Journal of Computer and Knowledge Engineering, Vol. , No.. 2023.
DOI:

Model-Driven Development of Android Audio-based Applications*

[1], x[2], y[3]

***Abstract*-- This paper presents a model-driven engineering framework designed to enhance the development of flexible, high-quality audio-based applications on mobile platforms. The framework comprises domain-specific metamodels, a graphical editor, and a transformation engine, enabling the automatic generation of application code and supporting customization within Android Studio. To address the challenges faced by developers in delivering effective audio applications, the framework provides a structured approach to simplify design and implementation processes. The framework's applicability is demonstrated through four case studies, highlighting its ability to create diverse audio-based Android applications. A detailed evaluation includes a comparison of development effort between the proposed model-driven approach and traditional coding methods, showing significant reductions in time and manual effort. Additionally, the framework is assessed using key software quality metrics such as maintainability, understandability, and extensibility. The findings demonstrate that the model-driven approach not only streamlines development but also improves the maintenance of applications, enabling developers to meet the growing demand for audio applications efficiently. By reducing development costs and enhancing productivity, this research contributes to the field of software engineering, offering a practical and adaptable methodology for audio-based application development.**

***Index Terms*—Model-Driven Development, Android Application, Audio-based Application, Modeling Language**

I. INTRODUCTION

THE development of modern software, particularly for complex systems, has driven the creation of new techniques like model-driven engineering (MDE) [1]. MDE uses high-level models to facilitate the development process, thus minimizing the time and cost of production [2]. More recently, the growing prevalence of smartphones and portable devices in the last few decades has resulted in an increasing demand for a wide range of applications [3], spanning even into audio-based applications such as audiobooks and language-learning applications. The nature of this rapid growth of mobile users has made it even more obvious that we need development frameworks that speed this process of delivery to market.

An audio-based Android application is designed to deliver content using recorded audio or text-to-speech (TTS) technology. Depending on their purpose, these apps may come with a wide range of features. Audio playback is a key feature (play, pause, and skip functions for audiobooks, podcasts, and music). TTS will handle everything from text-to-speech (TTS) options that allow the app to speak back to you to setting potential language options and even speech speed. Additional features may include an audio file management system, offline playback, and playlist support. Some audio-based apps also offer customization options and support for multiple languages, enhancing the user experience.

The explosion of smartphone usage, combined with the development of audio-based applications, has made the development of such applications difficult [4]. Flexibility, speed, and precision are all common challenges in software development, but the nature of audio-based applications may complicate matters with issues like real-time audio processing, multimedia integration, and even device compatibility. It proposes a framework to address these shortcomings by implementing a systematic, domain-centric strategy that streamlines development, improves maintainability, and accommodates a wide range of application requirements. The evaluation provides metrics such as understandability, extensibility, and maintainability to demonstrate the framework's effectiveness in addressing these challenges.

One of the most challenging aspects of audio-based application development is high flexibility. The user's needs are constantly changing, but the application must be able to accommodate them, and developers must take advantage of this flexibility to apply it when necessary. Furthermore, applications must be designed so that new features can be added without requiring extensive changes of existing code. Another major issue in this space is the need to streamline the personal maintenance and updating procedures for audio-based apps. As applications grow in size and complexity, making changes becomes more expensive and time-consuming. Such a problem is especially difficult for systems that need to be updated regularly. Traditional methods, with their inherent complexities, often fall short in addressing these issues.

Research has demonstrated that MDE, by employing high-level abstractions and automated code generation, allows developers to manage the complexities of software systems more effectively [5]. In the context of audio-based applications, this approach can significantly reduce development time, improve product quality, and enhance maintainability.

This paper proposes a model-driven engineering framework

* Manuscript received Revised, , accepted .

[1] Master Student, b

[2] Corresponding author. Assistant Professor, b

[3] Assistant Professor, b

Uncorrected proof

to address the challenges in creating adaptable audio-based applications at scale. More specifically, the research points towards adapting MDE techniques to improve the development, maintenance, and updating processes of android, audio-based applications. This study tackles the problem of how to apply model-driven engineering to enhance the process of developing, maintaining, and updating mobile audio-based applications. In other words, we combine domain-specific metamodels and a transformation engine to automate the code generation of audio-oriented applications. We intend to contribute to the filling of this significant gap in the literature, especially with regard to the development of flexible yet superior audio applications for android platforms.

The paper is organized as follows. Section II reviews the related work on model-driven engineering (MDE) and its application in Android application development. Section III presents the proposed framework for developing Android audio-based applications, detailing its components and functionality. Section IV evaluates the proposed framework through case studies and comparisons with existing frameworks. It provides the results and analysis of the evaluation, including a comparison of maintainability, understandability, and extensibility with other frameworks. Finally, Section V concludes the paper and suggests future work.

II. Related Work

The use of model-driven engineering (MDE) as a novel approach in software development has attracted considerable attention from researchers in recent years. This methodology, by providing tools and techniques that help developers manage complex systems through high-level models and automation tools, has quickly gained popularity among traditional software development methods.

Vaupel et al. [6], introduced a modeling language and infrastructure for developing Android applications that supports various user roles. This language allows developers to continuously adjust and modify application content, while end users utilize specific content. This approach enables the development of flexible applications at different levels of abstraction and includes the modeling of both standard and custom elements. They demonstrated this approach by creating two applications, a phonebook manager and a conference guide. The approach consists of three main parts: generating Android projects from models, deploying projects, and interpreting models by Android applications without needing redeployment.

Derakhshandi et al. [7], proposed a solution called MAndroid for developing 2D board games on Android. This method uses MDE to model and automatically convert models into code, making it easier to detect and resolve errors. The MAndroid framework fully generates classic multiplayer games for Android devices. The approach was evaluated by implementing three games and assessing their usability and performance. The software development process is divided into two phases: modeling and conversion in Eclipse, followed by compiling and building the code in Android Studio.

Blanco and Lucrecio [8] addressed the challenges of cross-platform development and proposed a new approach that supports the expansion and inclusion of new platforms. This approach, by using a general-purpose language, raises the level of abstraction and separates the software from platform details. Automatic conversions generate executable codes that can be deployed across different platforms. The proposed approach was evaluated in four stages, including reconstructing an existing system and testing with both experts and novice developers. Additionally, support for cross-platform testing was introduced.

Gharaat et al. [9], introduced a framework called ALBA for developing location-based Android applications. This framework includes a domain-specific modeling language, modeling tools, and a plugin for converting models into code. The modeling tool allows novice designers to model location-based applications accurately. Evaluations showed that ALBA is promising in terms of usability and the quality of generated applications. The framework allows users to design location-based applications using the ALBA editor and then convert them into Android code.

Ammar [10] proposed a model-based approach for developing mobile application user interfaces. This approach, by using modeling and model transformation, enables the automatic generation of user interfaces. The proposed system utilizes standards and technologies such as the Eclipse Modeling Framework (EMF) and the ATL transformation language. The process involves two main stages: defining meta-models (AUI, CUI, and FUI) and converting them into source code. This approach covers different levels of abstraction and generates the final user interface for the Android operating system.

Mehrabi et al. [11], proposed a framework called HealMA for model-driven development of IoT-based Android health monitoring applications. This framework is designed to manage the heterogeneity of hardware and software systems and accelerate the development of such systems. HealMA includes a domain-specific modeling language, a graphical modeling editor, validation rules, and a model-to-code transformation engine. Evaluations showed that this framework is practical and useful for the automatic generation of health monitoring applications. HealMA aids in the rapid development of remote health monitoring applications and consists of four main components: domain concepts, a graphical editor, validation rules, and a code transformation engine.

Shamsujoha et al. [12], in a comprehensive study, examined the use of model-driven engineering in the development of mobile applications. They reviewed more than a thousand research articles and demonstrated that this method, by simplifying the development process, increasing the level of abstraction, and improving software quality, can address the challenges of mobile application development. The most important objectives of MDE in this field are architecture, domain modeling, and automatic code generation. Additionally, the study showed that model-driven methods can significantly improve productivity, scalability, and software reliability. The results of this research can be very useful for developers and researchers in mobile software development.

Uncorrected Proof

José Barriga et al. [13], proposed an innovative approach to the development and simulation of IoT systems. This model-based approach enables the design and deployment of the most complex IoT environments without the need for manual programming. Using a comprehensive meta-model, this method allows users to graphically model various elements of an IoT system, such as sensors, actuators, and cloud nodes. The models are then automatically converted into code, creating the desired simulation environment. This approach demonstrated its practical applicability and efficiency through two case studies in the fields of smart buildings and agriculture.

Núñez et al. [14], proposed an innovative approach called "Web Model-Driven" for the development of mobile applications, which focuses on the data layer. This approach is designed to provide access to data even in offline conditions, allowing applications to be used without network connectivity. Moreover, the Web Model-Driven approach is compatible with various operating systems, helping developers easily develop applications for diverse platforms. This approach, by using data persistence concepts and defining meta-models and specific architectural models, enables the design of stable data resources and the automatic generation of code for different platforms such as Android and Windows Phone.

These studies are compared in TABLE I based on six criteria, including type of meta-model (MM) (UML profile or EMF), the generated programming language (Android or cross-platform), the type of modeling language editor (graphical with GMF tools, graphical with Sirius tools, or textual), the type of modeling (structural or behavioral), and the domain of the generated application.

TABLE I
Comparison of related work

Research	Year	Type of MM		Programming Language	Type of Language Editor	Auto Code Generation	Type of Model		Application Domain
		UML	EMF				Structural	Behavioral	
[6]	2014	✗	✓	Android	Graphical GMF	✓	✓	✓	Daily Applications
[7]	2021	✗	✓	Android	Graphical	✓	✓	✓	2D Multiplayer Games
[8]	2021	✓	✗	Multi-platform	Textual	✓	✓	✓	GUI & Domain Model
[9]	2021	✗	✓	Android	Graphical GMF	✓	✓	✗	Location-based Applications
[10]	2021	✗	✓	Multi-platform	Graphical GMF	✓	✓	✗	Graphical Interface
[11]	2022	✗	✓	Android	Graphical Sirius	✓	✓	✗	Health Monitoring Application
[12]	2020	✓	✓	Multi-platform	Graphical Tree-based	✓	✓	✗	Mobile Data Layer
[13]	2019	✓	✗	Android	Textual	✓	✓	✗	Commercial Store Application

Although model-driven approaches for Android and cross-platform applications have been the subject of numerous studies, audio-based applications have received relatively little attention in this context. By offering a coordinated, domain-specific modeling, automated code generation, and customization approach tailored to the needs of audio applications development, our framework fills this gap.

III. PROPOSED FRAMEWORK

In this section, the proposed framework for model-driven development of Android audio-based applications is examined. This framework is presented as a solution to accelerate and improve the development process of mobile audio-based applications. The various stages of this framework include modeling processes, validation, automatic code generation, and customization of the generated code for finalizing the product.

The main goal of the proposed solution is to provide a framework that improves the development of Android audio-based applications using model-driven engineering. This framework is based on domain-specific metamodels. The process of using the proposed framework includes several main steps, which are demonstrated in Figure 1.

In the first step, the requirements of the audio-based application are accurately collected. These requirements include various features that the application must support, such as audio capabilities, playback controls, and user interactions. These requirements are then represented in a high-level model that provides an overview of the application.

After defining the initial model, validation constraints are applied to the model. These constraints help ensure the model's correctness and completeness. Models containing defects or

Uncorrected Proof

errors are identified and corrected by the system to prevent problems in later development stages.

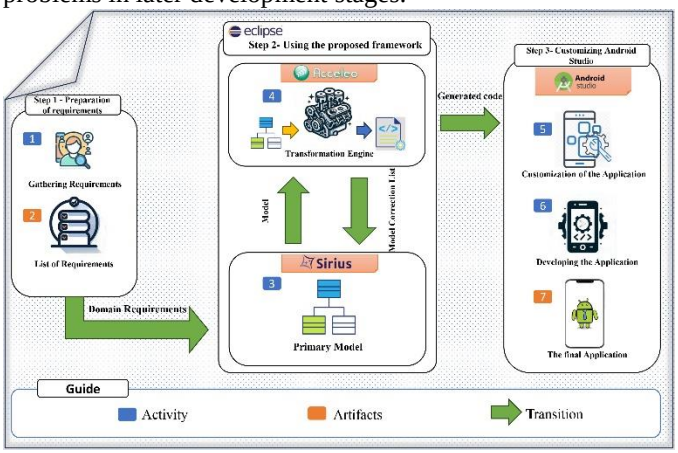


Fig. 1. The process of using the proposed framework to develop an audio application

After the model is approved, the transformation to code begins. For this purpose, automatic code generation tools like Acceleo are used to convert the defined models into Android executable code. This code includes all the necessary components for audio-based applications, such as classes related to activities, fragments, layouts, and resources.

If the developer requires specific changes or further customization, the generated code can be imported into Android Studio. Android Studio, as a complete development environment, allows manual modifications to the generated code. This stage is used to improve and finalize the code.

Innovation Boundary and Value Addition: The proposed framework is a novel solution specifically designed for the rapid development of audio-based mobile applications. The originality of the framework is evident in the following key aspects:

- **Metamodel Design**: The domain-specific metamodel was designed from scratch to represent the essential components of audio-based applications, such as audio playback, navigation flows, and user interactions. This metamodel distinguishes our framework from generic modeling approaches.
- **Graphical Editor**: A custom graphical editor was developed to enable intuitive modeling of applications, simplifying the design process for developers with limited coding knowledge.
- **Model-to-Code Transformation**: While the framework uses Acceleo as the transformation engine, the transformation templates were written specifically for this framework. These templates address the unique requirements of audio-based applications and generate Android-specific code tailored to the metamodel's constructs.
- **Domain-Specific Customizations**: The framework includes adaptations for efficiently handling audio assets and implementing real-time user interaction logic, which are not addressed by existing tools or frameworks.

By integrating these components, the framework offers a unique, efficient, and automated approach to developing audio applications, reducing both development time and effort. The originality of this research lies in its tailored approach to addressing the unique requirements of audio-based applications. The framework provides novel approaches that can facilitate the design and development process in this specific domain by combining a domain-specific metamodel, a dedicated graphical editor, and an automation process generation engine. These zoning features also distinguish it from more general-purpose MDE frameworks.

The framework is structured into four main sections, each focusing on a distinct aspect of the proposed solution. Section III.A introduces the metamodel that forms the foundation of the framework, III.B describes the graphical editor for visual modeling, III.C explains the transformation engine for automating code generation, and III.D details the final customization process in Android Studio.

A. Proposed Metamodel

In this section, concepts such as *activities* and *fragments* are inspired by the Android application architecture, as described in [15]. These elements are integrated into the proposed metamodel to ensure compatibility with existing Android development practices while extending their functionality for audio-based applications. The metamodel is presented in Figure 2. The provided image depicts a metamodel for an Android audio-based application. Here is an overview of each class and its relationships within the model:

1. **App**: The central class representing the overall Android application. It defines the application's name and method of handling audio (howToPlay), making it a starting point for defining the app's core attributes.
2. **Configuration**: Contains essential settings for the app, such as version, language, SDK versions, and screen orientation. This class is defined to centralize all configuration-related data, which is critical for setting up the app environment.
3. **ScreenOrientation** (enum): Specifies various screen orientations like portrait, landscape, and sensor. It helps standardize how the app adjusts to different device orientations.
4. **Language** (enum): Defines the languages supported by the app, such as English and Persian. This ensures that the app can cater to different linguistic audiences.
5. **HowToPlay** (enum): Indicates the method used to play audio, such as TextToSpeech or MediaPlayer. This class enables flexibility in how audio is handled within the app, based on its intended functionality.
6. **Activity**: Represents a screen or activity within the app, managing its orientation and behavior. This class is fundamental for structuring the app's interface and controlling how users interact with different screens.
7. **SplashActivity**: A specific type of Activity used for the splash screen, with an added delay time. This class ensures that the splash screen can be timed and managed independently from other activities.
8. **Theme**: Represents the app's visual theme, specifically aspects like the status bar color. This class is necessary

Uncorrected Proof

for maintaining the design consistency across the app's interface.

9. **Color**: Defines specific colors used in the app's theme. This class supports customization of the user interface by specifying named colors for different elements.
10. **Resource**: Represents any type of app resources, such as audio or image files. It centralizes access to different resources, making resource management easier.
11. **String**: Stores text strings used within the app. This class is essential for localization and for managing static text in a consistent way.
12. **File**: Represents individual files (e.g., audio or image files). It allows the app to manage multimedia content efficiently by linking it to the resource management system.
13. **Layout**: Defines the arrangement of UI components within an activity or fragment. This class is crucial for setting up how the app's interface is structured visually.
14. **Fragment**: Represents portions of a UI that can be reused within activities. This class allows for modular design, enabling developers to reuse UI components across different screens.
15. **Widget**: Represents individual UI elements, like buttons or images. This class is essential for managing user interface controls and how they interact within the app.
16. **Adapter**: Manages the connection between data and views (UI elements). It is defined to allow flexible handling of data in lists or other view components.
17. **ViewHolder**: Holds references to UI components, improving performance when binding data to views. This class is crucial in reducing resource usage when handling large datasets.
18. **Data**: Represents data entities used in the app. This class allows for the structured handling of data, linking it to the app's user interface and logic.
19. **ObjectClass**: Represents custom objects within the app's logic. It supports the creation of complex data structures for handling app-specific functionality.
20. **DataClass**: A structured class for managing datasets. This class helps with organizing and processing data in a structured and reusable way.
21. **Parameter**: Defines parameters passed between activities or fragments. It ensures flexibility by allowing inputs to be defined and processed dynamically.
22. **CardView, ButtonImage, TextView, ImageView**: Specialized widgets for displaying specific types of content, such as images, text, and buttons. These classes help manage specific UI components efficiently.
23. **Item**: Represents individual items that might appear in lists or collections within the app. This class ensures that each item's data is properly organized and managed.

In summary, each class in the metamodel is defined to encapsulate a specific aspect of the app's structure, resources, or behavior. This separation of concerns ensures that the app's design is modular, maintainable, and scalable.

B. Graphical Editor

A graphical editor is embedded in the proposed framework, allowing developers to visually model the components of their audio-based application. This editor enables the user to easily

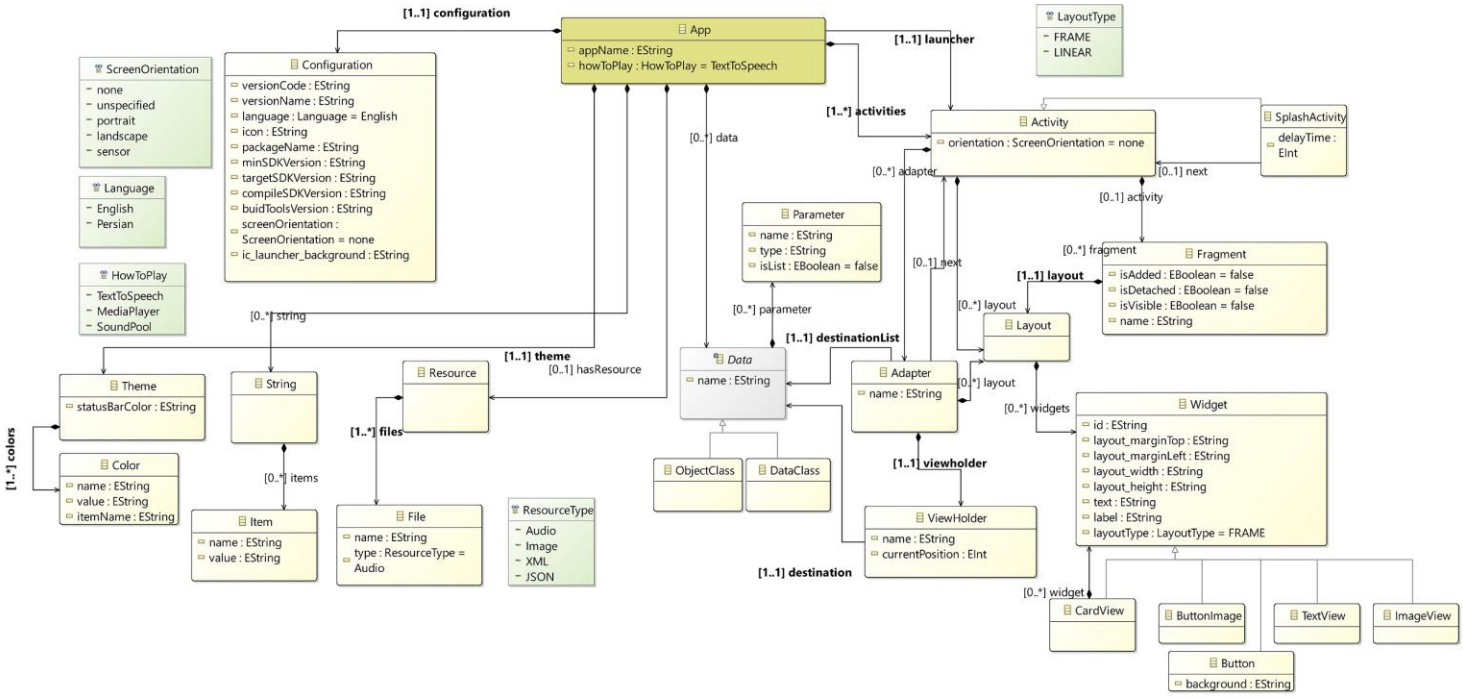
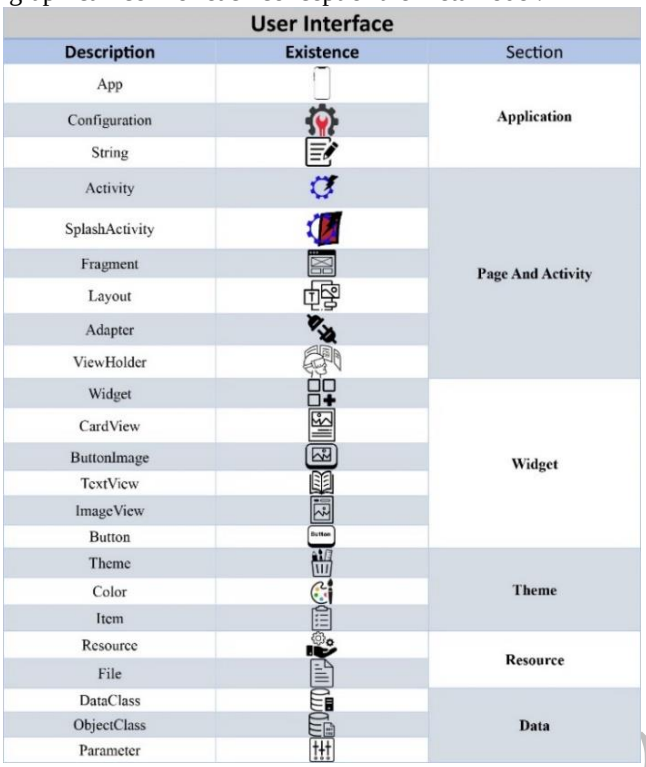


Fig. 2. The proposed metamodel

Proof

add elements like playback control buttons, audio files, and other user interface components to the model. Using this graphical tool facilitates the modeling process and allows developers to design applications more quickly. Figure 3 shows the graphical icon for each concept of the metamodel.



User Interface		
Description	**Existence**	Section
App		**Application**
Configuration		
String		
Activity		**Page And Activity**
SplashActivity		
Fragment		
Layout		
Adapter		
ViewHolder		
Widget		**Widget**
CardView		
ButtonImage		
TextView		
ImageView		
Button		
Theme		**Theme**
Color		
Item		
Resource		**Resource**
File		
DataClass		**Data**
ObjectClass		
Parameter		

Fig. 3. Mapping between metamodel elements and toolbox

The Section column organizes the components into logical groupings:

- **Application**: Deals with the overall structure and configuration of the app. It includes components like App, Configuration, and String, representing the high-level structure and setup of the app.
- **Page and Activity**: Focuses on user interaction and screen organization. It covers components like Activity, SplashActivity, Fragment, and Layout, all related to defining the app's pages and screen transitions.
- **Widget**: Involves the interface elements that users can interact with directly. It represents the UI controls (e.g., CardView, Button, ImageView) that users interact with.
- **Theme**: Manages the look and feel of the app. It includes components related to the app's appearance, like Theme and Color.
- **Resource**: Contains various resources like files and data that the app needs to function. It represents elements such as Resource and File, which manage assets like images, audio, and other external files.
- **Data**: Handles the internal data structures and information processing of the app. It covers DataClass, ObjectClass, and Parameter, which are related to handling and processing the app's data structure.

C. Transformation Engine

The transformation engine is one of the key components of the framework. This engine is responsible for converting high-level models into executable Android code. The transformation engine uses rules and algorithms based on the defined metamodel to generate appropriate code. This code is generated in languages such as Kotlin or Java and includes all the necessary components for a complete Android application.

The provided code snippets in Figure 4, are two Acceleo templates that generate Kotlin code for configuration and language classes. The first template creates a configuration class with properties like version code, version name, package name, and SDK versions. The second template generates an enum class for language codes, including English and Persian languages. These templates are used in a code generation process to automate the creation of these classes based on specific input data.

```
[template public generateConfiguration(configuration : Configuration)]
/**
 * Application configuration
 */
object Configuration {
    const val versionCode = "[configuration.versionCode/]"
    const val versionName = "[configuration.versionName/]"
    const val packageName = "[configuration.packageName/]"
    const val minSDKVersion = "[configuration.minSDKVersion/]"
    const val targetSDKVersion = "[configuration.targetSDKVersion/]"
    const val compileSDKVersion = "[configuration.compileSDKVersion/]"
    const val buildToolsVersion = "[configuration.buildToolsVersion/]"
    const val screenOrientation = "[configuration.screenOrientation.toString()/]"
}
[/template]

[template public generateLanguage(language : Language)]
enum class Language {
    [for (lang : Language | Language.allInstances())]
    [lang.toString().toUpperCase()/],
    [/for]
}
[/template]
```

Fig. 4. The file to convert the model to the configuration code and the type of user language of the application

D. Code Customization in Android Studio

The code generated by the transformation engine is transformed to Android Studio, where developers can manually customize and improve the code. This stage includes final optimizations, adding specific features, and making additional adjustments. Android Studio is used as a powerful tool for this purpose, enabling developers to make necessary changes to prepare their applications for release.

IV. EVALUATION

The purpose of the evaluation process was to thoroughly test the efficacy and adaptability of the suggested framework across multiple scenarios. Using four different case studies that provide insights into the approach's practical usability and scalability, this study demonstrates the framework ability to address the obstacles involved in developing audio-based applications. These applications were selected to demonstrate how the framework can handle a range of requirements, such as managing instructional content, playing sound, and displaying images. Additionally, a significant portion of the application code was automatically generated from the model. The scenarios demonstrate how the framework supports a broad range of complexity and functionality while abstracting away

Uncorrected Proof

the specifics of application programming.

In the following, first, the framework is evaluated based on four case studies in Section IV.A. Then, the development effort is estimated for each MDE and traditional approach in Section IV.B. Finally, in Section IV.C, the framework is compared with two other MDE frameworks, which are used to develop Android applications.

A. Case Study Evaluation

In this research, four Android applications are developed as case studies. These applications include features such as sound playback, image display, and management of educational information. These applications are modeled using the proposed framework, and the corresponding code is semi-automatically generated. The case studies are described as follows.

1) EnglishInSound Application: This application is designed for teaching English words with sound playback. Users can hear the pronunciation of words by touching images or words. The proposed framework enabled developers to create models for sound management and the user interface in a simple and efficient manner. Figure 5 presents the first and second pages of the app.

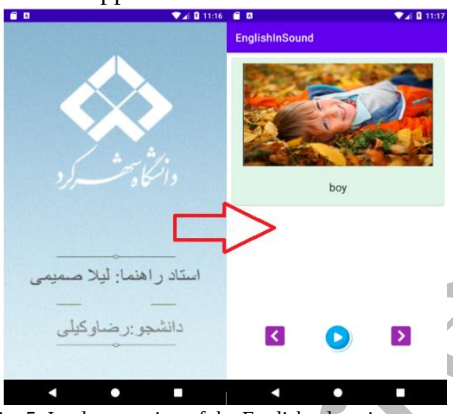


Fig. 5. Implementation of the English education app

2) FlagsOfCountries Application: This application is designed for teaching the flags of countries worldwide. Users can see the flag of each country, and by touching it, the name of the country is played aloud. This application uses the TextToSpeech library for sound playback. Figure 6 shows first and second pages of the app.

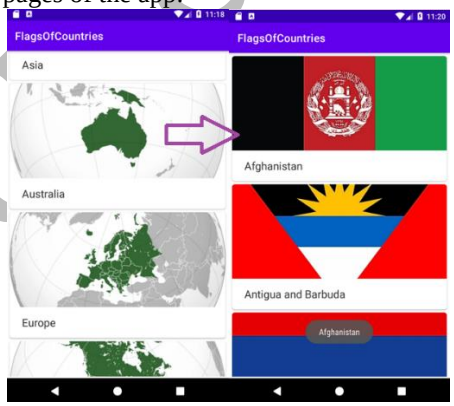


Fig. 6. Implementation of the flag education program of the continents of the world

3) AsiaFlags Application: The AsiaFlags application is similar to the previous application but displays only the flags of Asian countries. This application also utilized the proposed framework, allowing developers to quickly implement similar features in a simple model. Some parts of implementation code are differed from the previous app. Figure 7 shows one page of the app.

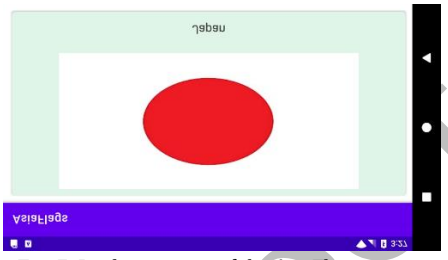


Fig. 7. Implementation of the AsiaFlags app

4) AsiaFlagsFragment Application: This application has the same functionality as the AsiaFlags program but uses the Fragment technology to load information (see Figure 8). The proposed framework, by supporting newer Android patterns (such as Fragments), allowed developers to easily use these features in the generated models and code.

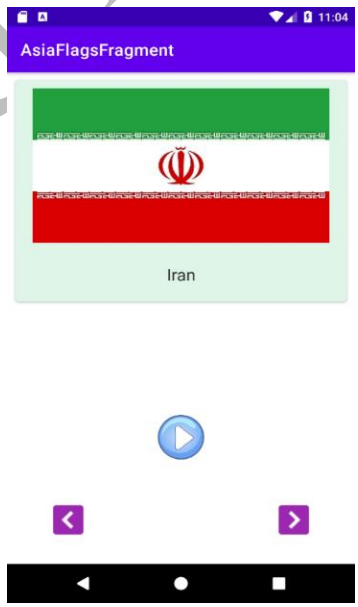


Fig. 8. Implementation of the AsiaFlagsFragment app

Table II compares the features of four different Android applications that use audio as a key component. The applications are categorized based on their program complexity, user interface design, data structure, purpose, and various capabilities.

The table shows the varying levels of complexity, from *Low* to *High*, and indicates how each program handles data structures, either using a JSON file or a class file, depending on the application's requirements. Additionally, it highlights key functional aspects, such as the information separation, display architecture patterns (Adapter or Fragment), and the audio playback configuration type, which ranges between TextToSpeech and MediaPlayer. For example, some applications focus on disaggregated training with features like

Unconfected Proof

displaying an item, while others are geared towards unsegregated training with broader functionalities, like displaying a list of items and providing a welcome page. This comparison gives insight into how different applications are designed and configured to meet their unique training purposes.

TABLE II
COMPARISON OF FOUR CASE STUDIES

Criterion		AsiaFlags	AsiaFlagsFragment	FlagsOfCountries	EnglishInSound
Program Complexity	Low	✸			
	Medium		✸		✸
	High			✸	
User Interface	Display a list of items			✸	
	Display an item	✸	✸		✸
Data Structure	JSON File		✸		✸
	Class File	✸		✸	
Purpose of the program	Disaggregated training	✸	✸	✸	
	Unsegregated training				✸
Information separation	Yes	✸	✸	✸	
	No				✸
Capabilities	Welcome page				✸
	Show case details	✸	✸	✸	
Display architecture pattern	Adapter	✸		✸	
	Fragment		✸		✸
Audio playback configuration type	TextToSpeech	✸	✸	✸	
	MediaPlayer				✸

The tree representation of the EnglishInSound application model is presented in Figure 9. The model consists of a main activity (MainActivity) that contains various layers and widgets such as buttonsLayout and buttons like nextButton, btnPlay, and prevButton. This application also includes a fragment (EnglishFragment) and a splash activity (SplashActivity). The model comprises resources such as sound files (soundItem) and data classes (EnglishData) with various parameters to manage the application's information. Additionally, a theme (Theme) and text strings (String) are used for different app settings. Figure 10 illustrates the graphical view of this model, which is designed by the graphical editor of the framework.

Table III presents the distribution of metamodel class instances across four applications developed using the proposed framework. Each row represents a specific metamodel class, and the values in the columns indicate the number of instances of that class in each application. Subcategories such as DataClass and ObjectClass under Data, and CardView, TextView, and other widgets under Widget, provide additional details about the structure of each application.

The analysis highlights the framework's adaptability in supporting diverse application requirements. The consistent presence of core classes, such as App, Configuration, and Activity, demonstrates the framework's ability to model essential components across different contexts. Meanwhile, the variability in the use of classes like Fragment, Adapter, and widgets reflects the framework's flexibility to accommodate unique application features, such as dynamic navigation or custom user interfaces. This adaptability enables developers to easily tailor the framework to meet specific needs, thereby enhancing its applicability in real-world scenarios.

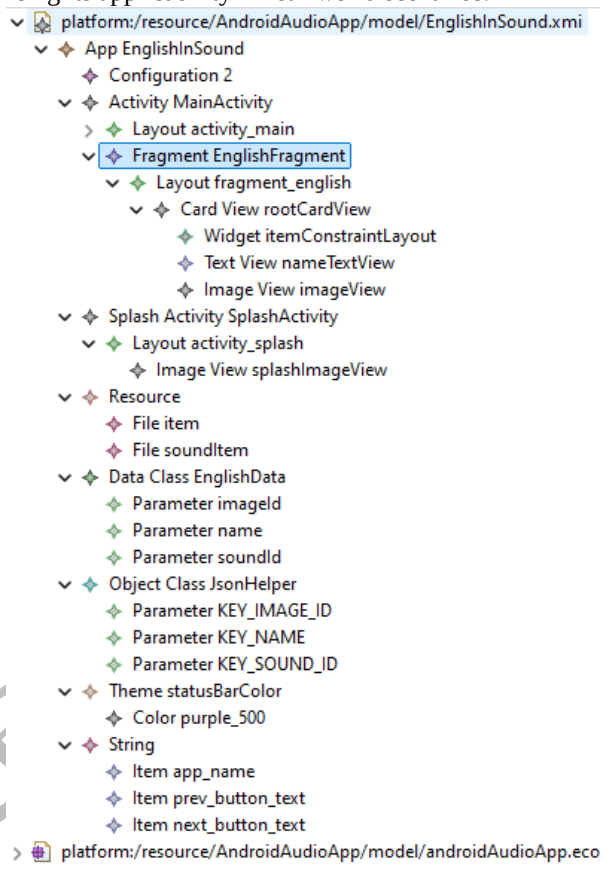


Fig. 9. Tree representation of the EnglishInSound app model

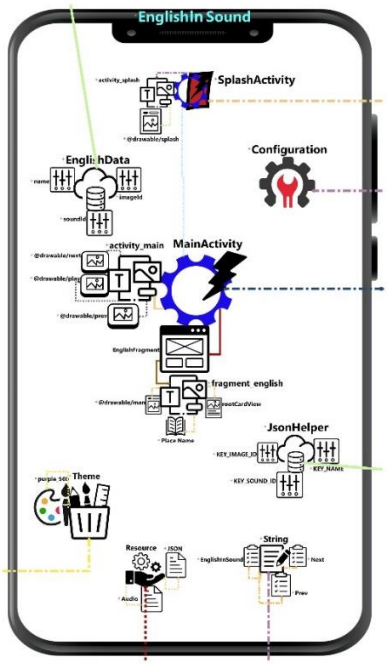


Fig. 10. Graphical view of the EnglishInSound app model

Uncorrected

TABLE III
DISTRIBUTION OF METAMODEL CLASS INSTANCES ACROSS APPLICATIONS

Metamodel Class	AsiaFlagsFragment	AsiaFlags	FlagsOfCountries	EnglishInSound
App	1	1	1	1
Configuration	1	1	1	1
Resource	1	1	0	1
Activity	1	1	2	1
SplashActivity	0	0	0	1
Fragment	1	0	0	1
Adapter	0	1	2	0
ViewHolder	0	1	2	0
Layout	2	2	4	3
String	1	1	1	1
Theme	1	1	1	1
Color	2	1	1	1
Item	3	1	1	3
File	2	1	0	2
Parameter	6	4	4	6
Data				
- DataClass	1	1	2	1
- ObjectClass	1	1	2	1
Widget				
- CardView	1	1	2	1
- ButtonImage	2	0	0	3
- TextView	1	1	3	1
- ImageView	1	1	2	2
- Button	1	0	0	0

After creating a model for each case study, the code generator of the framework is applied. By means of the code generator in the framework, 67% of the EnglishInSound app code is generated automatically from the given model. Here's a breakdown of the areas that can be automated:

- Activity and Fragment Setup: The basic structure of the `SplashActivity` and `MainActivity` classes, including the setup of the `Fragment` and the corresponding `JsonHelper` class, is generated automatically from the model.
- UI Components and Events: The layout and interaction of the UI components, such as the buttons, image views, and text views, is generated from the model's definition of the user interface. The click event handling for the play/pause button, as well as the previous and next buttons, is generated from the specification of the user interactions.
- Media Player Management: The code related to loading, playing, and controlling the audio files, as well as the integration with the UI components, is generated from the specification of the audio playback functionality in the model.
- Data Model and JSON Parsing: The `EnglishData` data class and the `JsonHelper` class, which handle the parsing of the JSON data, is generated directly from the data model.
- Fragment Replacement: The logic for replacing the `EnglishFragment` within the `MainActivity` is generated from the specification of the navigation and content management.

The remaining of the code would require manual implementation, such as any custom business logic or domain-specific functionality that is not directly captured in the metamodel.

Figure 11, the stacked bar chart, illustrates a comparison of the number of lines of code (LOC) generated automatically versus those written manually for four different applications. For example, AsiaFlags has 263 total lines of code, with 247 lines (94%) automatically generated and 16 lines must manually be written. FlagsofCountries shows 364 total lines, 323 of which (89%) were generated automatically. EnglishInSound had 278 total lines, 186 (67%) automatically generated, and AsiaFlagsFragment had 269 total lines, with 209 (78%) automatically generated. The data reveals that AsiaFlags had the highest percentage of automatically generated code, while EnglishInSound had the lowest.

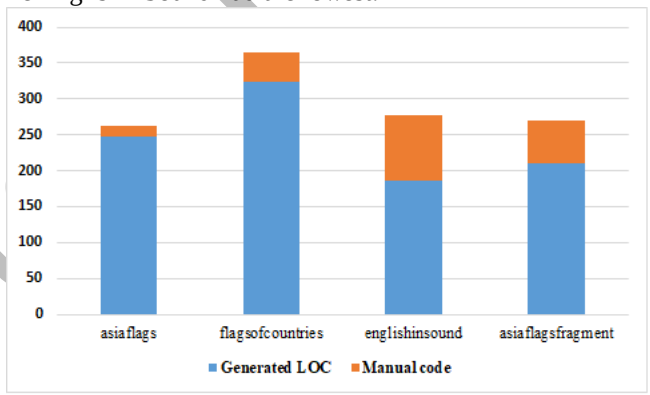


Fig. 11. Stacked bar chart comparing automatically generated and manually written lines of code (LOC) for the four case studies

B. Evaluation of Development Effort and Time

In subsection IV.A, Table III demonstrates the framework's adaptability through the distribution of metamodel class instances. Building on this, the evaluation in this section highlights how this flexibility reduces development effort and supports the creation of diverse application features, including dynamic navigation, custom user interfaces, and efficient data handling.

To evaluate the development effort required by the proposed model-driven framework and compare it to the traditional manual coding approach, we conducted an experiment with three participants of varying expertise levels:

- Participant 1: An expert in modeling with limited Android development experience.
- Participant 2: An expert in Android development with no prior exposure to model-driven engineering.
- Participant 3: A beginner with no significant expertise in either modeling or Android development.

The experiment consisted of the following stages as shown in Table IV:

Uncorrect Proof

1. **Learning Framework Concepts:** Participants familiarized themselves with the framework, including the metamodel structure, graphical editor, and transformation process. The required time varied based on prior knowledge, ranging from 1 hour for the modeling expert to 3 hours for the beginner. The learning phase is only required once and was excluded from the time calculations for individual case studies.
2. **Familiarization with the Graphical Editor:** All participants learned to use the graphical modeling tool, which has a simple and intuitive interface. The time required ranged from 30 minutes for Participant 1 to 1 hour for Participant 3.
3. **Model Creation:** For simple case studies, model creation required 20 minutes for Participant 1, 30 minutes for Participant 2, and 45 minutes for Participant 3. For complex models, the effort increased to 1 hour for Participant 1, 1.5 hours for Participant 2, and 2 hours for Participant 3.
4. **Manual Code Customization:** While the framework generates a significant portion of the code, certain custom logic must be added manually. The time required for this step is presented in Table VI.
5. **APK Generation:** The final step of generating and deploying the APK file on an Android device took 10 to 15 minutes for all participants.

TABLE IV
MODEL-DRIVEN DEVELOPMENT EFFORT

Phase	Modeling Expert	Android Expert	Beginner
Learning Framework Concepts	1 hour	2 hours	3 hours
Familiarization with Editor	30 min.	45 min.	1 hour
Model Creation (Simple)	20 min.	30 min.	45 min.
Model Creation (Complex)	1 hour	1.5 hours	2 hours
APK Generation	10 min.	10 min.	15 min.

For the traditional approach, we assumed that an Android development expert manually codes the entire application. The average effort per line of code (LOC) was estimated at 2 minutes, excluding time for learning Android programming. APK generation time was considered equivalent to the model-driven approach. The result shows in Table V. This calculation is done for manual code of MDE approach which is presented in Table VI.

TABLE V
Traditional Development Effort

Application	LOC	Manual Effort (Minutes)	APK Gen.	Manual Effort (Hours)
AsiaFlags	263	526	10	~9
FlagsOfCountries	364	728	10	~12 ⅓
EnglishInSound	278	556	10	~9.5
AsiaFlagsFragment	269	538	10	~9□

TABLE VI
Manual Code Customization Effort in the Model-Driven Approach

Application	LOC	Manual Effort (Minutes)	Manual Effort (Hours)
AsiaFlags	16	32	~0.5
FlagsOfCountries	41	82	~1⅕
EnglishInSound	92	184	~3
AsiaFlagsFragment	60	120	~2

Learning the framework's concepts is a one-time cost and was excluded from individual case study times. Once familiar, all participants could complete tasks efficiently.

By automating key aspects of the development process, the proposed framework significantly reduces the effort and time required to develop Android applications. While traditional coding requires 9 to 12 ⅓ hours per application, the model-driven approach completes the same task in 1.5 to 3 hours, even for beginners. This highlights the framework's efficiency and its ability to simplify application development for developers with varying levels of expertise.

The comparison of development effort between the model-driven and traditional approaches highlights the efficiency of the proposed methodology. The results demonstrate that by automating repetitive coding tasks and providing intuitive modeling tools, the framework reduces development time while maintaining a high standard of application quality.

C. Comparison with Other Frameworks

To compare the proposed framework with two other frameworks (ALBA [9] and HealMA [11]), three main criteria are considered: maintainability [16], understandability [17], and extensibility [18]. These criteria are to evaluate the quality of different frameworks in application development.

Maintainability refers to the ability to make changes and maintain the system without major alterations to its structure. Formula (1) shows maintainability as a function of the number of classes (NC), attributes (NA), references (NR), the maximum hierarchical level (DITmax), and the maximum fan-out (Fanoutmax). **Lower values shows better maintainability** [16].

$$\frac{NC + NA + NR + DIT_{Max} + Fanout_{Max}}{5} \quad (1)$$

Understandability refers to the developers' ability to understand the structure of the model and code. Formula (2) shows that Understandability is computed based on the number of predecessors (PREDc) and number of classes (NC). **Higher values demonstrate better understandability** [17].

$$\frac{\sum_{K=1}^{NC} PRED + 1}{NC} \quad (2)$$

Extensibility refers to the ability to add new features to the system without requiring extensive changes to the existing structure. Formula (3) for extensibility is based on the number of inherited features (INHF) and total number of features (NTF). **Higher values indicate better extensibility** [18].

Uncorrected Proof

$$\frac{INHF}{NTF} \quad (3)$$

As shown in Table VII, the proposed framework has achieved high maintainability due to the use of high-level models and automatic code generation. It received a better score compared to the other two frameworks, indicating its strong maintainability capabilities. Through visual representations and graphical tools, the proposed approach has provided better understandability. It offers higher understandability compared to the ALBA [9] and HealMA [11] frameworks. Additionally, the proposed framework excels in extensibility due to its higher flexibility in adding new features.

TABLE VII
COMPARISON OF THREE FRAMEWORKS

Framework	Maintainability	Understandability	Extensibility
Our approach	26.2	0.29	0.36
ALBA [9]	26.6	0.17	0.12
HealMA[11]	35.6	0.22	0.24

The results of the investigation demonstrate that the proposed framework offers a scalable and maintainable solution for upcoming developments in mobile software development, in addition to addressing the technical difficulties of creating audio-based applications. This study adds to the larger field of model-driven engineering by concentrating on domain-specific requirements and providing insights that can guide both academic research and practical uses.

V. CONCLUSION AND FUTURE WORK

This section concludes the findings of the study in Section V.A and outlines potential directions for future work in Section V.B.

A. Conclusion

Through this study, we highlight how model-driven engineering can be used to facilitate development for audio-based application domains. We describe the comprehensive, audio-based application design capabilities of our suggested framework, which eventually bridges the gap between practical implementation and high-level modeling.

The increasing demand for audio programs like audiobooks and language-learning applications has put this new pressure on developers to produce their products faster and more efficiently. The domain-specific modeling languages have shown a lot of promise in tackling these issues. The MDE framework is beneficial to software engineering researchers and practitioners because it can significantly reduce development time and effort without sacrificing application quality. Its embrace could boost progress in fields that want speedy, high-quality applications.

The study proposes an MDE-based holistic framework that encompasses three critical layers: (1) a domain-specific metamodel; (2) a graphical editor; (3) a model-to-code transformation tool. The metamodel encapsulates the core elements of audio-based applications, providing a high-level abstraction that simplifies the design process. The graphical editor allows developers to create and modify the models visually, without requiring extensive knowledge about programming, and the transformation tool automatically generates an Android executable code. All of these components work together to accelerate the development process and allow developers to create highly specialized apps with maximum efficiency.

The performance of the proposed framework validates its potential to tackle important issues in the development of audio-based applications. Establishing metrics such as maintainability, understandability, and extensibility highlights its potential for simplifying development processes and adapting to a variety of application requirements. The next steps would be to improve the flexibility and scalability of the framework so that it can be increasingly applicable to the more complex real-world use cases.

Although case studies have been done that reflect typical audio-based applications, the scalability of this framework to more advanced, real-world use cases still requires further research. This current evaluation encapsulates the frameworks that allow it to automate key aspects of the development process, reducing manual effort and accelerating the creation of various applications. These results indicate that the framework holds promise for real-world applicability, especially in domains requiring rapid development and high customization. In the future, we plan to test the framework on complex real-world applications to see how well it scales and performs in harder conditions.

The demonstrated effectiveness of this paradigm in reducing development workload and increasing software quality suggests that similar concepts could be applied to a variety of fields where adaptability and short time-to-market are critical. With increasingly specialized applications in demand, such an approach provides a solid foundation for addressing new software engineering challenges.

B. Future Work

Although the proposed framework yields promising results, there is still opportunity for development to enhance its applicability and performance:

1. Support for Dynamic Structures: Future versions of the framework could add support for adaptation at runtime and dynamic behaviors. This can lead to more effective handling of resources that can be loaded on the fly as well as components that are generated at runtime, allowing for increased responsiveness and flexibility within applications.

2. Scalability to Complex Applications: The current framework has been tested with relatively simple case studies. Future work would involve applying the framework to real-world, complex applications to assess its scalability and identify areas for further, and undoubtedly necessary, improvement. This would provide a much more robust validation of its capabilities across a wide range of scenarios.

3. Cross-Platform Portability: Currently, the framework is intended for Android development. To make it even more

Uncorrected Proof

useful, extending the training to support additional platforms, such as iOS or cross-platform frameworks (e.g., Flutter), would increase its versatility and applicability across diverse development environments.

4. Advanced Evaluation and Comparison: More in-depth practical experiments and comparisons with traditional development methods can help provide additional validation of the effectiveness and efficiency of the framework. Comparative quantitative analyses on development costs, time savings, application performance, etc., would yield more profound insights and showcase its advantages.

5. Extending Metamodel and Tools: Further automating domain-specific elements in the metamodel through the graphical editor and transformation tool could potentially simplify the development further. If the utility and adaptability of the framework were increased by supporting a wider range of programming languages and application types.

In addressing these directions, the framework presented in this paper will be able to evolve, allowing developers to respond to the increasing requirements laid by the development of larger and faster-changing applications in a more effective and efficient manner.

REFERENCES

[1] R. F. Paige, N. Matragkas, and L. M. Rose, "Evolving models in model-driven engineering: State-of-the-art and future challenges," *Journal of Systems and Software*, vol. 111, pp. 272-280, 2016.

[2] R. France and B. Rumpe, "Model-driven development of complex software: A research roadmap," in *Future of Software Engineering (FOSE'07)*, 2007: IEEE, pp. 37-54.

[3] M. Abbasi *et al.*, "In-Depth Analysis of Mobile Apps Statistics: A Study and Development of a Mobile App," in *2023 18th Iberian Conference on Information Systems and Technologies (CISTI)*, 2023: IEEE, pp. 1-7.

[4] L. M. Poupis, D. Rubin, and L. Lteif, "Turn up the volume if you're feeling lonely: The effect of mobile application sound on consumer outcomes," *Journal of Business Research*, vol. 126, pp. 263-278, 2021.

[5] A. M. Rapatsalahy, H. Razafimahatratra, T. Mahatody, M. Ilie, S. Ilie, and R. N. Razafindrakoto, "Automatic Generation of Object-Oriented Code from the ReLEL Requirements Model," *SYSTEM THEORY, CONTROL AND COMPUTING JOURNAL*, vol. 1, no. 1, pp. 36-47, 2021.

[6] S. Vaupel, G. Taentzer, J. P. Harries, R. Stroh, R. Gerlach, and M. Guckert, "Model-driven development of mobile applications allowing role-driven variants," in *International Conference on Model Driven Engineering Languages and Systems*, 2014: Springer, pp. 1-17.

[7] M. Derakhshandi, S. Kolahdouz-Rahimi, J. Troya, and K. Lano, "A model-driven framework for developing android-based classic multiplayer 2D board games," *Automated Software Engineering*, vol. 28, no. 2, pp. 1-57, 2021.

[8] J. Zanuzzio Blanco and D. Lucrédio, "A holistic approach for cross-platform software development," *arXiv e-prints*, p. arXiv: 2104.14614, 2021.

[9] M. Gharaat, M. Sharbaf, B. Zamani, and A. Hamou-Lhadj, "ALBA: a model-driven framework for the automatic generation of android location-based apps," *Automated Software Engineering*, vol. 28, no. 1, pp. 1-45, 2021.

[10] L. B. Ammar, "An Automated Model-Based Approach for Developing Mobile User Interfaces," *IEEE Access*, vol. 9, pp. 51573-51581, 2021.

[11] M. Mehrabi, B. Zamani, and A. Hamou-Lhadj, "HealMA: a model-driven framework for automatic generation of IoT-based Android health monitoring applications," *Automated Software Engineering*, vol. 29, no. 2, pp. 1-41, 2022.

[12] M. Shamsujjoha, J. Grundy, L. Li, H. Khalajzadeh, and Q. Lu, "Developing Mobile Applications Via Model Driven Development: A Systematic Literature Review," *Information and Software Technology*, vol. 140, p. 106693, 2021.

[13] J. A. Barriga, P. J. Clemente, E. Sosa-Sánchez, and Á. E. Prieto, "SimulateIoT: Domain Specific Language to Design, Code Generation and Execute IoT Simulation Environments," *IEEE Access*, vol. 9, pp. 92531-92552, 2021, doi: 10.1109/ACCESS.2021.3092528.

[14] M. Núñez, D. Bonhaure, M. González, and L. Cernuzzi, "A model-driven approach for the development of native mobile applications focusing on the data layer," *Journal of Systems and Software*, vol. 161, p. 110489, 2020.

[15] L. Guo, *The First Line of Code: Android Programming with Kotlin*. Springer Nature, 2022.

[16] M. Genero and M. Piattini, "Empirical validation of measures for class diagram structural complexity through controlled experiments," in *5th International ECOOP workshop on quantitative approaches in object-oriented software engineering*, 2001: Citeseer.

[17] F. T. Sheldon and H. Chung, "Measuring the complexity of class diagrams in reverse engineering," *Journal of Software Maintenance and Evolution: Research and Practice*, vol. 18, no. 5, pp. 333-350, 2006.

[18] T. Arendt, F. Mantz, and G. Taentzer, "Uml model quality assurance techniques," *Philipps-Univ. Marburg, Marburg, Germany, Tech. Rep*, 2009.

Uncorrected Proof