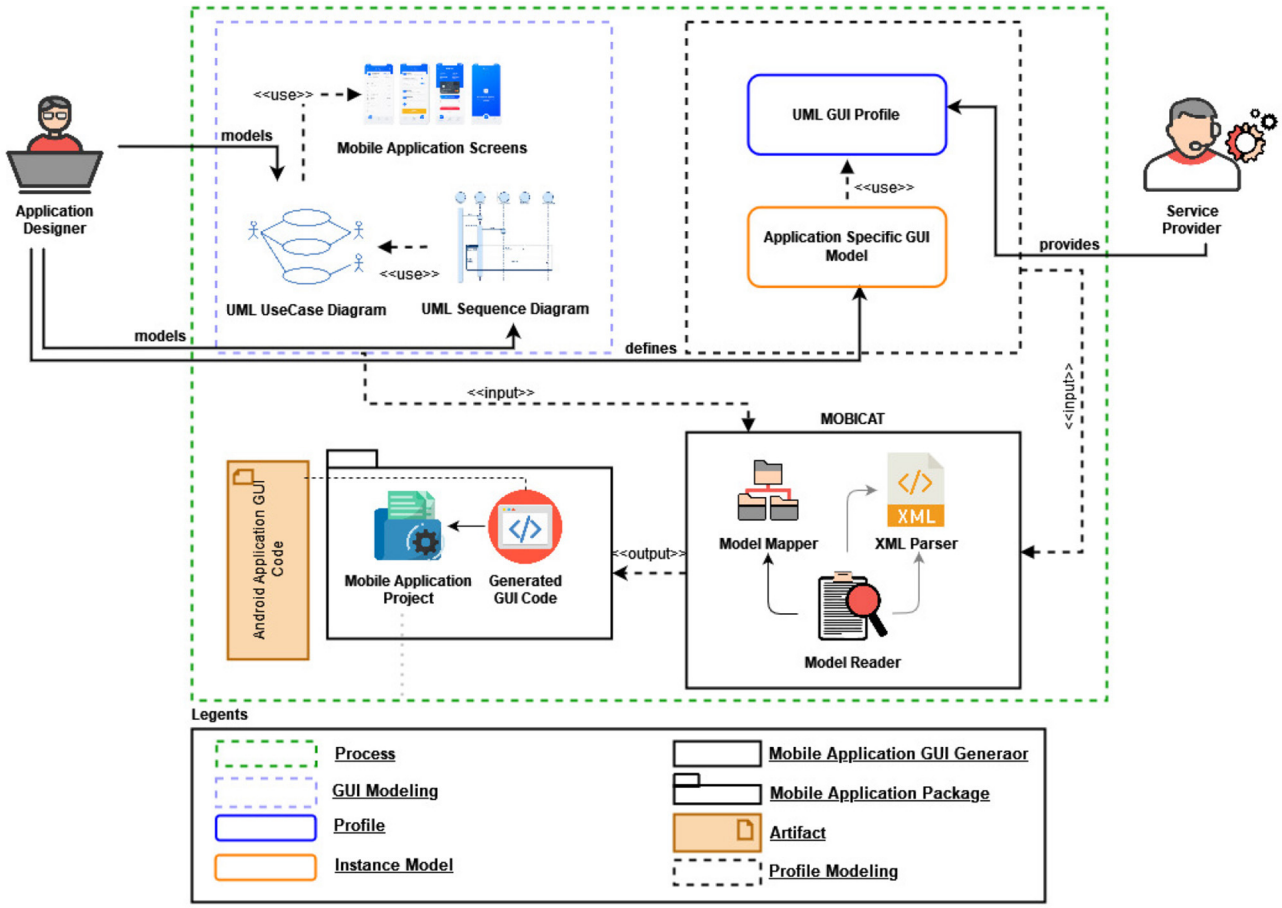

FIGURE 2
A high-level view of the proposed MDE GUI approach.

erases one digit. Pressing the Calculate button navigates the user to the next screen Figure 1B, which shows the calculated amount and the added tip amount. The third screen Figure 1C is the SplitBill layout, which shows the divided calculated amount among the total number of persons having the meal. The last two screens can be opened from either the Input screen or the total calculated amount screen using the menu items in the action bar of the layout. The *About* choice on the menu indicates the fourth screen Figure 1D, named About, with info about the application. The *Settings* choice on the menu navigates the user to the fifth screen Figure 1E.

4 Proposed approach

This section provides a detailed overview of the proposed model-based approach for GUI code generation for mobile applications (Android applications). The proposed code generation approach provides ease in managing the components of the GUI along with the appropriate event for user interaction. The proposed approach is developed to support the GUI so that the application designer will stipulate the components of GUI and events, and the code will be generated for an Android application.

Figure 2 demonstrates the conceptual model of the proposed approach. In the proposed model, we distinguish between two key roles: the application designer and the service provider. Each role plays a distinct yet complementary part in the development process, contributing to the project's overall success. The role of the service provider primarily involves the technical implementation of the proposed approach for Android applications. Specifically, the service provider guides the application designer in translating conceptual ideas into tangible GUI designs for the application under development (AUD). Additionally, they are responsible for creating GUI profiles using core components or widgets, along with appropriate event associations. Collaboration with the application designer ensures alignment with project objectives and technical feasibility. Determining GUI features for the profile is based on domain analysis, a foundation for identifying prevalent GUI components relevant to the targeted application domains.

On the other hand, the application designer serves as the architect of the user experience. They are responsible for capturing user requirements and defining the navigation flow within the AUD. This involves capturing user requirements and AUD navigation flow, followed by GUI profile modeling and mobile application layout modeling using UML diagrams.

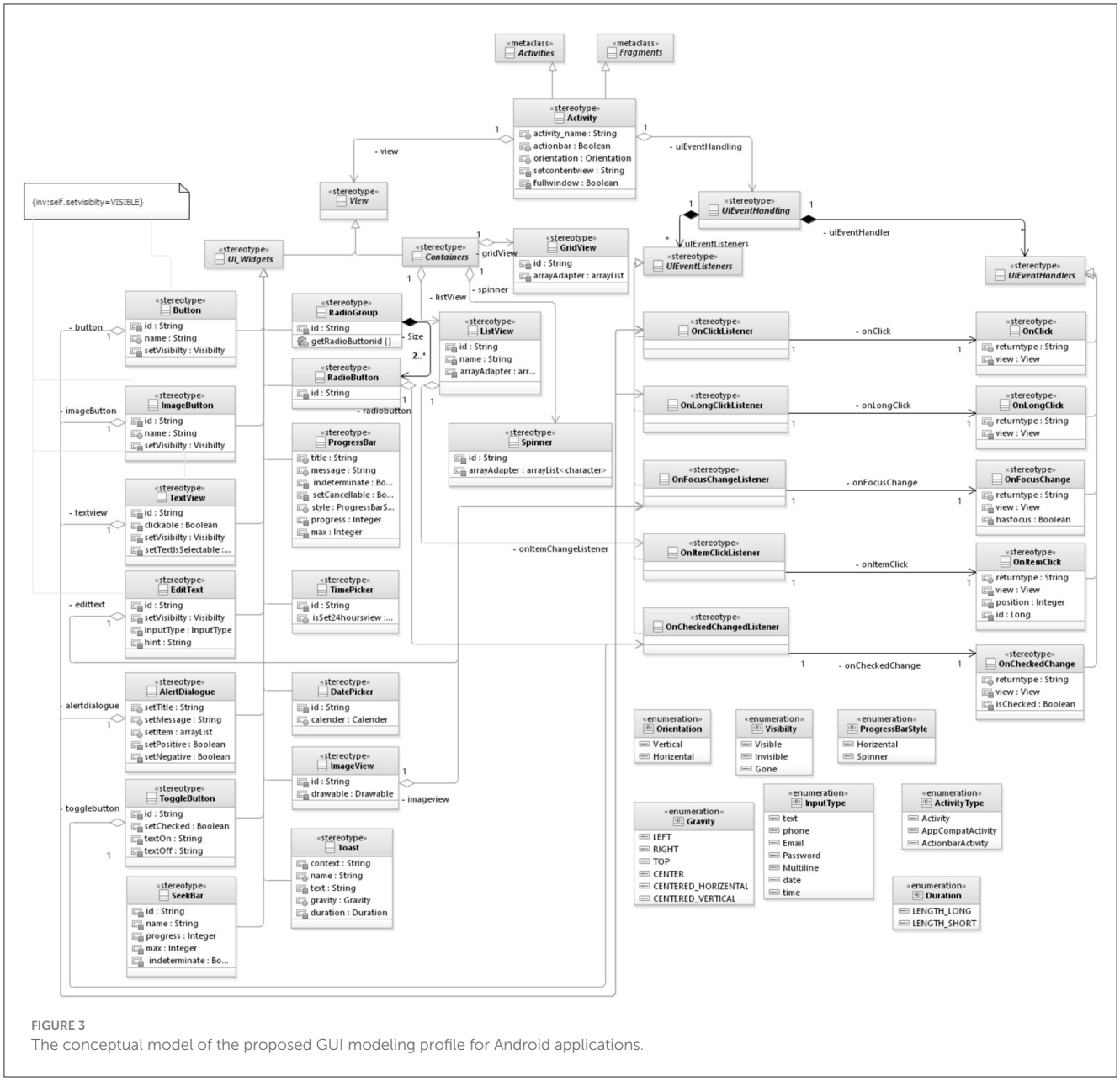

FIGURE 3
The conceptual model of the proposed GUI modeling profile for Android applications.

Additionally, the application designer utilizes model-driven engineering techniques to address GUI component design.

The application designer selects use case and sequence diagrams from the UML diagrams for mobile application modeling. The UML use case diagram, in association with the mobile application screens, captures the requirements of the mobile application. The application designer models the UML sequence diagram at the unit level to depict the navigation among the application activities.

The application-specific GUI model and the UML diagrams are utilized to generate code for the GUI of a particular mobile application. The MOdel-Based GUI Code GenerATor (MOBICAT) tool supports automating the process of creating the GUI. The tool's main inputs are the GUI profile and UML diagrams; finally, MOBICAT provides complete GUI-related code as an output.

4.1 UML-based GUI modeling profile

This section proposes a GUI profile based on UML that allows designers to capture domain-specific GUI components for mobile applications. The profile definition methodology recommended by Selic (2007) is utilized. Various GUI features are identified based on domain analysis and open sources. The significant features defined for service providers include Widgets (e.g., Button, ProgressBar, Toasts, and EditText), Containers (e.g., ListView, spinner, and GridView), and UI Event handling (e.g., UIEventListener and UIEventHandler).

The GUI profile diagram consists of two major classes: View and UIEventHandling as shown in Figure 3. The View package represents stereotypes corresponding to widgets and containers, while the UIEventHandling package comprises stereotypes for

TABLE 1 The description of UML-based GUI profile stereotypes.

Sr	Component of GUI	Stereotype mapping	Class
1	Button	android.widgets. Button	UIWidgets
2	RadioGroup	android.widgets. RadioGroup	UIWidgets
3	RadioButton	android.widgets. RadioButton	UIWidgets
4	ImageButton	android.widgets. ImageButton	UIWidgets
5	TextView	android.widgets. TextView	UIWidgets
6	ProgressBar	android.widgets. ProgressBar	UIWidgets
7	EditText	android.widgets. EditText	UIWidgets
8	TimePicker	android.leanback. widgets.TimePicker	UIWidgets
9	DatePicker	android.app. DatePicker	UIWidgets
10	ImageView	android.widgets. ImageView	UIWidgets
11	ToggleButton	android.widgets. AppcompatToggleButton	UIWidgets
12	SeekBar	android.widgets. SeekBar	UIWidgets
13	ListView	android.widgets. ListView	Container
14	GridView	android.widgets. GridView	Container
15	Spinner	android.widgets. Absspinner	Container
16	Toast	android.widgets. Toast	UIWidgets
17	AlertDialogue	android.app. AlertDialogue	UIWidgets

UIEventListeners and UIEventHandlers. The Activity class is extended with metaclasses to support the lifecycle events of Android applications.

The View package includes 16 concepts, such as Button, RadioGroup, ImageButton, TextView, ProgressBar, EditText, TimePicker, DatePicker, Image-View, ToggleButton, SeekBar, ListView, GridView, Spinner, Toast, and AlertDialogue. The UIEventHandling package is divided into UIEventListeners and UIEventHandlers, each containing five concepts.

Table 1 presents the components of the GUI, their stereotype mapping, and their sources. The GUI components are categorized into View and UIEventHandling, further divided into UI widgets and containers. The components are sourced from previous literature, Android Studio (the official IDE for Android), and developers.android (official open-source website) (Developer.Android, 2020). There are 17 components, with three belonging to the container class (ListView, GridView, and Spinner) and 14 to UIWidgets (Button, ProgressBar, AlertDialogue, ImageButton, and so on). The mappings and sources vary for each component.

```
context ProgressBar inv:
        self.setCancellable =True
        and self.indeterminate =False
        and self.progress.value = 0
        and self.max.value = 100
        and self.setCancellable =True
```

FIGURE 4
OCLconstraint for ProgressBar.

The GUI profile includes enumerations for managing component properties, such as Orientation, Visibility, ProgressBarStyle, and Gravity. ActivityType and InputType specify the type of activity and input for components. Duration is an enumeration for the timespan of widgets. Figure 4 shows an OCL constraint that specifies the ProgressBar widget using a particular invariant attribute.

The button stereotype, obtained from previous literature on GUI generation, is part of the UI widgets package. The Android operating system represents the button as android.widgets.Button. It is a crucial component of the GUI and facilitates user interaction with the mobile application. In this work, the button is associated with UIEvents, specifically OnClickListerner and OnLongClickListener stereotypes, which enable interaction and navigation between activities.

Table 2 presents OCL constraints that require developers/modelers to specify the button's ID and name. As depicted in Figure 3, the ID and name are expected to be string data types.

4.1.1 RadioGroup

The radiogroup stereotype, obtained from previous literature and presented in Table 1, creates a mutually exclusive selection among a set of radio buttons. When one radio button within a radiogroup is selected, any previously selected radio button in the same radiogroup is automatically deselected. This stereotype belongs to the UIWidgets package. In the Android operating system, the radiogroup is represented as android.widgets.RadioGroup. Figure 3 illustrates that a Radiogroup can contain two or more radiobuttons from the same category (2..*). Table 2 includes OCL constraints that enforce requirements for developers/modelers, such as defining the ID for the RadioGroup stereotype and restricting the size definition.

4.2 Requirements gathering of mobile application

Developing the use cases is the first step in gathering the user requirements of a system. In the proposed model-driven engineering approach, the application designer begins with

TABLE 2 GUI modeling profile OCL constraints.

Sr	Components of GUI	OCL constraints
1	Button	not self.id.oclIsUndefined() or not self.name.oclIsUndefined()
2	RadioGroup	not self.id.oclIsUndefined() and self→collect(id)→size() ≥ 2
3	RadioButton	not self.id.oclIsUndefined()
4	ImageButton	not self.id.oclIsUndefined() or not self.name.oclIsUndefined()
5	TextView	not self.id.oclIsUndefined() and self.setVisibility = Visible
6	ProgressBar	not self.title.oclIsUndefined() or not self.message.oclIsUndefined()
7	EditText	not self.id.oclIsUndefined() and not self.inputType.oclIsUndefined() and self.setVisibility = Visible
8	TimePicker	not self.id.oclIsUndefined()
9	DatePicker	not self.id.oclIsUndefined()
10	ImageView	not self.id.oclIsUndefined()
11	ToggleButton	not self.id.oclIsUndefined()
12	SeekBar	not self.id.oclIsUndefined() or not self.name.oclIsUndefined()
13	ListView	not self.id.oclIsUndefined() or not self.name.oclIsUndefined()
14	GridView	not self.id.oclIsUndefined() or not self.name.oclIsUndefined()
15	Spinner	not self.id.oclIsUndefined() or not self.name.oclIsUndefined()
16	Toast	not self.name.oclIsUndefined() or not self.text.oclIsUndefined() or not self.duration.oclIsUndefined()
17	AlertDialogue	not self.settitle.oclIsUndefined() or not self.setmessage.oclIsUndefined()

designing the real use cases (Larman, 2012) that are useful in recording and gathering the requirements (Kulak and Guiney, 2012). The real use cases explain the requirements of the mobile application with the support of mobile application screens (user interfaces) and the UML use case diagram. The concrete and typical way of writing mobile applications is grounded on modeling the use case (Bittner and Spence, 2003). The concrete style of writing the requirements of mobile applications includes detailed user interfaces while describing use cases. For that reason, mobile application screens are necessary before the modeling of use cases. Note that these mobile application screens are only to capture the primary activities of the application, and the information is only needed to handle the user activities in the mobile application.

4.2.1 Mobile application screens

Mobile application screens are essential for recording and gathering the application's requirements and can be used as a system prototype. Mobile application screens are mostly developed using native development tools (Android Studio; Google, 2020b). These development tools enable us to automatically generate the code (.xml) for the application screens.

In the proposed model-driven engineering approach, we only need the application screens to have the widget's XML tag without specifying the properties of user interface widgets. The application designer utilizes these tools to create and automatically generate the code for the mobile application screens.

Notice that the proposed approach focuses on the controller class, acting as a bridge between the user interface and business logic and specifying several properties of the GUI components of mobile applications.

4.2.2 Modeling use cases

The UML standard is used to model the use cases. The use cases are written using the style proposed by Larman (2012). Figure 4 shows the UML use case diagram of the TippyTipper case study. Figure 4 shows five use cases, including *Enter bill amount*, *Calculate total amount*, *View split bill*, *Setting*, and *About*. The use cases are designed in a highly interruptible setting where other applications' calls, messages, and notifications can interrupt the current application. This property is, by default, in all the designed use cases.

4.3 Behavior modeling of mobile application

After gathering the requirements using the use case diagram, the next step is to model the behavioral aspect of the mobile application. The UML sequence diagram obtains details of the application's navigation and Android activity lifecycle methods. The activities interacting with one another are considered classes and are modeled using the sequence diagram.

The application designer models the sequence diagram to tackle the application's navigation and activity lifecycle events. The designer follows the guidelines of Merino et al. (2018) to model the UML sequence diagram. In Figure 3, the application designer develops a UML sequence diagram for each class extracted from the requirements modeling. The system sequence diagram of the TippyTipper case study is shown in Figure 5. An appropriate gesture or UIEvent is modeled in each class, and the lifecycle methods are triggered while capturing their interaction with the

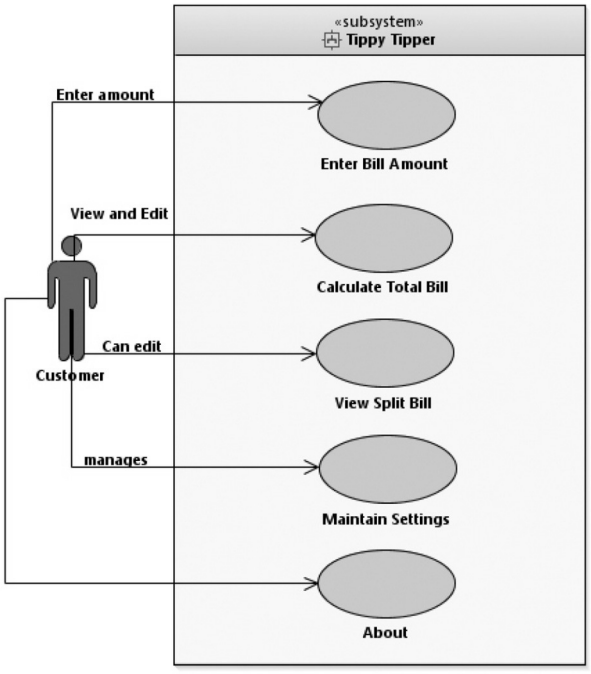

FIGURE 5
The UML use case diagram for the TippyTipper mobile application.

mobile application. Figure 5 shows five classes that have necessary navigation information among the activities. The sequence diagram modeled by the application designer should also specify the flow of interaction and the methods used to navigate among them. The menu options that appeared in the action bar or the activity are also designed in Figure 5.

For the TippyTipper case study, when the user enters the bill amount, the onCreate activity lifecycle method is triggered in the TippyTipper class, and the appropriate application screen appears. After Entering the bill amount, the user taps the calculate button, the OnClick UIEvent is called, and it navigates the user to the Total class in the application's directory. Also, by pressing the back button, the onResume lifecycle method is triggered, and the user is redirected back to the main activity.

5 Proposed model based GUI code generator (MOBICAT) tool

The MOBICAT (**MO**del-**B**ased GU**I** **C**ode Gener**AT**or) tool is developed to generate the mobile application GUI code automatically. The MOBICAT tool generates the GUI and controller codes, which bridges GUI, business logic, and the appropriate UIEvent with a particular GUI widget. To generate the GUI code, the MOBICAT tool inputs the developed instance model of the modeling profile and the UML diagrams (i.e., UML use case diagram and UML sequence diagram) of the AUD. Notice that the MOBICAT tool only supports Android applications but can be extended to other mobile application platforms.

Figure 6 presents the detailed architecture of the MOBICAT tool. The tool is developed using Eclipse IDE. The tool's architecture comprises three main components: (i) ModelReader, (ii) ModelMapper, and (iii) Application GUI generator. The ModelReader reads the UML modeling Profile and UML diagram. However, the ModelMapper maps the data according to the Android Platform template. Moreover, the Application GUI code generator generates the complete code per the directory of the mobile application.

The first component of the architecture of the MOBICAT tool is ModelReader. This component takes three inputs: (i) UML GUI Profile, (ii) Application Instance Model, and (iii) UML diagrams. The Application Instance model depends on the UML GUI profile, while the other UML diagrams are not dependent on one another. Therefore, two independent inputs exist in the ModelReader component, and the ModelReader reads the inputs in parallel. The application Instance model is developed in Eclipse IDE, which stores the Instance model in a (.xml document). The XML document is then given as input to the ModelReader, and the complete information is stored in a data structure for the GUI code generation. According to Figure 7, the information is then transferred to the next component of the tool for further processing.

The second component of the MOBICAT tool is ModelMapper. The ModelMapper takes the inputs as UML GUI Profile Information, Sequence Diagram Information, and Use Case Diagram Information, as shown in Figure 7. The component maps the information using the template or standard set by the Android platform. This component is also implemented in Eclipse IDE. The Active activity classes are generated, and the UIWidgets belonging to the specific class are mapped.

The third component in the MOBICAT tool is the Application GUI code generator, which generates the code and adds it to the appropriate directory. This component also takes two inputs: Sequence diagram information and Application information. The information about the sequence diagram is given as input to handle the navigation flow and the appropriate lifecycle methods. Figure 7 shows that the component generated the code per the Android platform and the specific directory in the mobile application package.

6 Evaluation

This section presents the research questions in Section 6.1, followed by the Experimental design in Section 6.2, and a description of example applications in Section 6.3.

6.1 Research questions

The objective of the evaluation is to investigate the effectiveness of the proposed approach in providing automated support to GUI generation for mobile applications. More precisely, the aim is to evaluate whether the proposed tool can capture and manage components of GUI, their appropriate UIEvents, limitations in the design phase, and the automation support. The overall objective of the research questions is to investigate the suitability and

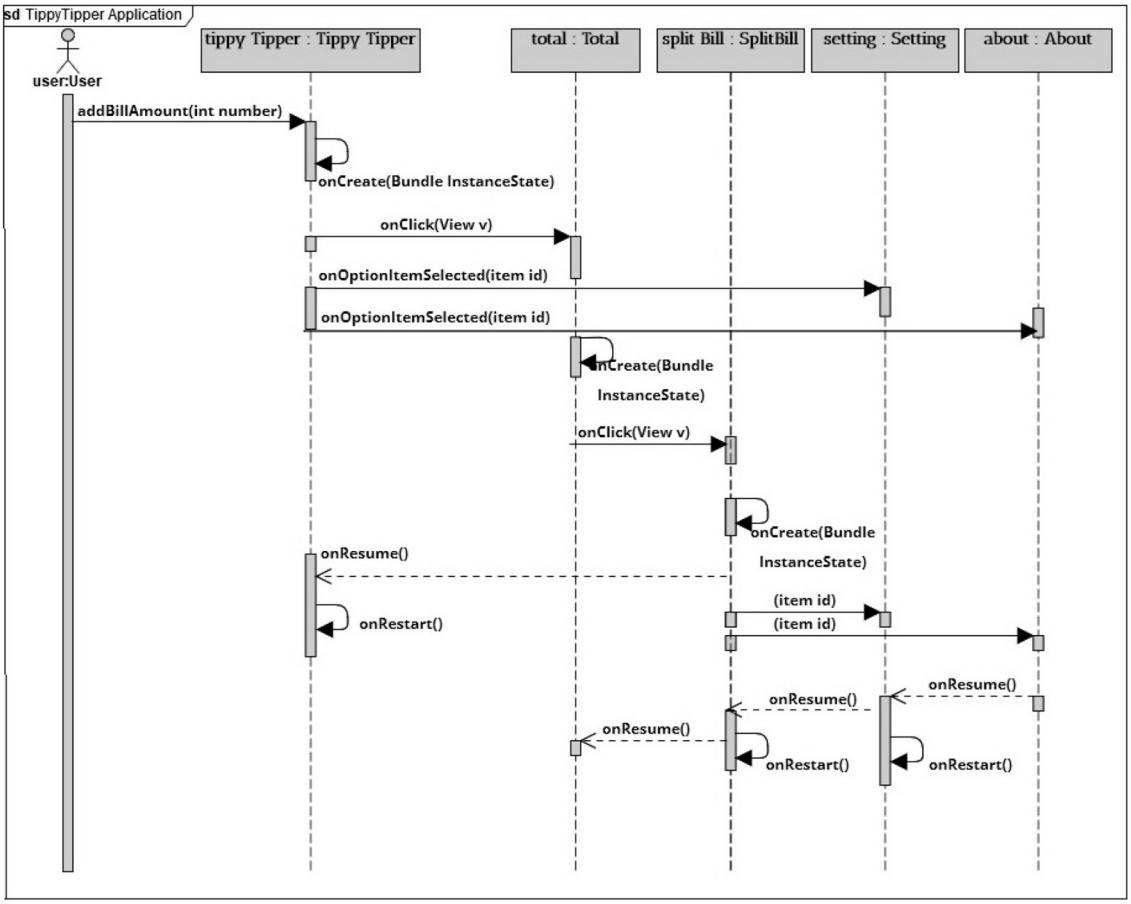

FIGURE 6
The UML sequence diagram for the TippyTipper mobile application.

applicability of the proposed approach. Based on the above objectives, the following two research questions are defined.

RQ1: *How MOBICAT is effective in capturing the GUI components*?
This research question examines whether the proposed approach captures the components of GUI and the appropriate UIEvent.

RQ2: *How MOBICAT outperforms the state-of-the-art MDE approaches*?
This research question provides a comparative analysis based on different features among the current state-of-the-art MDE approaches and MOBICAT.

6.2 Experimental design

This section illustrates the experimental design used to validate the effectiveness of the proposed approach. In Table 3, four evaluation tasks (T1–T4) are designed to answer RQ1.

For RQ2, we have implemented the existing MDE approaches in our environment and empirically evaluated them based on different types of methods related to the GUI of the application. Moreover, a comparative analysis is also performed based on the features of the approaches. Rational Software Architect (RSA)[2] is used for modeling the mobile application. Using the Ecore plugin, the proposed profile is developed in Eclipse Modeling Framework (EMF).[3] All the experimental activities are performed on a 64-bit Microsoft Windows operating system, Intel Core i3 processor (2.40 GHz), 12 GB RAM, and 750 GB HDD. The code generated from the proposed approach is executed in Android Studio using the virtual device (Huawei Y9 Prime 2019) to evaluate the output.

6.3 Example applications

The TippyTipper application is used as an example throughout the paper. This section discusses Notepad and ContactManager.

The Notepad[4] mobile application belongs to the Productivity category of the Android Play Store and is an open-source application. The application consists of three screens. When the application is launched, the MainActivity screen is shown to the user. The MainActivity Screen allows the user to

2 https://www.ibm.com/products/rational-software-architect-designer
3 http://www.eclipse.org/modeling/emf/
4 https://github.com/farmerbb/Notepad

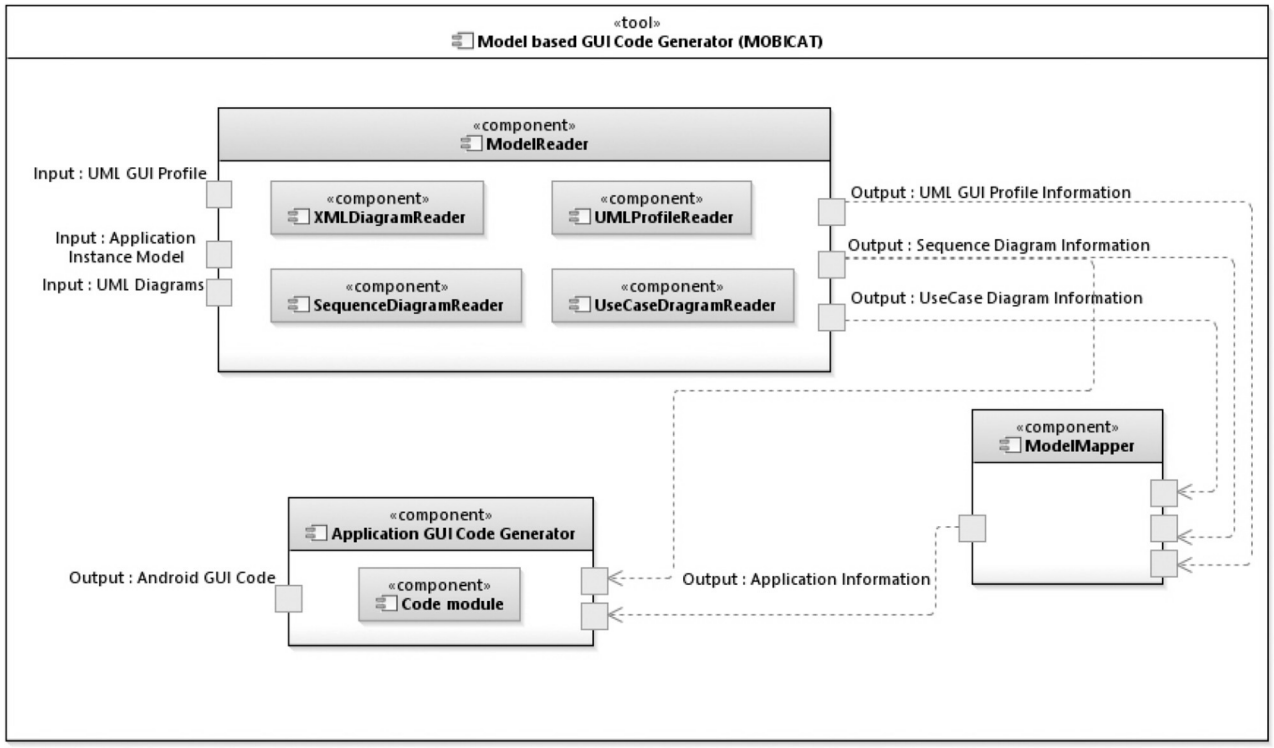

FIGURE 7
The mobile application GUI generator (MOBICAT) tool architecture.

TABLE 3 Evaluation tasks and metrics.

Column 1	Column 2	Column 3
T1	Calculating the number of GUI features captured using the view package in the proposed approach for the jth example application.	The total no. of Features in jth example application. (F_j)
T2	Calculating the number of activity methods captured using the UIEvents package in the proposed approach for the jth example application.	The total no. of methods in jth example application. (M_j)
T3	Calculating the total number of lines of code generated using the proposed approach for the example application's jth activity (a(lOC)).	Generated line of code (Total LOC=$\sum_{i=1}^{n} a_i$ (LOC))
T4	Calculating the OCL constraints applied on each component of GUI of the activity of the example application.	The total number of OCL constraints in the jth example application. (C_j)

view an existing document and also facilitates the user in adding a new document. By pressing the add document button or edit doc button, the application navigates the user to the NoteEditActivity where the user can edit the existing or new document. The third screen or layout of the application is the Setting activity. The ContactManager[5] application belongs to the Business category and is also an open-source Android application. The application helps manage and manipulate the user's contacts. The ContactManager application consists of two activities, i.e., ContactAdder and ContactManager. The ContactAdder activity allows the user to add a new contact to the application. In contrast, the ContactManager activity allows users to manage and handle the application's existing contacts.

The proposed model-driven engineering approach for automatic GUI generation in all the case studies (TippyTipper, Notepad, ContactManager) uses the UML-based GUI profile and UML diagrams. The application designer develops the proposed Profile (Application-specific) and UML diagram, i.e., use case diagram and sequence diagram. The Profile contains all the widgets and UIEvents suitable for the activity. The application-specific instance model depends on the activity's requirements and consists of features (widgets) and UIEvents.

5 https://android.googlesource.com/platform/development/

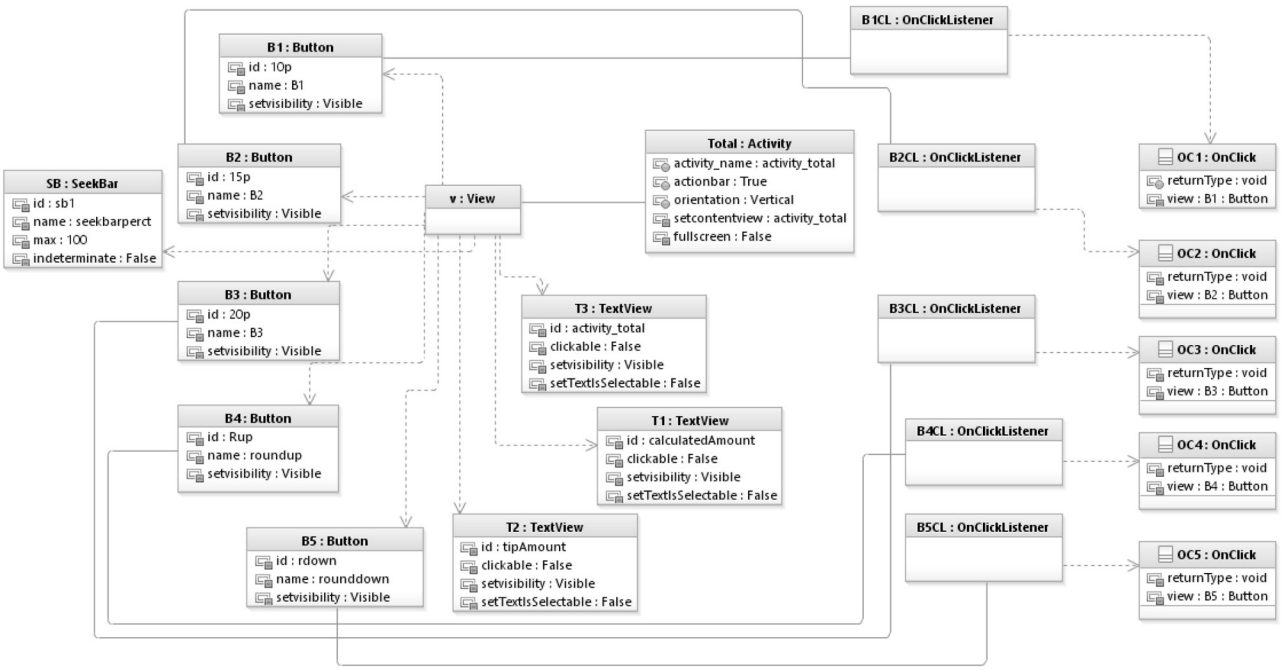

FIGURE 8
Excerpt of one of the generated instances model from GUI profile.

TABLE 4 Mobile application GUI generated code statistics.

Application	No. of activities	Activities	No. of methods (M_j)	Line of code (LOC)	Application category
TippyTipper	5	TippyTipper	18	214	Tool
		Total	10	122	
		Split Bill	5	65	
		Setting	2	60	
		About	6	92	
Notepad	3	MainActivity	12	192	Productivity
		NoteEditActivity	8	102	
		Setting	3	62	
ContactManager	2	ContactAdder	6	113	Business
		ContactManager	5	99	

7 Results and discussion

This section presents the results and discussion of the evaluation and answers the research questions (RQ1 and RQ2) discussed in Section 6.1.

7.1 Results for RQ1

This section presents the detailed results of generating the GUI of case study applications (TippyTipper, Notepad, and ContactManager) through the proposed model-driven engineering approach.

Figure 8 shows the Application-specific instance model generated using the proposed UML-based GUI profile. The Total Activity consists of five Buttons, three TextViews, one SeekBar, and the respective UIEvent, which is suitable and according to the activity's requirement. The UIEventHandling mechanism is also handled using the Listeners and Handlers appropriate to the widgets. The Listener contains only a single handler function and is limited to a single feature (widget). The Application-specific instance models show which type of UI widgets the activity contains and which widget is associated with which UI event.

Table 4 shows the statistics of the proposed model-driven engineering approach applied to the TippyTipper, Notepad, and ContactManager case study applications. For the TippyTipper android application, the proposed approach generated 41 methods,

TABLE 5 The statistics of the proposed GUI modeling profile.

Application	Features	No. of features (F_j)	OCL constraints (C_j)
TippyTipper	TextView, EditText, Button, Container	4	16
	SeekBar, Button, TextView, Container	4	14
	Button, TextView	2	8
	ListView, RadioButton, TextView	3	4
	Button, TextView, Container	3	4
Notepad	Spinner, TextView, Button, Container, TimePicker, EditText, DatePicker	6	7
	TextView, Spinner, Button, TimePicker, DatePicker	5	5
	ListView, EditText, TextView, Button, Alert dialogue	5	5
ContactManager	Spinner, TextView, Button, EditText, ImageView, Toast	5	11
	ListView, Button, RadioGroup, RadioButton, Toast	5	5

TABLE 6 Comparison of statistics between MOBICAT and existing approaches.

Approaches		UI events	Lifecycle events	Workflow model (methods)	View model (methods)	Generated lines of code
LIZARD (Marin et al., 2015)	TippyTipper	12	-	-	13	328
	Notepad	2	-	-	6	121
	ContactManager	5	-	-	3	102
Botturi et al. (2013)	TippyTipper	6	-	-	10	202
	Notepad	3	-	-	8	162
	ContactManager	1	-	-	3	129
Sabraoui et al. (2013)	TippyTipper	-	-	-	12	173
	Notepad	-	-	-	7	103
	ContactManager	-	-	-	5	66
MOBICAT	TippyTipper	38	17	16	18	553
	Notepad	13	10	9	19	356
	ContactManager	5	6	8	9	212

including all the lifecycle and user interaction methods. Of 41 methods, 18 are from the TippyTipper screen, 10 are from the total screen, five are from the SplitBill screen, two are from the setting, and six are from the About screen. The total lines of code generated from the MOBICAT tool using the proposed approach are 553. The proposed approach generated 23 methods for the Notepad application, including all the lifecycle and user interaction methods. Of 23 methods, 12 are from the MainActivity screen, 8 are from the NoteEditActivity screen, and 3 are from the Setting screen.

The total line of code generated from the MOBICAT tool using the proposed approach is 356. For the ContactManager application, the proposed approach generated 11 methods, including all the lifecycle and user interaction methods. Six methods are from the ContactAdder screen, and five are from the ContactManager screen. The total line of code generated from the MOBICAT tool using the proposed approach is 212.

The statistics of generated UML-based GUI profiles for the case studies (TippyTipper, Notepad, and ContactManager) are shown in Table 5. For the TippyTipper case study, 16 stereotypes are generated using the proposed UML-based profile. Four stereotypes belong to the TippyTipper activity, four to the total activity, two to the SplitBill activity, three to the setting activity, and three to the about activity. There are 46 OCL constraints generated from the application-specific instance model. In the case of the Notepad mobile application, 16 stereotypes are generated using the proposed UML-based profile. Six stereotypes belong to the Main Activity activity, five to the NoteEditActivity activity, and five to the Setting activity. A total of 17 OCL constraints are generated from the application-specific instance model. For the ContactManager case study, ten stereotypes are generated using the proposed UML-based profile. Five stereotypes belong to the ContactAdder activity, and five are from the ContactManager activity. A total of 16 OCL constraints are generated from the application-specific instance model.

Table 6 compare the statistics of MOBICAT and existing approaches based on model-driven engineering. The results revealed that MOBICAT outperformed the other approaches LIZARD (Botturi et al., 2013; Sabraoui et al., 2013;

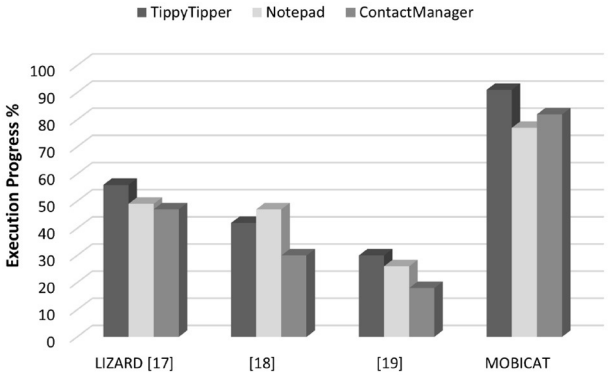

FIGURE 9
Execution progress-based comparison between MOBICAT and existing approaches.

Marin et al., 2015) because the proposed approach deals with all the components and methods (ViewModel, WorkflowModel, lifecycle, and UIEvents) related to the GUI. To evaluate the impact of the correctness of generated code, the code is evaluated using the available set of test scripts developed using ROBOTIUM[6] to verify the GUI components and methods.

Figure 9 reports the percentage of Execution progress of the generated code obtained from the test script results on the applications used in the study. The test script result shows that MOBICAT achieved maximum execution progress of 91% on the Tippytipper application, 82% on ContactManager, and a minimum of 77% on the Notepad application. In contrast, the LIZARD achieved 56, 49, and 47% execution progress on the TippyTipper app, Notepad app, and ContactManager application, respectively. Compared with MOBICAT and LIZARD, the approaches proposed in Botturi et al. (2013) and Sabraoui et al. (2013) achieved <50% execution progress. Based on the test scripts results, MOBICAT provides a maximum percentage of execution progress when applied to the considered case studies and outperforms the other reported approaches by covering all the aspects (e.g., *completeness*, *usability*, *reuse*, *interaction*, *workflow events*, and *lifecycle events*) of GUI.

The results from the case studies show that the proposed approach effectively generates GUI for mobile applications per the user's requirement. The manual development of these GUIs would have required a substantial redundant effort.

7.2 Results for RQ2

After observing the proposed model-driven engineering approach's practicality, we have compared it to the existing approaches reported in the literature. While this comparison provides valuable insights into our approach's efficacy, it may not encompass all GUI generation methodologies documented in the vast body of literature.

The decision to focus on a subset of existing solutions was made to facilitate a manageable and focused validation process. Attempting to include every possible approach would be impractical and potentially overwhelming, detracting from the depth and clarity of the analysis.

Table 7 presents the comparison of the proposed approach with existing approaches that are based on model-driven architecture. The approaches are critically evaluated using different evaluation parameters. The approach LIZARD proposed in Marin et al. (2015) deals with only two phases of the software development lifecycle (SDLC), i.e., design and development phases. The approach depends on the meta-model, but no formal language is used to deal with the constraints of the metamodel. The proposed approach supports five features (widgets) related to the GUI. LIZARD only provides a single UIEvent, such as onClick, and lacks support for the lifecycle events of the Android activity, as shown in Table 7. Android and Windows phones are targeted, and the subsequent language is generated using the proposed approach of each platform, respectively.

However, in Botturi et al. (2013), the reported approach supports the requirement phase, design phase, and development phase of the SDLC. Their proposed approach (Botturi et al., 2013) also depends on model-driven engineering and provides a meta-model; however, no formal language is used to deal with the constraints. Moreover, the approach supports seven features (widgets) related to the GUI. The approach deals with only single UIEvents and provides the code for Android and Windows phones. In contrast, in Sabraoui et al. (2013), the approach handles the design and development phase and provides no mechanism for handling Android lifecycle events. The approach is also based on model-driven engineering but lacks formal support for the constraints and cardinalities of their proposed meta-model.

The approach supports six features (widgets) related to the GUI. No support exists for user interaction events in the presented model. The approach supports both the Android and Blackberry mobile application platforms. The MOBICAT supports the requirements, design, and development phases of SDLC. Moreover, MOBICAT is based on a metamodel that handles all aspects of generating GUI for Android applications. Notice that the proposed approach effectively handles 17 features, excluding the UIEvents handling. In the proposed approach, we have provided a formal language to handle each widget's constraints related to the GUI for Android applications.

Moreover, the proposed approach utilizes the UML models for the requirement phase and handles the application's navigation flow under development. The proposed approach considers all UIEventHandling methods and listeners and provides a concrete mechanism for them. The language generation that supports the GUI using the proposed approach is JAVA.[7]

8 Threats to validity

This section addresses the threats that may influence the proposed approach's performance. The four forms of threats are external validity, internal validity, construct validity, and conclusion validity.

6 https://github.com/RobotiumTech/robotium

7 https://github.com/Intellectual-hutt/MOBICAT

10.3389/fcomp.2024.1397805

TABLE 7 Comparative analysis of MDE approaches.

Approaches	Process				No. of feature (widget(s))	Lifecycle event(s)	Meta-model	OCL	UML model	Listeners					Handlers					Languages	Android	Windows Phone	BlackBerry
	Requirement	Analysis	Development	Testing						OnClick Listener	OnLong Click Listener	OnFocus Change Listener	OnItem Click Listener	OnChecked Change Listener	OnClick	OnLong Click	OnFocus Change	OnItem Click	OnChecked Change				
Marin et al. (2015) LIZARD	✓		✓		5		✓			✓					✓					XML & C#	✓	✓	
Botturi et al. (2013)	✓		✓		7		✓		✓				✓					✓		XML	✓	✓	
Sabraoui et al. (2013)	✓		✓		6		✓													XML	✓		✓
MOBICAT	✓	✓	✓	✓	17	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	JAVA	✓		

8.1 External validity threat

The threat to the external validity of the proposed approach is its applicability to all the categories of the Android mobile application. To reduce this threat, we have selected three categories of Android mobile applications from different domains using the Google Play store, i.e., TippyTipper from the tool category, Notepad from the productivity category, and ContactManager from the Business category of mobile applications. This permits us to validate our approach to three massively different mobile applications. We do not claim that the obtained results are generalizable for all categories of mobile applications; however, this is a common threat among all the empirical studies.

8.2 Internal validity threat

The chances of the experiment's internal validity threat are related to the instrumentation during the execution of the experiment. To minimize the possibility of this threat, we evaluated the instrumented source code using the available set of test scripts to verify the GUI components and methods.

8.3 Construct validity threat

The relationship between theory and observation may threaten the experiment's construct validity. To mitigate this threat, we will have several sessions with the domain experts during the evaluation procedure.

8.4 Conclusion validity threat

The threat to the conclusion's validity is related to the challenges that affect the ability to draw an accurate conclusion. To reduce this threat, a comparative analysis is performed between the existing model-driven GUI development approaches and the proposed approach.

9 Research implications

Section 9.1 presents the research implications of the proposed approach to the software engineering research community. Section 9.2 presents the research implications to state-of-the-practice.

9.1 State-of-the-art implications

There exist various model-based graphical user interface modeling approaches (Botturi et al., 2013; Sabraoui et al., 2013; Marin et al., 2015), or UI modeling languages, i.e., IFML (OMG, 2016) for mobile applications modeling and source code generation. Still, these approaches may be leveraged (Usman et al., 2017). To that end, our proposed approach incorporates all the GUI components and model-driven engineering to gain the advantage of both theories. Moreover, the proposed approach deals with all

the methods related to the user interface of mobile applications and facilitates the SE research community with these methods.

9.2 State-of-the-practice implications

For practitioners, the current industrial practice is to develop the GUI, the controller class separately, and the business logic (Heitkötter et al., 2015). This practice is considered tedious and resource-consuming. Simultaneously, the complexity of sustaining the change in the application's requirements also exponentially increases over time (Kang et al., 1990; Joorabchi et al., 2013). To overcome this challenge, this study provides a model-based approach that automatically generates the GUI per user requirements. The results show that the proposed approach is beneficial in reducing the overall development cost and effort. Moreover, our proposed model-driven engineering approach, coupled with automation tools like MOBICAT, is tailored to meet the demand for rapid time to market, delivery, and deployment of mobile applications. By leveraging UML-based GUI modeling profiles, our approach streamlines the development process, enabling quick iteration and adaptation of GUI designs. This ensures that subsequent releases of mobile applications can be efficiently developed and deployed, allowing consecutive releases to be delivered with agility and speed.

10 Conclusion and future work

GUI development for mobile applications poses significant challenges regarding resource consumption and time to market. This paper has addressed these challenges by proposing a model-driven approach for automatically generating graphical user interfaces (GUIs) and controller classes for mobile applications. By providing a GUI modeling profile tailored for Android applications, we have enabled the specification of domain-specific GUI concepts and streamlined the development process. Through the use of the proposed approach and the developed Model-Based GUI Code Generator (MOBICAT) tool, we have successfully demonstrated the automated generation of GUIs for various categories of Android applications, including TippyTipper (Tool), Notepad (Productivity), and ContactManager (Business).

The detailed results obtained from the performed experiments demonstrate the efficacy of the proposed model-driven engineering approach for GUI generation. For instance, the application-specific instance models generated using our UML-based GUI profile showcase the appropriate UI elements and event-handling mechanisms. Moreover, the GUI-generated mobile application statistics highlight the number of activities, methods, and lines of code generated for each case study application. Additionally, the statistics on the generated UML-based GUI profiles detail each application's features, constraints, and stereotypes. Furthermore, our comparison with existing approaches underscores the superiority of our approach in terms of UI events, life cycle events, workflow models, and lines of code generated.

From a future research viewpoint, we focus on several research pointers for further research and development. First, we plan to extend our proposed approach to support other mobile application platforms beyond Android, enhancing its applicability and reach. Additionally, we intend to explore ways to integrate additional features and functionalities into the GUI modeling profile that could further enhance the flexibility and usability of the proposed approach.

Data availability statement

The original contributions presented in the study are included in the article/supplementary material, further inquiries can be directed to the corresponding author.

Author contributions

HZ: Writing—original draft, Writing—review & editing. SU: Writing—original draft, Writing—review & editing. AM: Writing—original draft, Writing—review & editing. HN: Writing—original draft, Writing—review & editing.

Funding

The author(s) declare financial support was received for the research, authorship, and/or publication of this article. This work was supported by Johannes Kepler Open Access Publishing Fund and the federal state Upper Austria.

Acknowledgments

First, we thank the Software Reliability Engineering Group (SREG) members for their continued support and feedback throughout this research. Secondly, we would like to thank Shaukat Ali (Simula Research Laboratory, Norway) for providing valuable feedback on the initial version of the manuscript.

Conflict of interest

The authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

Publisher's note

References

Abbors, F., Ahmad, T., Truscan, D., and Porres, I. (2012). "MBPeT: a model-based performance testing tool," in *2012 Fourth International Conference on Advances in System Testing and Validation Lifecycle* (Wilmington, NC: IARIA XPS Press), 31.

Acosta-Vargas, P., Zalakeviciute, R., Luján-Mora, S., and Hernandez, W. (2019). "Accessibility evaluation of mobile applications for monitoring air quality," in *International Conference on Information Technology and Systems* (Cham: Springer), 638–648.

Akiki, P. A., Bandara, A. K., and Yu, Y. (2014). Adaptive model-driven user interface development systems. *ACM Comput. Surv.* 47, 1–33. doi: 10.1145/2597999

Allamanis, M., Peng, H., and Sutton, C. (2016). "A convolutional attention network for extreme summarization of source code," in *International Conference on Machine Learning*, 2091–2100.

Beltramelli, T. (2018). "pix2code: generating code from a graphical user interface screenshot," in *Proceedings of the ACM SIGCHI Symposium on Engineering Interactive Computing Systems* (New York, NY: Association for Computing Machinery), 1–6. doi: 10.1007/978-981-16-3802-2_12

Bernaschina, C., Comai, S., and Fraternali, P. (2018). Formal semantics of OMG's Interaction Flow Modeling Language (IFML) for mobile and rich-client application model driven development. *J. Syst. Softw.* 137, 239–260.

Bittner, K., and Spence,I. (2003). *Use Case Modeling*. Boston, MA: Addison-Wesley Professional.

Botturi, G., Ebeid, E.,Fummi, F., and Quaglia, D. (2013). "Model-driven design for the development of multi-platform smartphone applications," in *Proceedings of the 2013 Forum on specification and Design Languages (FDL)* (Paris: IEEE), 1–8.

Chen, S., Fan, L., Su, T., Ma, L., Liu, Y., and Xu, L. (2019). "Automated cross-platform GUI code generation for mobile apps," in *2019 IEEE 1st International Workshop on Artificial Intelligence for Mobile (AI4Mobile)* (Hangzhou: IEEE), 13–16.

Czarnecki, K., Antkiewicz, M., Kim, C. H. P., Lau, S., and Pietroszek, K. (2005). "fmp and fmp2rsm: eclipse plug-ins for modeling features using model templates," in *Companion to the 20th Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications* (New York, NY), 200–201. doi: 10.1145/1094855.1094934

da Silva, L. P., and Brito e Abreu, F. (2014). "Model-driven gui generation and navigation for android bis apps," in *2014 2nd International Conference on Model-Driven Engineering and Software Development (MODELSWARD)* (Lisbon: IEEE), 400–407.

Developer.Android (2020). *Android Developers*. Available online at: https://developer.android.com/ (accessed February 1, 2024).

Franzago, M., Muccini, H., and Malavolta, I. (2014). "Towards a collaborative framework for the design and development of data-intensive mobile applications," in *Proceedings of the 1st International Conference on Mobile Software Engineering and Systems* (New York, NY: Association for Computing Machinery), 58–61. doi: 10.1145/2593902.2593917

Google (2020a). *Tippy Tipper Application*. Available online at: https://code.google.com/archive/p/tippytipper (accessed February 1, 2024).

Google (2020b). *Android Studio*. Available online at: https://developer.android.com/studio/index.html (accessed February 1, 2024).

Górski, T. (2021). Continuous delivery of blockchain distributed applications. *Sensors* 22:128. doi: 10.3390/s22010128

Heitkötter, H., Kuchen, H., and Majchrzak, T. A. (2015). Extending a model-driven cross-platform development approach for business apps. *Sci. Comput. Progr.* 97, 31–36. doi: 10.1016/j.scico.2013.11.013

Jha, N., and Mahmoud, A. (2019). Mining non-functional requirements from app store reviews. *Empir. Softw. Eng.* 24, 3659–3695. doi: 10.1007/s10664-019-09716-7

Jia, X., and Jones, C. (2012). "Cross-platform application development using AXIOM as an agile model-driven approach," in *International Conference on Software and Data Technologies* (Berlin, Heidelberg: Springer), 36–51.

Joorabchi, M. E., Mesbah, A., and Kruchten, P. (2013). "Real challenges in mobile app development," in *2013 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement* Baltimore, MD: IEEE.

Kang, K. C., Cohen, S., Hess, J. A., Novak, W. E., and Peterson, A. S. (1990). *Feature-Oriented Domain Analysis (FODA) Feasibility Study. No. CMU/SEI-90-TR-21*. Pittsburgh, PA: Carnegie-Mellon University, Software Engineering Institute.

Ko, M., Seo, Y., Min, B., Kuk, S., and Kim, H. S. (2012). "Extending UML meta-model for android application," in *2012 IEEE/ACIS 11th International Conference on Computer and Information Science* (Shanghai: IEEE), 669–674.

Kulak, D., and Guiney. (2012). *Use Cases: Requirements in Context*. Boston, MA: Addison-Wesley.

Larman, C. (2012). *Applying UML and Patterns: an Introduction to Object-Oriented Analysis and Design and Iterative Development*. Pearson Education India.

Marin, I., Ortin, F., Pedrosa, G., and Rodriguez, J. (2015). Generating native user interfaces for multiple devices by means of model transformation. *Front. Inf. Technol. Electron. Eng.* 16, 995–1017. doi: 10.1631/FITEE.1500083

Merino, L., Ghafari, M., Anslow, C., and Nierstrasz, O. (2018). A systematic literature review of software visualization evaluation. *J. Syst. Softw.* 144, 165–180. doi: 10.1016/j.jss.2018.06.027

Min, B. K., Ko, M., Seo, Y., Kuk, S., and Soo Kim, H. (2011). "A UML metamodel for smart device application modeling based on Windows Phone 7 platform," in *TENCON 2011-2011 IEEE Region 10 Conference* (Bali: IEEE), 201–205.

Nudelman, G. (2013). *Android Design Patterns: Interaction Design Solutions for Developers*. Hoboken, NJ: John Wiley & Sons.

Núñez, M., Bonhaure, D., González, M., and Cernuzzi, L. (2020). A model-driven approach for the development of native mobile applications focusing on the data layer. *J. Syst. Softw.* 161:110489. doi: 10.1016/j.jss.2019.110489

OMG (2016). *The Interaction Flow Modeling Language (IFML)*. Available online at: https://www.ifml.org/ (accessed October 7, 2019).

Park, D. S. (2018). Future computing with IoT and cloud computing. *J. Supercomput.* 74, 6401–6407. doi: 10.1007/s11227-018-2652-7

Pham, X. L., Nguyen, T. H., Hwang, W. Y., and Chen, G. D. (2016). "Effects of push notifications on learner engagement in a mobile learning app," in *2016 IEEE 16th International Conference on Advanced Learning Technologies (ICALT)* (Austin, TX: IEEE), 90–94.

Planas, E., Daniel, G., Brambilla, M., and Cabot, J. (2021). Towards a model-driven approach for multiexperience AI-based user interfaces. *Softw. Syst. Model.* 20, 997–1009. doi: 10.1007/s10270-021-00904-y

Qasim, I., Anwar, M. W., Azam, F., Tufail, H., Butt, W. H., and Zafar, M. N. (2020). A model-driven mobile HMI framework (MMHF) for industrial control systems. *IEEE Access* 8, 10827–10846. doi: 10.1109/ACCESS.2020.2965259

Ruiz, J., Serral, E., and Snoeck, M. (2019). Evaluating user interface generation approaches: model-based versus model-driven development. *Softw. Syst. Model.* 18, 2753–2776. doi: 10.1007/s10270-018-0698-x

Sabraoui, A., Abouzahra, A., Afdel, K., and Machkour, M. (2019). "MDD approach for mobile applications based on DSL," in *2019 International Conference of Computer Science and Renewable Energies (ICCSRE)* (Agadir: IEEE), 1–6. doi: 10.1109/ICCSRE.2019.8807572

Sabraoui, A., Koutbi, M. E., and Khriss, I. (2012). "GUI code generation for Android applications using an MDA approach," in *2012 IEEE International Conference on Complex Systems (ICCS)* (Agadir: IEEE), 1–6.

Sabraoui, A., Koutbi, M. E., and Khriss, I. (2013). A MDA-based model-driven approach to generate GUI for mobile applications. *Int. Rev. Comput. Softw. J.* 8, 845–852.

Safdar S. A., Iqbal M. Z., and Khan M. U. (2015). "Empirical evaluation of UML modeling tools—a controlled experiment," in *European Conference on Modelling Foundations and Applications* (Cham: Springer), 33–44.

Safdar, S. A., Lu, H., Yue, T., Ali, S., and Nie, K. (2020). A framework for automated multi-stage and multi-step product configuration of cyber-physical systems. *Softw. Syst. Model* 20:8. doi: 10.1007/s10270-020-00803-8

Salihu, I., Ibrahim, R., Ahmed, B. S., Zamli, K. Z., and Usman, A. (2019). AMOGA: a static-dynamic model generation strategy for mobile apps testing. *IEEE Access* 7, 17158–17173. doi: 10.1109/ACCESS.2019.2895504

Selic, B. (2007). "A systematic approach to domain-specific language design using UML," in *10th IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing (ISORC'07)* (Santorini: IEEE), 2–9.

Son, H. S., Kim, W. Y., and Kim, C. (2013). MOF based code generation method for android platform. *Int. J. Softw. Eng. Appl.* 7, 415–426. doi: 10.1016/j.scico.2018.11.002

Taba, S. E. S., Keivanloo, I., Zou, Y., Ng, J., and Ng, T. (2014). "An exploratory study on the relation between user interface complexity and the perceived quality," in *International Conference on Web Engineering* (Cham: Springer), 370–379.

Thüm, T., Kästner, C., Benduhn, F., Meinicke, J., Saake, G., and Leich, T. (2014). FeatureIDE: an extensible framework for feature-oriented software development. *Sci. Comput. Progr.* 79, 70–85. doi: 10.1016/j.scico.2012.06.002

Usman, M., Iqbal, M. Z., and Khan, M. U. (2014). "A model-driven approach to generate mobile applications for multiple platforms," in *2014 21st Asia-Pacific Software Engineering Conference, vol. 1* (Jeju: IEEE), 111–118.

Usman, M., Iqbal, M. Z., and Khan, M. U. (2017). A product-line model-driven engineering approach for generating feature-based mobile applications. *J. Syst. Softw.* 123, 1–32. doi: 10.1016/j.jss.2016.09.049

Usman, M., Iqbal, M. Z., and Khan, M. U. (2020). An automated model?based approach for unit—level performance test generation of mobile applications. *J. Softw.* 32:e2215. doi: 10.1002/smr.2215

Usman, M., Nadeem, A., and Kim, T. (2008). "UJECTOR: a tool for executable code generation from UML models," in *2008 Advanced Software Engineering and Its Applications* (Hainan: IEEE), 165–170.

Vegendla, A., Duc, A. N., Gao, S., and Sindre, G. (2018). A systematic mapping study on requirements engineering in software ecosystems. *J. Informat. Technol. Res.* 11, 49–69. doi: 10.4018/JITR.2018010104

Yang W., Prasad M. R., and Xie T. (2013). "A grey-box approach for automated GUI-model generation of mobile applications," in *International Conference on Fundamental Approaches to Software Engineering* (Berlin; Heidelberg: Springer), 250–265.