Low-Code Flutter Application Development Solution

Atefeh Nirumand
Luxembourg Institute of Science and Technology
Luxembourg
atefeh.nirumand@list.lu

Iván Alfonso
Luxembourg Institute of Science and Technology
Luxembourg
ivan.alfonso@list.lu

Jordi Cabot
Luxembourg Institute of Science and Technology
University of Luxembourg
Luxembourg
jordi.cabot@list.lu

ABSTRACT

Mobile applications have become an integral part of our lives, mostly by improving the efficiency of our daily tasks. Although the Graphical User Interface (GUI) is essential for the ease and efficient use of these applications, developing them well can be a complex task. The manual coding of GUIs can be time-consuming and pose obstacles to the development of essential business features. To address this challenge, we propose a low-code solution to support the development of mobile applications including the GUI. We have extended BESSER, an open-source low-code platform, to provide a sublanguage for modeling GUIs. Aditionally, we have developed a code generator for mobile Flutter applications. The resultant code adheres to Flutter best development practices, ensuring the development of high-quality applications. The complete infrastructure, including the code generator and associated resources, is accessible in an open-source repository.

CCS CONCEPTS

• **Software and its engineering** → *Software creation and management*.

KEYWORDS

Mobile applications, Graphical User Interface (GUI), Low-code platform, Code generator

ACM Reference Format:
Atefeh Nirumand, Iván Alfonso, and Jordi Cabot. 2024. Low-Code Flutter Application Development Solution. In *ACM/IEEE 27th International Conference on Model Driven Engineering Languages and Systems (MODELS Companion '24), September 22–27, 2024, Linz, Austria.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3652620.3688330

1 INTRODUCTION

The ubiquity of mobile applications in our daily lives has made their development increasingly significant within the software industry [12, 28]. As these applications continue to revolutionize various aspects of our lives, ranging from communication and productivity to entertainment and commerce, their development has become a pivotal focus for developers and organizations alike.

One crucial aspect of mobile application development is the Graphical User Interfaces (GUIs) that align with the unique characteristics and capabilities of mobile platforms [28]. However, developing GUIs is a time-consuming task that often hinders developers from dedicating sufficient time to other essential features and methods of the application [4, 28]. To address this challenge, low-code development approaches have emerged, aiming to minimize manual coding and empower both professional programmers and non-technical users in the software development process [9, 10].

Low-code development for GUIs offers several advantages, including automation, interoperability, usability, and flexibility. Automation plays a vital role in streamlining the development process, while interoperability ensures compatibility across diverse systems and technologies. Usability and flexibility contribute to the creation of user-centric interfaces that can be tailored to specific requirements. In the context of mobile app development, low-code development for GUIs offers additional advantages that enhance productivity. These platforms empower developers to rapidly create GUIs without extensive manual coding, expediting the development process, enabling quicker iterations, and reducing time-to-market for applications [8, 22].

Existing studies in the field of mobile application development, with a specific focus on GUI generation, have revealed some limitations (as reported by [22, 28]). The majority of approaches often do not include comprehensive system modelling, which results in the generation of incomplete functional applications. Many Model-Driven development (MDD) solutions fail to address critical considerations related to the content and data layer, such as data source placement and communication. These issues require further investigation to address data handling and specify data sources effectively. In addition, existing MDD-based tools and languages for code generation suffer from limited flexibility and produce low-quality code. Furthermore, there is a lack of tools specifically designed for non-technical developers.

This research aims to address the aforementioned limitations providing a low-code solution that support mobile application development by automating the generation of Flutter [1] code for UI components and facilitating the implementation of CRUD operations (Create, Read, Update, and Delete) on a database. Our contribution is twofold: 1) extending the BESSER Universal Modeling Language (B-UML) [2] with a new sublanguage for modeling GUIs, and 2) developing a code generator for Flutter applications. Our approach combines the benefits of Model-Driven engineering (MDE) and GUI modeling, offering a comprehensive solution for low-code Flutter app development.

[1] https://flutter.dev/

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
MODELS Companion '24, September 22–27, 2024, Linz, Austria
© 2024 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 979-8-4007-0622-6/24/09
https://doi.org/10.1145/3652620.3688330

Our B-UML extension for modeling GUIs is based on the Interaction Flow Modeling Language (IFML) [18], but it is not fully compliant. We prioritize pragmatism by abstracting the necessary concepts for GUI modeling while ensuring easy integration of our approach with other modeling languages. Furthermore, our metamodel extends and references key concepts from the B-UML metamodel, while also allowing for potential coupling with other modeling languages.

Furthermore, our code generation process for Flutter applications incorporates specific features that we have developed. For instance, our generator provides a wide range of widgets for building interactive user interfaces and handles UI events for user interaction. Moreover, the generated code follows Flutter's best practices, ensuring the development of high-quality applications. By integrating our extended B-UML notation and specialized code generator, we present a powerful open-source tool that empowers developers to design and implement Flutter applications with minimal manual coding. This bridging of the gap between modeling and code generation streamlines the software development process, allowing developers to create high-quality Flutter applications efficiently.

The paper is structured as follows: Section 2 presents the basic concepts that form the foundation of our research. Following that, in Section 3, we focus on extending the BESSER platform. We introduce the extended B-UML notation and discuss our code generator. To illustrate the motivation behind our research, Section 4 provides an illustrative example. Section 5 offers an overview of the related work, providing insights into the existing research in the field. Section 6 outlines directions for future work. Finally, in Section 7, we draw conclusions from our research.

2 BASIC CONCEPTS

This section provides background information on Low code development, mobile application development, Flutter, and IFML.

Low-code Development. Low-code development is an innovative software development approach that prioritizes the use of visual development platforms and cloud-based tools to minimize the reliance on manual coding. Unlike traditional development methods, low-code development platforms are designed to accommodate not only professional programmers but also non-technical users or citizen developers, empowering them to actively participate in the software development process [10].

In the realm of low-code development, the BESSER platform [2] has emerged as an open-source solution. With the goal of simplifying software development, BESSER provides a low-code platform that empowers both professional programmers and non-technical users to actively participate in the development process. To define domain models within BESSER, the platform leverages B-UML (BESSER's Universal Modeling Language). In this paper, we consider B-UML to be equivalent to UML class diagrams, offering a familiar and standardized notation for describing domain models. This equivalence ensures clarity and facilitates effective communication among project teams, as stakeholders can utilize their existing knowledge of UML class diagrams when working with B-UML in the BESSER environment.

Mobile Application Development. Mobile application development encompasses the creation of software applications designed specifically for mobile devices, including smartphones and tablets [28]. These applications are customized to leverage the unique capabilities and features of mobile devices, enabling users to seamlessly interact with them. Given the widespread use and popularity of mobile devices, mobile application development has become a critical aspect of the software industry [23]. This process entails designing user interfaces and implementing functionalities that cater to the unique characteristics and capabilities of mobile platforms.

Flutter. Flutter, an open-source SDK developed by Google, offers a high-performance solution for creating cross-platform applications. It adopts a widget-based development approach, where the user interface is constructed using reusable building blocks called widgets. Flutter leverages the Dart [2] programming language, which provides strong typing and supports rapid iterative development through its just-in-time (JIT) compiler. Furthermore, Dart offers the advantage of being compiled into native machine code using ahead-of-time (AOT) compilation, which enhances the performance of production-ready applications by optimizing the execution speed and resource utilization. Developers can leverage the versatility and efficiency of the Flutter framework by writing code once and deploying it across multiple platforms, including mobile, web, and desktop.

IFML. The Interaction Flow Modeling Language (IFML) is a well-established and standardized modeling language in the field of software engineering. It provides a standardized notation and methodology for describing the interaction flows and behavior of user interfaces. Designers and developers can employ IFML to create models that specify the structure and behavior of the user interface, encompassing aspects such as screen navigation, input and output data, and interaction patterns [7].

3 EXTENDING BESSER PLATFORM

This work presents extensions made to the BESSER platform, focusing on two key aspects. Firstly, the B-UML metamodel has been extended to incorporate a GUI modeling language, enhancing the platform's capabilities for modeling GUIs. This extension empowers developers to specify GUI components and their interactions, facilitating comprehensive UI design.

Secondly, the platform now includes a backend generator responsible for generating a database schema within the application. Additionally, a front-end generator for the Flutter application has been implemented, providing a GUI that supports CRUD operations (Create, Read, Update, and Delete) on the database.

Figure 1 provides an overview of the extended BESSER platform architecture. At its core, this platform leverages B-UML (BESSER's Universal Modeling Language) as the fundamental language for defining domain models.

The following subsections provide a comprehensive overview of the B-UML language, with a focus on the GUI modeling part, as well as the code generator.

[2] https://dart.dev/

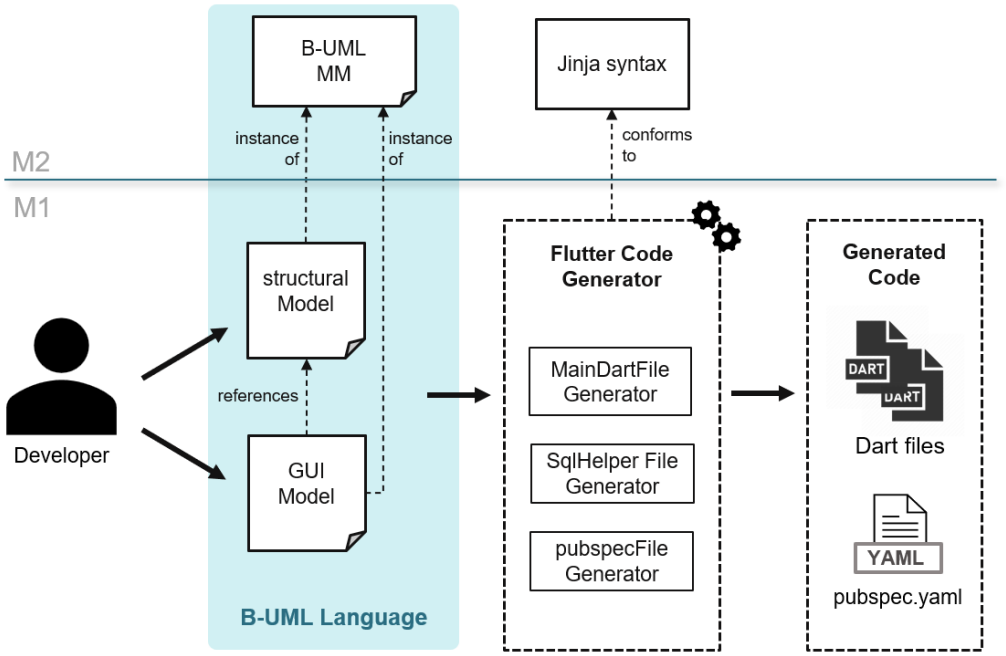


Figure 1: Overview of extended BESSER platform for Flutter application development.

3.1 Extending B-UML for GUI specification

To effectively utilize the code generator capabilities of the BESSER platform, two models are required from the user: (1) a structural model that contains information about the application itself, including classes, attributes, and relationships defining the underlying data structure, and (2) the GUI model for specifying various elements such as lists, buttons, screens, and view components that constitute the GUI.

The Structural Metamodel submodule, available in [19], provides a framework for specifying domain models using fundamental concepts found in a class diagram. It enables the definition of the static structure of a system, such as a mobile application. While the specifics of the structural model are not discussed in this paper due to space limitations, further details can be found in our previous work [2]. Therefore, the remainder of this section focuses on the B-UML extension for GUI models.

For the definition of the structure and behavior of GUIs, we have extended the B-UML metamodel, influenced by IFML specification. By leveraging the concepts from IFML, the BESSER platform ensures compatibility and interoperability with existing UI modeling tools and frameworks. It is important to note that our extension does not aim to cover the entire scope of IFML. Instead, we have selectively adopted relevant concepts to enhance the modeling of GUIs within the BESSER platform. IFML has gained significant recognition and acceptance within the field of UI modeling, making it a valuable standard that influences our extension.

Figure 2 illustrate the B-UML metamodel extension for GUI specification. At the top level, the *Application* metaclass encapsulates properties such as the application's name, package, modules, description, and screen compatibility (a boolean parameter indicating whether the application is adaptable to various types of devices with different screen sizes and resolutions). *Application* serves as the central entity in the metamodel, providing a means to define and organize GUI components. An *Application* can have one or multiple *Module(s)*, representing distinct functional units or components within the GUI application.

Within each *Module*, there can be one or multiple *Screen(s)*, which represent specific screens or interfaces within the GUI application. They define the visual layout and user interaction for a particular part of the application. *Screen(s)* can contain various *ViewElement(s)*. To structure the layout of a *Screen*, the *ViewContainer* element is utilized. *ViewContainer* acts as a container or layout component that holds other view elements or components, providing a way to organize and arrange the placement of view elements within the screen's layout.

The *ViewElement* represents the visual elements of the user interface, encompassing a wide range of components such as images, buttons, data lists, input fields, forms, and menus. It defines the individual components that make up the UI and is organized within the *ViewContainer*.

The concepts proposed in the metamodel are deliberately designed to transcend traditional developer-oriented paradigms, focusing instead on significant and common concepts that are fundamental to user interfaces, particularly within the context of mobile applications. These concepts encompass common elements and interactions that are intuitive not only to experienced developers but also to users who lack specific expertise in mobile GUI development.

By establishing a connection between the technical implementation and the user-centric design, our approach provides non-developers, frequently designated as citizen developers, with the

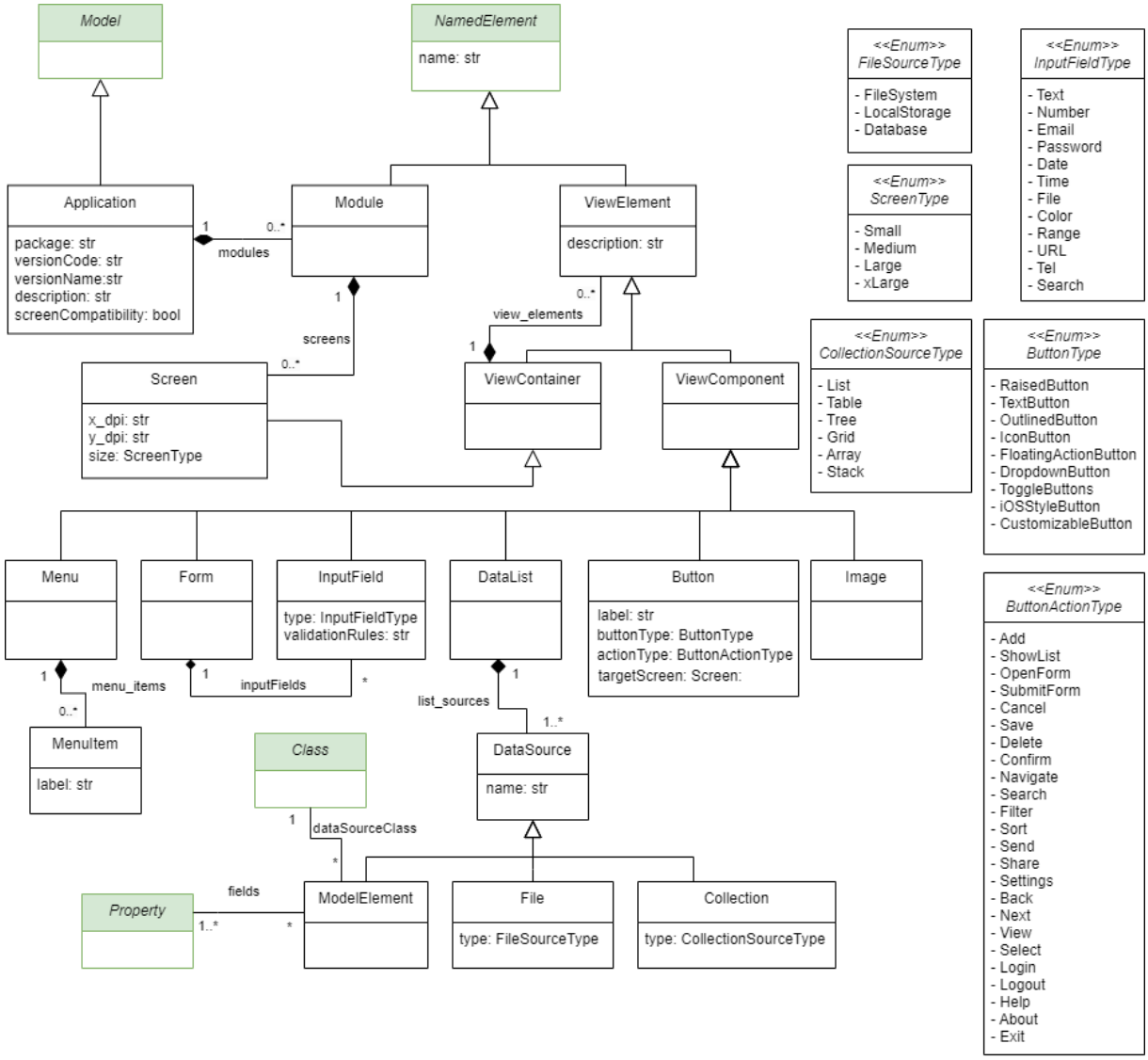


Figure 2: Graphical User Interface Metamodel (the green metaclasses representing the structural metamodel of B-UML).

ability to engage with the design of applications in a more effective manner. The metamodel enables a more intuitive mapping to widgets and UI event handling in the generated code. This alignment ensures that users with diverse technical backgrounds can comprehend and employ these concepts, thus enhancing the accessibility and usability of the GUI design process. This approach offers a distinctive advantage in that it simplifies complex development tasks and enables a broader range of users to contribute to application development in a meaningful manner.

By integrating the structural and GUI submodules of B-UML, the BESSER platform provides a modeling language to define the static structure of a system using the Structural Metamodel, while enabling the specification of interactive graphical user interfaces through the GUI Metamodel. By leveraging the combined power of these submodules, the BESSER platform empowers developers to streamline the development process and efficiently create GUIs for their applications.

3.2 Code generation

Consistent with the other BESSER code generators, our Flutter application generator uses Jinja templates [3] to produce different software artifacts (executable code). Specifically, three Model-to-Text (M2T) transformations are executed from the structural and GUI models (see Figure 1):

- **MainDartFile Generator**: this generator produces Dart code serving as the entry point for the Flutter application.

[3] https://palletsprojects.com/p/jinja/

Table 1: Widget Support and UI Event Handling in the Code Generator.

Feature (widget(s))	UIEvents
MaterialApp, Scaffold, AppBar, Text, Center, Column, ElevatedButton, SizedBox, Navigator, MaterialPageRoute, TextField, DropdownButtonFormField, DropDownMultiSelect, FloatingActionButton, CircularProgressIndicator, ListView, GestureDetector, ListTile, IconButton	onPressed (for buttons), onChanged (for text fields), value changed event handling, showModalBottomSheet (showing a modal bottom sheet), setState (triggering state changes), Snackbar handling, Future (for async operations), initState (initialization logic), Routing using Navigator

Table 2: List of Supported Best Practices in the Generated Code [11].

Best Practice	Explanation
Enhancing Code Modularity and Reusability	Keep widgets compact and reusable to promote code modularity and minimize duplication.
Efficient State Management with Widget Tree	Utilize the widget tree to manage the app's state effectively, ensuring organized and comprehensible code structure.
Code Readability and Consistency	Utilize the Dart analyzer command line tool to enhance code readability and maintain consistency.
Optimizing Efficiency and User Experience	Eliminate extraneous loops, use constants over literals, and minimize object creation to enhance application efficiency.
Implementing Material Design Guidelines	Adhere to Material Design principles in Flutter for a consistent and user-friendly interface design.
Simplifying and Streamlining UI Design	Keep the UI simple and clean by focusing on clear layout, focused actions, and minimalist styling.
Thoughtful Layout and Spacing	Pay attention to layout and spacing considerations for a well-structured and visually appealing interface.

The resulting *main.dart* file contains the actual code responsible for configuring the application and structuring its components. This file is essential in setting up the initial structure of the application, defining the root widget, and managing primary navigation logic. Moreover, it incorporates necessary dependencies and imports to leverage Flutter's UI components and other functionalities. By registering required routes and initializing essential services, the *main.dart* file plays a pivotal role in ensuring the application's functionality. The code incorporates the implementation of widget display, business logic, and the management of various screens and user interactions, thereby providing a robust foundation for both behaviour and data-driven operations. As a result, the *main.dart* file is a critical element that directly contributes to the overall functionality and usability of the application.

- **SqlHelperFile Generator**: it holds the responsibility of generating the *sql_helper.dart* file, a vital component containing essential code for managing an SQLite database within the backend of a Flutter application. This file encapsulates a set of functions that facilitate various operations, including table creation, data retrieval, and data manipulation. The functions embedded in *sql_helper.dart* support the CRUD operations for the entities defined in the structural model. Therefore, developers gain the ability to efficiently manage entities and their associations within the SQLite database. As a result, the *sql_helper.dart* file assumes a pivotal role in the backend architecture of the application, enabling seamless data management and manipulation. Its significance lies in its capability to ensure the smooth functioning of the application's backend, thereby contributing substantially to the overall effectiveness and efficiency of data-related operations.
- **pubspecFile Generator**: this component generates a YAML file, known as *pubspec.yaml*, within a Flutter application. By simplifying the process of dependency management, the generator streamlines the development workflow. The *pubspec.yaml* file contains valuable information such as project-specific metadata, dependencies, and their corresponding versions.

Our code generator provides the foundational elements of a functional mobile application, including the specified user interface, while allowing for further customization and enhancement by developers. This baseline code includes essential CRUD operations, and developers can extend or modify it to incorporate additional business logic or complex data interactions. Our approach ensures that while the generated code covers core functionalities, developers have the flexibility to adapt and expand it to meet diverse application scenarios.

The approach employed in this generator encompasses three important features that greatly enhance its functionality and usefulness:

- **Handling Cardinalities**. The generator handles various cardinalities present in the domain model. It seamlessly supports cardinality types such as one-to-many, many-to-one, one-to-one, and zero-to-many relationships. This ensures that the generated output accurately reflects the intended cardinalities specified in the domain model, resulting in reliable and functional code.
- **Widget Support and UI Event Handling**. The generator provides support for a diverse range of widgets used in building interactive user interfaces. Table 1 showcases the versatility of the generator in handling widgets. One notable example is the *ElevatedButton* widget that triggers actions or operations in the mobile application. The *onPressed* event associated with this widget allows developers to define the desired functionality when the button is pressed. By leveraging the generator's capabilities, developers can easily incorporate the *ElevatedButton* widget and customize the *onPressed* event to efficiently manage user interactions, enhancing the overall development process and user experience.
- **Adherence to Flutter Best Practices**. The generated code adheres to established Flutter best practices, ensuring the development of high-quality applications. A list of these best practices is provided in Table 2. To gain a more comprehensive understanding of these practices, it is recommended to consult the official Flutter documentation site [11].

Listing 1: Example Jinja Template for Generating Elevated Buttons for Screen Navigation.

```
1  {%- for component in screen.view_elements %}
2     {%- if component is is_Button %}
3          {%- if component.buttonType.value ==
               'Raised Button' and
               component.actionType.value ==
               'Navigate'%}
4            ElevatedButton(
5              onPressed: () {
6                Navigator.push(
7                 context,
8                  MaterialPageRoute(
9                   builder: (context) =>
10                    const {{component
11                      .targetScreen.name}}(),
12                 ),
13                ).then((value) {
14                  setState(() {});
15                });
16             },
17             child: const
                   Text('{{component.label}}'),
18           ),
19           const SizedBox(height: 16),
20         {%- endif %}
21     {%- endif %}
22 {%- endfor %}
```

Listing 1 provides a code snippet that exemplifies the usage of a Jinja template to generate Dart code responsible for creating elevated buttons within a user interface. The template employs a loop to iterate through the view elements of a screen (line 1). Within this loop, it checks if the element corresponds to a button and satisfies specific criteria (lines 2-3), such as being a raised button with a navigation action. Once these conditions are met, the template proceeds to generate code for constructing an *ElevatedButton* widget with the relevant properties (lines 4-19).

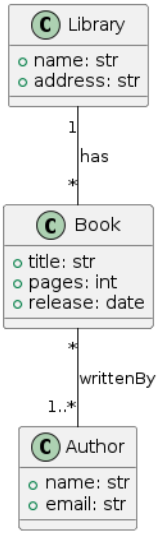


Figure 3: Libby Domain Model.

The *onPressed* function of the generated button utilizes the capabilities of the *Flutter Navigator* class to facilitate navigation to a designated screen specified by the component's *targetScreen* value. Upon completion of the navigation, a callback function is invoked to update the state of the current screen using *setState*. These functions and DART code required for navigation buttons is automatically generated by the code generator.

4 RUNNING EXAMPLE

To demonstrate our proposed approach, we present an illustrative case study inspired by the Libby app [4], developed by OverDrive [5]. The Libby app stands as a leading digital library platform, revolutionizing users' access to and enjoyment of books from local public libraries.

Figure 3 showcases a reduced model created using PlantUML [6], highlighting key components within the domain, such as the Book, Library, and Author entities. By harnessing the capabilities of the BESSER platform, we utilize the Flutter code generator to automatically generate the executable code required for a customized Flutter application. This application is specifically designed to efficiently handle book management within public libraries.

To utilize the capabilities of the code generator effectively, application designers are required to provide the structural model and GUI model in accordance with the B-UML language. In BESSER, the structural model can be created using three distinct concrete

[4] https://meet.libbyapp.com/
[5] https://company.overdrive.com/meet-libby/
[6] https://plantuml.com/

syntaxes: 1) using a grammar to define models following the PlantUML syntax; 2) providing an image (e.g., a picture of a hand-drawn model on a board) which is converted into a structural model; and 3) leveraging the B-UML library to craft models using Python. The structural model encapsulates the classes, attributes, and relationships that define the underlying data structure. Similarly, the GUI model is defined using Python code, allowing for the specification of various elements such as lists, buttons, screens, and view components that constitute the GUI.

By leveraging the power of BESSER and the Flutter code generator, designers can seamlessly transform their structural and GUI models into executable code, significantly reducing development effort and accelerating the creation of a customized Flutter application for managing books from local public libraries.

Listing 2: The GUI Model (Illustrating the Libby Example).

```
# Application definition:
libby_app = Application(
    name="Library Management",
    modules={
        Module(
            name="module_name",
            screens={
                Screen(
                    name="Book Library Manager",
                    view_elements={
                        Button(
                            name="Library List Button",
                            buttonType=ButtonType.RaisedButton,
                            actionType=ButtonActionType.Navigate,
                            targetScreen=LibraryListScreen
                            # Other info omitted for brevity purposes
                        ),
                        Button(
                            name="Author List Button",
                            buttonType=ButtonType.RaisedButton,
                            actionType=ButtonActionType.Navigate,
                            targetScreen=AuthorListScreen
                            # Other info omitted for brevity purposes
                        ),
                        Button(
                            name="Book List Button",
                            buttonType=ButtonType.RaisedButton,
                            actionType=ButtonActionType.Navigate,
                            targetScreen=BookListScreen
                            # Other info omitted for brevity purposes
                        )
                    }
                    # Other info omitted for brevity purposes
                ),
                LibraryListScreen,
                AuthorListScreen,
                BookListScreen
            }
            # Other info omitted for brevity purposes
        )
    }
    # Other info omitted for brevity purposes
)
```

Listing 2 shows an excerpt of the GUI model for the considered example, written using our defined concrete syntax. In this example, the *libby_app* application is defined (lines 34-38), and it consists of a main page named *MyHomeScreen* composed of several UI elements, such as the *libraryButton*, *authorButton*, and *bookButton*. Each button is defined with a name, button type, action type, and a target screen to navigate to when pressed. The *libraryButton* performs a navigation action to the *LibraryListScreen*, while the *authorButton* and *bookButton* navigate to the *AuthorListScreen* and *BookListScreen*, respectively. The *MyHomeScreen* serves as the home page screen, titled "Book Library Manager" and incorporates the aforementioned buttons as its view elements. Additionally, the GUI model encompasses a module named *MyModule* with the name "module_name," comprising multiple screens such as *MyHomeScreen*, *LibraryListScreen*, *AuthorListScreen*, and *BookListScreen*. Lastly, the *libby_app* application is defined as "Library Management" and includes *MyModule* as its module.

This snippet showcases the utilization of our defined concrete syntax to specify the GUI model, enabling a clear representation of the application's structure, screens, and navigation elements.

The generated code demonstrates not only the fundamental navigation patterns and widget rendering but also handles complex scenarios involving data binding, user interactions, and dynamic responses.

As illustrated in Table 1, our system is designed to accommodate a comprehensive range of *UIEvents*. By integrating these UIEvents within the generated codebase, developers with varying levels of experience could interact with and construct the application's logic. This approach aligns with the fundamental principles of a low-code environment, enabling the development of robust and interactive user interfaces.

Figure 4 presents the user interface of the "Book Directory" page within the Flutter application, which is constructed using the generated code. This intuitive interface empowers users to efficiently manage book information, aligning with the Digital library domain model. To delve deeper into this example and gain a comprehensive understanding, the complete source code and a detailed guide can be accessed in the BESSER repository [19].

5 RELATED WORK

The generation of GUI code for mobile applications is an evolving research area within mobile application development [17, 28]. Various approaches have been proposed, falling into categories such as modeling, static analysis, dynamic analysis, and other innovative techniques.

Model-based approaches utilize UML diagrams and UML modeling profiles like IFML [18] and LIZARD [15] to generate native GUIs for mobile apps. For example, Bernaschina et al. [5] proposed the use of UML sequence and class diagrams to design the GUI, emphasizing the consideration of the complete lifecycle. Other approaches leverage static and dynamic analysis techniques [20], such as the technique proposed by Yang et al. [27], which introduced a method for statically extracting supported events from the GUI of an application using static analysis. They further validated the model's events through dynamic crawling. Fernandez et al. [3] present Mobia modeler, a tool that simplifies the creation process of mobile applications for non-technical users, focusing on empowering users without programming skills to design and develop mobile apps. Acerbis et al. [1] propose an approach based on IFML and the WebRatio [7] platform for web and mobile application development, illustrating how IFML can be used to create models and generate code for web and mobile applications. Zafar et al. [28] presented MOBICAT, an innovative MDE approach that automates GUI code generation and controller class creation for Android applications.

[7] https://www.webratio.com/site/content/en/home

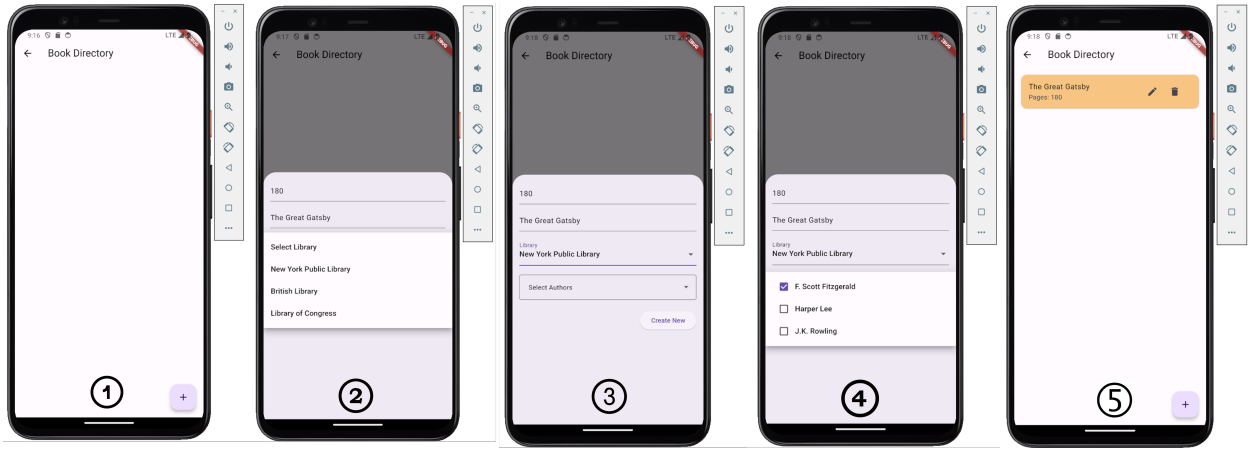


Figure 4: User Interface of the *Book Directory* Page.

This approach involves modeling domain-specific features of mobile applications and capturing requirements using UML diagrams, ultimately leading to seamless automated GUI generation.

Model-driven development (MDD) approaches in mobile application development aim to accelerate delivery, deployment, and time to market [22, 28]. These approaches can be categorized as UML-based, feature-based, or utilizing other innovative notations and guidelines. Ruokonen et al. [21] focus on MDD techniques for mobile business processes, providing insights into the application of MDD in this context. They discuss the benefits and challenges of using models to develop mobile business processes. Jaber et al. [13] introduce a high-level modeling language specifically designed for efficient design, implementation, and testing of Android applications, leveraging models to improve the development process and highlighting the benefits of using this language. Vaupel et al. [26] propose an MDD approach for Android and iOS applications. Their approach supports role-based app variability by introducing a modeling language and infrastructure that enable flexible app development. It allows for the modeling of individual elements to accommodate specific behaviors based on roles. UML-based approaches employ a subset of UML and UML profiles, such as PELLET [24], to capture GUI concepts. Additionally, other approaches incorporate domain-specific languages (DSLs), as demonstrated by the work of Ko et al. [14].

The concept of user interface specification and automatic generation has been the subject of ongoing research, as evidenced by the work of Vanderdonckt and Bodart [6, 25]. Their work highlights the significance of maintaining consistency and usability across diverse platforms, underscoring the pivotal role of specification models in attaining these objectives. Similarly, Just-UI [16] proposes a user interface specification model that integrates user requirements into the development process, guaranteeing that generated interfaces align with user expectations and needs. Such specification models offer a structured approach to UI generation, which can be applied to mobile app development to ensure consistency and usability.

The existing studies conducted in the field of mobile application development, specifically focusing on GUI generation, have highlighted several limitations [22, 28]. Only a small fraction of studies have successfully modeled the entire system, resulting in a fully functional app, highlighting the lack of comprehensive system modeling in most approaches. Moreover, the code generators employed in these studies often focus primarily on generating the database and higher-level code, neglecting the complete app generation process. Many MDD-based solutions place significant emphasis on UI/UX and business logic while overlooking critical considerations related to the content and data layer. Decisions regarding data source placement, communication between the app and data source, and handling these issues, including specifying the source of data, are frequently overlooked and warrant further investigation. Additionally, the majority of proposed MDD-based tools/languages for code generation suffer from limited flexibility and produce poor quality code. Furthermore, there is a lack of tools specifically designed to support non-technical developers.

To address these limitations, our proposal focuses on extending the BESSER low-code platform to support GUI development for mobile applications. Our tool encompasses a comprehensive code generator that facilitates the creation of user-friendly interfaces for CRUD operations. Specifically, it generates the necessary Flutter code for managing an SQLite database within the backend of a Flutter application, as well as a front-end GUI for performing CRUD operations. Additionally, our low-code solution handles crucial aspects such as data source placement, communication between the app and data source, and specifying the source of data. Our code generator adheres to best practices in Flutter development, ensuring the production of high-quality code. Moreover, our tool caters to non-technical developers, empowering them to utilize it for generating code for workable applications.

6 FUTURE WORK

In this section, we outline potential avenues for future work to enhance our research on generating low-code Flutter app development.

Automating the Generation of UI Models. One promising avenue for future work is to automate the generation of UI models from mock-ups, eliminating the need for designers to invest significant time and effort in manual modeling tasks. Despite the

availability of user-friendly modeling editors, designers may be hesitant to manually create UI models due to the associated complexities and overhead. To address this, our code generator can be enhanced to seamlessly translate mock-ups into GUI models, leveraging the principles of MDE. By extracting relevant information from mock-ups, the code generator can automatically generate corresponding GUI models, significantly reducing the manual effort required from designers. This automation empowers designers to focus more on high-level design decisions and the creative aspects of the mobile application.

Furthermore, we envision going beyond a one-shot transformation from mock-ups to GUI models. We propose integrating a chatbot within the development process to enable designers to interact with the UI in a more dynamic and iterative manner. The chatbot serves as a conversational interface, facilitating designer requests for style changes, modifications to the page structure, and other UI adjustments. This chatbot-driven interaction fosters continuous collaboration between designers and the UI, resulting in more flexible and adaptable mobile applications.

Simplifying the Process of Creating CRUD-like GUI models (for Flutter Apps). Another important area for future work is to simplify the process of creating CRUD-like GUI models by automating the transformation from structural models to GUI models, specifically targeting Flutter applications that support CRUD operations on data. To achieve this, our approach can be enhanced by leveraging MDE techniques, with a particular focus on B-UML. By automatically generating CRUD-like GUI models tailored to Flutter applications, we can reduce the manual effort required from designers in creating GUI models, allowing them to allocate their time and energy towards other critical aspects of application development.

Additionally, our research vision includes the integration of a chatbot as a means for designers to interact with the generated CRUD-like GUI models. This chatbot facilitates dynamic changes and adjustments to the database operations, empowering designers to request modifications, add new buttons, or define custom functionalities. By incorporating this interactive communication channel, our approach empowers designers with greater control and flexibility over the database operations. Designers can easily fine-tune the generated CRUD-like GUI models to align with their specific application requirements, resulting in more efficient and tailored mobile applications.

Deriving GUI Models from Existing Implementations for Modernization Scenarios. An additional area of future work involves leveraging BESSER in modernization scenarios. This entails deriving GUI models from an existing GUI implementation and subsequently transitioning the application to a Flutter implementation. By employing reverse engineering techniques and BESSER's capabilities, the essential components, structure, and behavior of the original GUI implementation can be analyzed and extracted. This information serves as the basis for automatically generating GUI models that faithfully represent the user interface of the existing application. The derived GUI models then facilitate a smoother migration process to Flutter. Designers can utilize these generated models as a solid foundation for developing the user interface within the Flutter framework.

Expanding the Coverage of GUI Features and UI Events. To further enhance the capabilities of the code generator, a potential area for future work involves expanding the coverage of GUI features and user interaction events (UI Events). This expansion aims to capture a wider range of GUI elements and interactions. To accomplish this, future research can explore the incorporation of additional modeling techniques or analyze real-world mobile applications to identify and address any existing limitations in GUI representation and interaction.

Extending Support for Other Mobile App Development Platforms. Another area for future work is the extension of support for other mobile app development platforms beyond the current focus on Flutter. This extension could involve researching and integrating other popular mobile app development platforms, such as React Native [8], into the code generator. Accomplishing this would require understanding the specific requirements, design patterns, and conventions of these frameworks, and adapting the code generation process accordingly. By extending support to other platforms, the code generator can cater to a broader audience of developers working on different mobile app development frameworks, expanding its applicability and reach.

Conducting Evaluations and Validation Studies. An essential area for future work is conducting extensive evaluations and validation studies of the proposed approach in real-world scenarios. These studies will provide valuable insights into the approach's effectiveness, usability, and performance. Furthermore, conducting comparative evaluations against existing manual modeling approaches and alternative automated solutions can assist in identifying the benefits and limitations of our approach. This will also clarify its unique contributions to the field, allowing us to refine the approach to better meet the needs of developers and further establish its advantages over traditional methods.

7 CONCLUSION

This research presents an innovative approach to low-code Flutter application development, starting with the extension of the BESSER platform using the BESSER Universal Modeling Language (B-UML). The extended B-UML metamodel incorporates a new sublanguage specifically designed for modeling Graphical User Interfaces (GUIs). Furthermore, a code generator is developed to automate the generation of Flutter code for UI components and streamline CRUD operations on databases. This code generator leverages the principles of MDE and GUI modeling to translate the designed B-UML models into functional Flutter code. The generated code adheres to Flutter best practices, ensuring the development of high-quality applications.

Overall, this research contributes to the advancement of low-code development in the Flutter ecosystem by providing an open-source solution that combines the strengths of B-UML and a specialized code generator. By bridging the gap between modeling and code generation, this approach streamlines the software development process and empowers developers to create Flutter applications with minimal manual coding.

[8] https://reactnative.dev/

8 ACKNOWLEDGMENTS

This project is supported by the Luxembourg National Research Fund (FNR) PEARL program, grant agreement 16544475.

REFERENCES

[1] Roberto Acerbis, Aldo Bongio, Marco Brambilla, and Stefano Butti. 2015. Model-driven development based on omg's IFML with webratio web and mobile platform. In *Engineering the Web in the Big Data Era: 15th International Conference, ICWE 2015, Rotterdam, The Netherlands, June 23-26, 2015, Proceedings 15*. Springer, Springer, 605–608. https://doi.org/10.1007/978-3-319-19890-3_39

[2] Iván Alfonso, Aaron Conrardy, Armen Sulejmani, Atefeh Nirumand, Fitash Ul Haq, Marcos Gomez-Vazquez, Jean-Sébastien Sottet, and Jordi Cabot. 2024. Building BESSER: an open-source low-code platform. In *International Conference on Business Process Modeling, Development and Support*. Springer, 203–212. https://doi.org/10.1007/978-3-031-61007-3_16

[3] Florence Balagtas-Fernandez, Max Tafelmayer, and Heinrich Hussmann. 2010. Mobia modeler: Easing the creation process of mobile applications for non-technical users. In *Proceedings of the 15th international conference on Intelligent user interfaces*. ACM, 269–272. https://doi.org/10.1145/1719970.1720008

[4] Tony Beltramelli. 2018. pix2code: Generating code from a graphical user interface screenshot. In *Proceedings of the ACM SIGCHI Symposium on Engineering Interactive Computing Systems*. ACM, 1–6. https://doi.org/10.1145/3220134.3220135

[5] Carlo Bernaschina, Sara Comai, and Piero Fraternali. 2018. Formal semantics of OMG's Interaction Flow Modeling Language (IFML) for mobile and rich-client application model driven development. *Journal of Systems and Software* 137 (2018), 239–260. https://doi.org/10.1016/j.jss.2017.11.067

[6] François Bodart and Jean Vanderdonckt. 1994. On the Problem of Selecting Interaction Objects. In *BCS HCI*. 163–178.

[7] Marco Brambilla and Piero Fraternali. 2014. *Interaction flow modeling language: Model-driven UI engineering of web and mobile apps with IFML*. Morgan Kaufmann.

[8] Alessio Bucaioni, Antonio Cicchetti, and Federico Ciccozzi. 2022. Modelling in low-code development: a multi-vocal systematic review. *Software and Systems Modeling* 21, 5 (2022), 1959–1981. https://doi.org/10.1007/s10270-021-00964-0

[9] Jordi Cabot. 2020. Positioning of the low-code movement within the field of model-driven engineering. In *Proceedings of the 23rd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings*. ACM, 1–3. https://doi.org/10.1145/3417990.3420210

[10] Davide Di Ruscio, Dimitris Kolovos, Juan de Lara, Alfonso Pierantonio, Massimo Tisi, and Manuel Wimmer. 2022. Low-code development and model-driven engineering: Two sides of the same coin? *Software and Systems Modeling* 21, 2 (2022), 437–446. https://doi.org/10.1007/s10270-022-01038-5

[11] Flutter. 2024. Performance best practices. Retrieved June 11, 2024 from https://docs.flutter.dev/perf/best-practices

[12] Statista Market Forecast. 2024. App – Worldwide. Retrieved June 11, 2024 from https://www.statista.com/outlook/amo/app/worldwide#revenue

[13] Mohamad Jaber, Yliès Falcone, Kinan Dak-Al-Bab, John Abou-Jaoudeh, and Mostafa El-Katerji. 2018. A high-level modeling language for the efficient design, implementation, and testing of Android applications. *International Journal on Software Tools for Technology Transfer* 20 (2018), 1–18. https://doi.org/10.1007/s10009-016-0441-2

[14] Minhyuk Ko, Yong-Jin Seo, Bup-Ki Min, Seunghak Kuk, and Hyeon Soo Kim. 2012. Extending UML meta-model for android application. In *2012 IEEE/ACIS 11th International Conference on Computer and Information Science*. IEEE, IEEE, 669–674. https://doi.org/10.1109/ICIS.2012.48

[15] Ignacio Marin, Francisco Ortin, German Pedrosa, and Javier Rodriguez. 2015. Generating native user interfaces for multiple devices by means of model transformation. *Frontiers of Information Technology & Electronic Engineering* 16, 12 (2015), 995–1017. https://doi.org/10.1631/FITEE.1500083

[16] Pedro J Molina, Santiago Meliá, and Oscar Pastor. 2002. Just-ui: A user interface specification model. In *Computer-Aided Design of User Interfaces III: Proceedings of the Fourth International Conference on Computer-Aided Design of User Interfaces*. Springer Science & Business Media, 63–74. https://doi.org/10.1007/978-94-010-0421-3_5

[17] Manuel Núñez, Daniel Bonhaure, Magalí González, and Luca Cernuzzi. 2020. A model-driven approach for the development of native mobile applications focusing on the data layer. *Journal of Systems and Software* 161 (2020), 110489. https://doi.org/10.1016/j.jss.2019.110489

[18] OMG. 2024. O.M.G.: About interaction flow modeling language specification. Retrieved June 11, 2024 from https://www.omg.org/spec/IFML

[19] Github repository. 2024. BESSER. Retrieved June 11, 2024 from https://github.com/BESSER-PEARL/BESSER

[20] Jenny Ruiz, Estefanía Serral, and Monique Snoeck. 2019. Evaluating user interface generation approaches: model-based versus model-driven development. *Software & Systems Modeling* 18, 4 (2019), 2753–2776. https://doi.org/10.1007/s10270-018-0698-x

[21] Anna Ruokonen, Lasse Pajunen, and Tarja Systa. 2008. On model-driven development of mobile business processes. In *2008 Sixth International Conference on Software Engineering Research, Management and Applications*. IEEE, IEEE, 59–66. https://doi.org/10.1109/SERA.2008.30

[22] Md Shamsujjoha, John Grundy, Li Li, Hourieh Khalajzadeh, and Qinghua Lu. 2021. Developing mobile applications via model driven development: A systematic literature review. *Information and Software Technology* 140 (2021), 106693. https://doi.org/10.1016/j.infsof.2021.106693

[23] Statista. 2024. Number of mobile devices worldwide 2020-2025. Retrieved June 11, 2024 from https://www.statista.com/statistics/245501/multiple-mobile-device-ownership-worldwide/

[24] Muhammad Usman, Muhammad Zohaib Iqbal, and Muhammad Uzair Khan. 2020. An automated model-based approach for unit-level performance test generation of mobile applications. *Journal of Software: Evolution and Process* 32, 1 (2020), e2215. https://doi.org/10.1002/smr.2215

[25] Jean M Vanderdonckt and François Bodart. 1993. Encapsulating knowledge for intelligent automatic interaction objects selection. In *Proceedings of the INTERACT'93 and CHI'93 Conference on Human Factors in Computing Systems*. ACM, 424–429. https://doi.org/10.1145/169059.16934

[26] Steffen Vaupel, Gabriele Taentzer, René Gerlach, and Michael Guckert. 2018. Model-driven development of mobile applications for Android and iOS supporting role-based app variability. *Software & Systems Modeling* 17 (2018), 35–63. https://doi.org/10.1007/s10270-016-0559-4

[27] Wei Yang, Mukul R Prasad, and Tao Xie. 2013. A grey-box approach for automated GUI-model generation of mobile applications. In *International Conference on Fundamental Approaches to Software Engineering*. Springer, Springer, 250–265. https://doi.org/10.1007/978-3-642-37057-1_19

[28] Haroon Zafar, Saif Ur Rehman Khan, Atif Mashkoor, and Habib Un Nisa. 2024. MOBICAT: a model-driven engineering approach for automatic GUI code generation for Android applications. *Frontiers in Computer Science* 6 (2024), 1397805. https://doi.org/10.3389/fcomp.2024.1397805