ISSN 1846-6168 (Print), ISSN 1848-5588 (Online)
https://doi.org/10.31803/tg-20241004154503

Professional paper
Received: 2024-10-04, Accepted: 2024-10-21

A Model-Driven Architecture Solution for Multi-Platform Mobile App Development

Ayoub Korchi*, Mohamed Karim Khachouch, Younes Lakhrisi

Abstract: This paper presents a comprehensive Model Driven Architecture (MDA) approach for multi-platform mobile app development. We introduce a UML-based metamodel that encapsulates essential mobile app elements, including views, controls, resources, and events. Our approach leverages the Acceleo code generation tool to transform Platform Independent Models (PIMs) adhering to this metamodel into platform-specific source code. We demonstrate the effectiveness of our method through a case study, generating Android user interface code from a sample PIM. The results show that our approach can significantly streamline the development process for multi-platform mobile apps, reducing the need for platform-specific coding. This work contributes to the field of model-driven mobile development by providing a flexible and extensible framework for automatic code generation across multiple mobile platforms.

Keywords: Cross-platform; MDA; Mobile-development

1 INTRODUCTION

Smartphones have evolved over the years to become essential companions, incorporating a wide range of applications that simplify our daily lives. Since their emergence in the early 2000s [1], smartphone sales have seen exponential growth. In 2007, the year the iPhone was launched, around 122 million smartphones were sold worldwide [2, 3]. By 2023, this number had surged to over 1.3 billion units [4], highlighting the ubiquity of smartphones and the significance of mobile applications in our routines.

However, the proliferation of smartphones comes with the challenge of mobile platform diversity, which has led to the rise of cross-platform development. Each mobile operating system has its own specific programming languages and development tools, making native app development for each platform complex and costly [5]. This technical diversity requires developers to manage multiple development environments and maintain separate codebases, increasing the risk of inconsistencies and bugs.

To address this diversity, cross-platform development has emerged as a solution, allowing developers to create an application once and deploy it on multiple platforms like Android and iOS [6]. Cross-platform development can be implemented through various approaches such as compilation, interpretation, component-based methods, cloud solutions, or hybrid approaches. Among these, Model-Driven Architecture (MDA) is particularly noteworthy for its ability to abstract platform complexity, providing a more adaptable solution for developers.

Model-Driven Architecture (MDA), developed by the Object Management Group (OMG), is a software development approach based on the principle of separating application functionality from platform-specific implementation details. MDA focuses on creating Platform Independent Models (PIMs) that capture application functionality, which can then be automatically transformed into Platform-Specific Models (PSMs) and executable code. This method enables consistent and efficient code generation across various platforms, reducing development time and enhancing maintainability [7].

In our approach to mobile application development, we leverage MDA to automate source code generation, transforming PIMs into PSMs specific to each target platform. This automation reduces the development effort required for each platform, while ensuring code consistency and allowing flexibility for platform-specific adaptations.

The main objectives of this research are to explore the potential of MDA for streamlining mobile app development and to provide a framework that enhances efficiency and maintainability. The primary contributions of this work include a comprehensive UML-based metamodel for mobile applications, the implementation of an Acceleo-based code generation framework, and a demonstration of our solution through a case study, which illustrates the automatic generation of an Android user interface from a sample PIM model.

This article is structured as follows: Section 2 presents a literature review of existing cross-platform solutions and the background of MDA. Section 3 outlines the research methodology, including a description of the MDA approach, the proposed metamodel, and the Acceleo tool. Section 4 provides a case study demonstrating our solution, and Section 5 concludes with perspectives for future work.

2 LITERATURE REVIEW

To systematically review existing literature on model-driven mobile application development, we utilized the Kitchenham method [8, 9], a structured approach in software engineering for collecting, analyzing, and synthesizing data. This method involves three main phases: planning, execution, and reporting (Fig. 1). During the planning phase, we established research questions, search protocols, and criteria for study inclusion and exclusion. The execution phase entailed a comprehensive search across various databases, selecting relevant studies, assessing quality, and extracting key data. Finally, in the reporting phase, we synthesized findings to present a thorough overview, highlighting gaps and establishing a foundation for our research.

Our study centers on the Model-Driven Architecture (MDA) approach to mobile source code generation, which has become a crucial area due to the increasing demand for robust, maintainable, and scalable mobile applications. The MDA methodology supports mobile development by enabling developers to create high-level models that can be

automatically transformed into platform-specific code. This capability facilitates consistency between specifications and implementation, minimizes human error, and accelerates development. However, MDA presents challenges such as complex transformations, platform diversity, and integration with existing development tools. Our research questions, therefore, seek to identify the main models, methodologies, tools, benefits, limitations, and empirical results related to MDA in mobile application development.

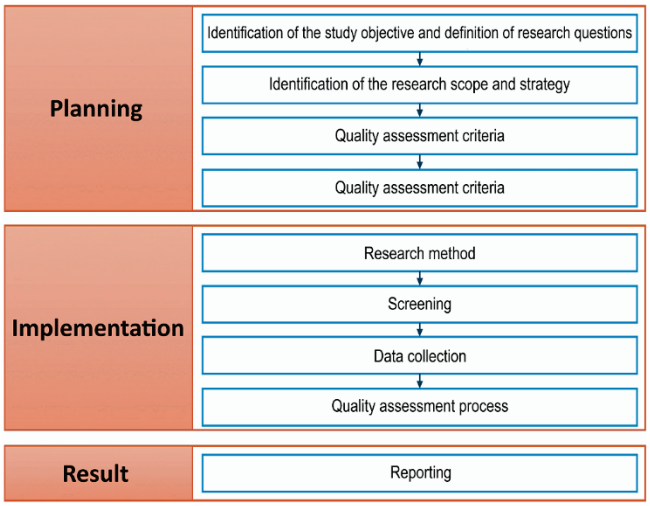

Figure 1 Kitchenham's SLR process

2.1 Research Questions

- What are the main models and methodologies used for generating mobile source code with MDA?
- What tools and development environments support this code generation?
- What are the advantages and limitations of using MDA for mobile code generation?
- What empirical studies and practical results have been documented on the effectiveness of this approach?
- What best practices and challenges have been encountered in implementing MDA for mobile applications?

2.2 Research Objectives

- Provide a comprehensive overview of the models and methodologies for mobile code generation using MDA.
- Identify available tools and development environments.
- Analyze the benefits, limitations, and challenges of this approach.
- Synthesize existing empirical results on the effectiveness of MDA.

2.3 Inclusion Criteria

- Articles published between 2000 and 2023.
- Studies focused on mobile code generation using MDA.
- Peer-reviewed journal and conference publications.
- Articles written in English.

2.4 Exclusion Criteria

- Non-peer-reviewed studies (e.g., white papers, blogs).
- Publications irrelevant to mobile code generation or not using MDA.
- Incomplete articles or those lacking concrete experimental data.

2.5 Data Sources

- Academic databases: IEEE Xplore, ACM Digital Library, SpringerLink, ScienceDirect, Google Scholar.
- Conference proceedings and journals focused on software engineering, modeling, and mobile development.

2.6 Search Strategy

- Formulating search queries with keywords like "Model-Driven Architecture", "MDA", "code generation", "mobile applications", "model-to-code", "source code generation".
- Applying time (2000-2023) and language (English) filters to refine results.
- Conducting manual searches in selected study bibliographies to identify additional relevant articles (snowball approach).

2.7 Metamodeling Approaches in MDA for Mobile Applications

Various studies emphasize the role of metamodels in the MDA framework. Metamodels provide the structure for Platform Independent Models (PIMs) and support the transformation into Platform Specific Models (PSMs). For example, [10] proposed a metamodel to design Android GUIs through UML object diagrams. Similarly, [11] utilized domain-specific languages (DSLs) to define the PIM, facilitating transformations to PSMs for Android. These approaches establish consistency in PIM-to-PSM transformations but often introduce additional complexity in terms of learning requirements and modeling expertise.

More recent works, such as [12], have extended this research by systematically reviewing MDA-based mobile development frameworks. Their study identifies recurring metamodeling approaches and highlights the need for adaptable frameworks capable of handling rapidly evolving mobile technologies. Our work contributes by offering a more streamlined metamodeling process, emphasizing ease of use and maintainability.

2.8 Code Generation Techniques

Code generation is a core aspect of MDA and is facilitated through various transformation languages and tools. Earlier works, such as [13], implemented transformations using QVT for M2M and Acceleo for M2T, generating Android-specific code. The Acceleo tool, in particular, supports the model-to-text (M2T) transformation essential for generating XML layouts and Java code. However, defining and maintaining transformation rules

often require specialized skills, as noted by [14].

Newer methodologies incorporate tools like Xtend, as used by authors in [15], who demonstrated an Android development approach based on MDA with flexible and efficient M2T transformations. These approaches, however, remain limited in platform coverage and frequently require substantial manual intervention. Our work seeks to minimize these issues by using Acceleo templates that streamline transformation steps and reduce the need for extensive tool proficiency.

2.9 Platform Coverage and Portability

Cross-platform support remains a significant motivation behind MDA approaches. Authors in [16] explored platform variability in MDA through role-based app development for both Android and iOS, highlighting the need for adaptable frameworks. However, these solutions tend to concentrate on UI elements, often neglecting aspects such as sensor integration, event handling, and complex data management, which limits their applicability for complete mobile applications. Our approach addresses these gaps by proposing a metamodel capable of supporting various platform-specific features while ensuring consistency across platforms.

2.10 Synthesis of Gaps in Existing Research

While existing studies have advanced MDA-based mobile application development, several gaps persist:

- **Complexity in Transformation:** Current solutions require specialized knowledge of transformation languages and tools (e.g., QVT, ATL), making them challenging to adopt for general developers. Our solution addresses this by simplifying the transformation process with predefined Acceleo templates.
- **Platform-Specific Limitations:** Many MDA approaches primarily focus on Android and lack support for other mobile platforms. Our work expands on these limitations by proposing a metamodel that allows for easy extension to other platforms.
- **Partial Coverage of Application Components:** Previous studies primarily address UI elements, leaving out critical functionalities like sensor integration, event handling, and advanced resource management. Our approach integrates these aspects into the metamodel to enable a more holistic mobile development process.

By addressing these gaps, our solution aims to provide a more accessible, efficient, and robust framework for MDA-based mobile development, enabling a streamlined transition from high-level models to platform-specific implementations.

3 RESEARCH METHODOLOGY

Our solution is founded on the MDA methodology, so the initial subsection will offer a comprehensive review of this approach, highlighting its strengths and weaknesses. We will then present the source metamodel we have developed, which plays a pivotal role in our work. This metamodel is essential to ensuring consistency when creating the platform-independent model. Lastly, we will display the Acceleo tool, which enabled us to directly generate native code from the platform-independent model.

3.1 MDA Approach

Model-Driven Architecture (MDA) is a software development approach that emphasizes the creation of abstract models and their transformation into executable code. This approach is based on the principle of separation of concerns, where high-level aspects of the application are modeled independently of the underlying technology. There are three levels of models: Conceptual Independent Model (CIM), Platform-Independent Model (PIM), and Platform-Specific Model (PSM), which ultimately leads to the generation of source code [17] [18] [19].

The Computation-Independent Model (CIM) is the first crucial step in the Model-Driven Architecture (MDA) software development process. By capturing business requirements in an abstract and general way, the CIM provides a solid foundation for the further development of the application while ensuring its independence from technical implementation details [20].

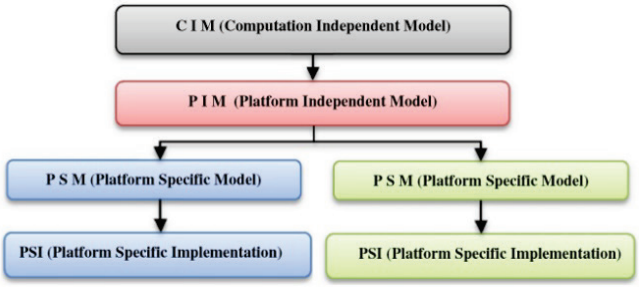

Figure 2 Model levels in the MDA approach

The Platform-Independent Model (PIM) represents an important intermediate stage in the Model-Driven Architecture (MDA) software development process. By offering a detailed yet generic representation of the application's functionality, the PIM allows the transition from the abstract business needs defined in the CIM to more concrete software specifications, while still preserving independence from specific implementation technologies [21].

The Platform-Specific Model (PSM) is a critical stage in the Model-Driven Architecture (MDA) software development process. By adapting the PIM models to meet the specific requirements of the target platform, the PSM enables the automatic generation of optimized, executable code tailored to the deployment platform.

Code generation within the Model-Driven Architecture (MDA) enables the efficient conversion of the Platform-Specific Model (PSM) into executable source code for the target platform. This automated process ensures consistency, quality, and efficiency in the generated code, while also facilitating the maintenance and evolution of the software application.

Model transformation is a key step in the Model-Driven Architecture (MDA) software development process. By converting models from one level of abstraction to another, this transformation helps create more detailed or platform-specific representations of the software application, ensuring consistency and quality in the results [22].

The MDA approach relies on the creation of platform-independent models (PIM) as a starting point, allowing the separation of design from technical implementation details. The PIMs are then transformed into platform-specific models (PSM), which allows the same model to be adapted to different technologies. This transformation can lead to automatic code generation, accelerating the development process and facilitating portability across various technical environments [23, 24].

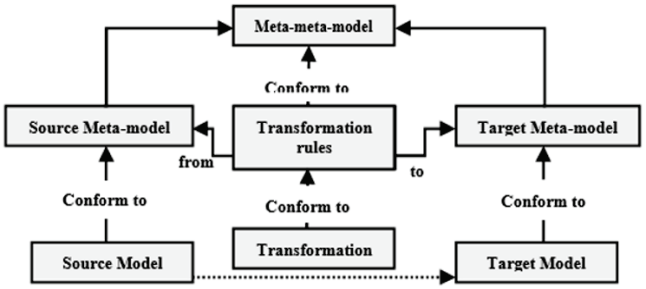

Figure 3 MDA Transformation process

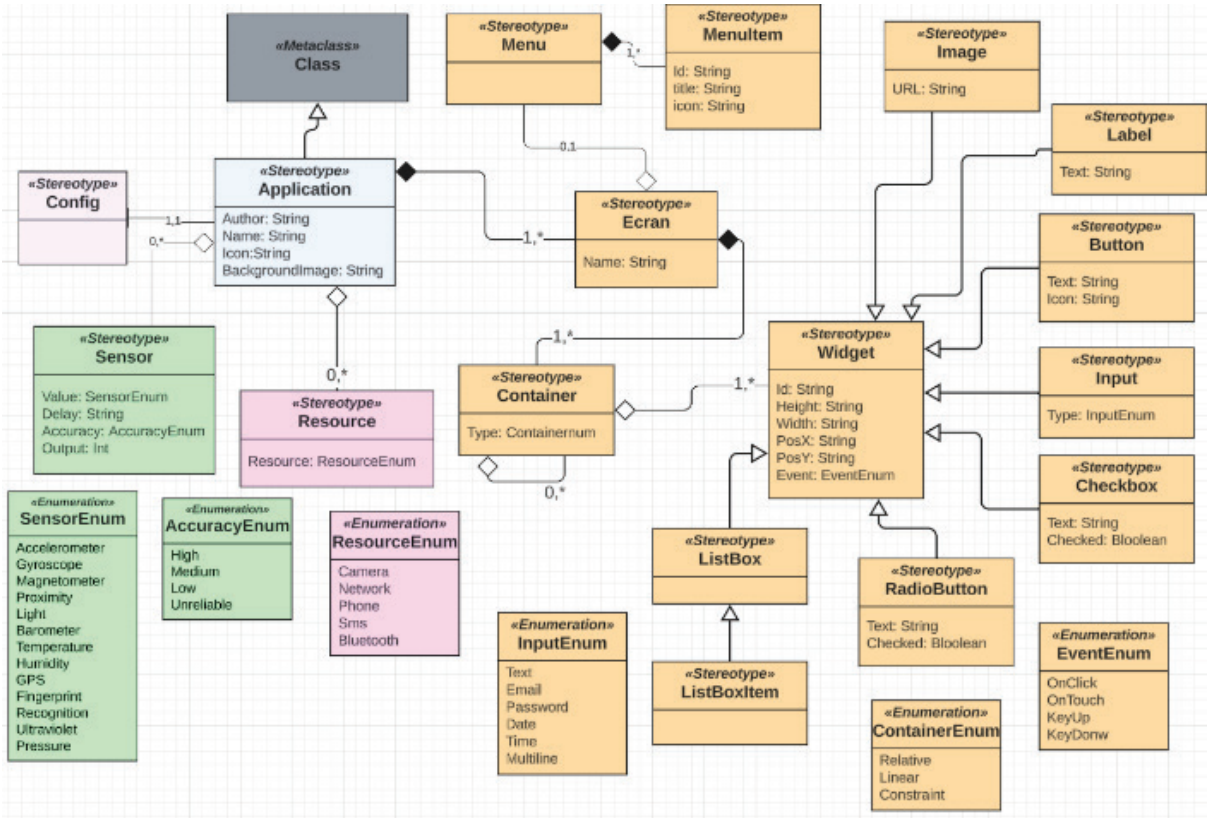

Figure 4 Proposed source metamodel

3.2 Source Meta-model

This metamodel, implemented as a UML profile, captures the essential elements required for mobile application development. UML (Unified Modeling Language) is widely recognized and supported across various development tools, making it an ideal choice for modeling complex systems in a standardized manner.

At the core of the UML profile, the Application entity is defined with attributes such as author, name, and background image, which represent general information about the mobile application. The Screens (or pages) within the application contain Containers (Layouts) and Widgets, which are crucial for creating a flexible user interface. The Widgets are interactive elements such as buttons, labels, checkboxes, input fields, and lists (ListBox), each with specific attributes like ID, size, position, and associated events (e.g., onClick or onTouch). These elements correspond to essential UI components in Android, such as Button, TextView, EditText, and Spinner.

Justification of Key Metamodel Elements:

- Menus and MenuItems: Menus structure the application's navigation, enhancing user experience by allowing easy access to various functionalities. These elements correspond to Android's Menu and MenuItem components.
- Input Types and Layouts: Enumerations like InputEnum define input types (e.g., text, email, password) and ContainerEnum for layout types (e.g., relative, linear, constraint). This is critical for adapting UI elements to specific Android layouts, ensuring a flexible and adaptable design.
- Resources and Sensors: To support modern mobile app functionality, the metamodel incorporates resources such

as email, messaging, and camera, as well as Embedded Sensors like GPS, accelerometers, and proximity sensors. These elements facilitate interaction with the physical environment, enabling the development of applications that are context-aware and responsive to real-world changes.

The Tab. 1 summarizes key elements of the metamodel and their Android equivalents, providing an overview of how each component maps to the Android platform.

Table 1 Mapping between PIM and Android Code

Metamodel Element	Description	Android Equivalent
Application	General app information	N/A
Screen	UI Screen or Page	Activity/Fragment
Container	Layout for UI Elements	LinearLayout, RelativeLayout, ConstraintLayout
Widget - Button	Interactive button	Button
Widget - Label	Display text	TextView
Widget - Input	User input field	EditText
ListBox	Dropdown list	Spinner
Menu/MenuItem	Navigation structure	Menu, MenuItem
Sensor	Device sensor integration	SensorManager
Resource	Device functionality (e.g., camera)	Camera, TelephonyManager

3.3 PIM to PSM Transformation Process

The transformation from Platform Independent Model (PIM) to Platform Specific Model (PSM) is a critical step in the MDA approach. This process involves mapping abstract, platform-agnostic elements from the PIM to platform-specific components defined in the PSM, tailored to Android in our case.

- Modeling the Application in PIM: We start by defining the UI elements and application structure in the PIM, conforming to the source metamodel. Elements such as Screens, Containers, and Widgets are instantiated with specific attributes and relationships, creating a high-level representation of the application's structure.
- Transformation Rules with Acceleo: Using Acceleo, we define transformation rules that map each PIM element to its Android equivalent. For example:

Container elements in the PIM map to Android Layouts such as LinearLayout or ConstraintLayout.

Button widgets transform to Android's Button components with attributes like text, onClick, and layout properties.

Input Fields are translated to EditText with appropriate input types (e.g., text, password) from the InputEnum.

Acceleo templates define these rules, ensuring that attributes like ID, size, and event handling are faithfully converted to platform-specific implementations.

Generation of Platform-Specific Code: The PSM is processed by Acceleo's M2T (model-to-text) transformation capabilities, generating the final Android XML layout files and Java classes. The transformation ensures fidelity to the Android platform while maintaining alignment with the original model design. Generated files, such as AndroidManifest.xml and the XML layout files, are structured according to Android's requirements, providing a ready-to-use code base for the application.

Fig. 5 is a flowchart illustrating the steps from metamodel creation to code generation.

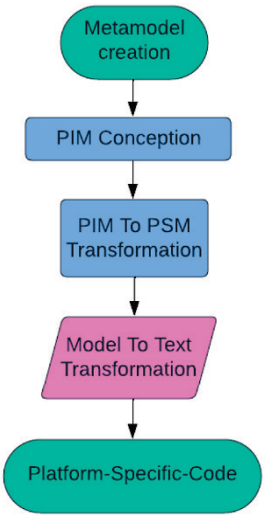

Figure 5 steps of platform-specific-code generation

3.4 Acceleo Tool

Acceleo is an open-source code generation tool within the Eclipse ecosystem, primarily used in the Model-Driven Architecture (MDA) approach to generate source code from models. It is based on the MOF Model to Text Transformation (MTL) standard from the Object Management Group (OMG), making it compatible with other UML-based modeling tools.

Key Features of Acceleo:

- Code Generation Language: Acceleo uses its own transformation language, inspired by the Object Constraint Language (OCL), allowing complex logic and expressions to define rules for generating code from models.
- Eclipse Integration: Acceleo is integrated with the Eclipse environment, making it easy to use with other modeling tools like the Eclipse Modeling Framework (EMF) and Papyrus for UML.
- Support for Multiple Languages: While often used for Java code generation, Acceleo can generate code in other languages such as C++, Python, or PHP, depending on project needs.
- EMF Model Support: Acceleo works with EMF models, making it ideal for code generation in the MDA context.

Acceleo generates source code from PSMs (Platform-Specific Models) based on predefined rules, enabling quick implementation of software systems from models. It allows customization of the generated code through its transformation language, letting developers create templates and rules for flexible code generation. Acceleo can also be integrated into automated development processes, such as continuous integration pipelines, to automatically generate code from updated models.

Advantages of Acceleo for MDA:

- Reduced Development Time: By generating code from models, Acceleo shortens the time from design to implementation.
- Flexibility and Customization: Its transformation language enables highly customizable code generation, adaptable to various platforms and technologies.
- Eclipse Ecosystem Integration: Being part of Eclipse, Acceleo works seamlessly with other modeling tools, supporting consistent MDA workflows.
- Support for Multiple Programming Languages: Acceleo's ability to generate code in different languages makes it versatile for various project types.

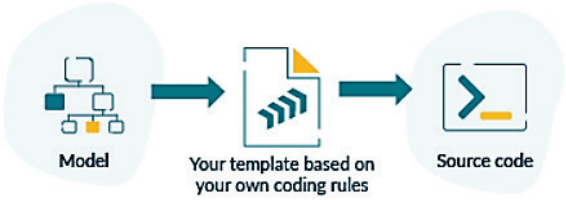

Figure 6 Acceleo process

Using Acceleo, our solution consists of three steps. Firstly, we develop a platform-independent-model conform to our source meta-model. This Platform-independent-model models the UI component of an application. Then, we create an Acceleo template, which defines how the target code should be generated from the input platform-independent-model. Finally, Acceleo processes the platform-independent-model with the template created, generating the platform-specific code. The following figure shows the Acceleo process according to Eclipse Foundation, 2021.

3.5 Results and Discussion

In this section, we will present a step-by-step demonstration of our solution using a case study. For this case study, we have selected an Android application that allows adding a new book to the library database. This application will consist of two graphical interfaces: the first will be dedicated to authentication, and the second will serve for adding a new book to the database. The latter will include a title, a label and a text field for the book's name, a label and a text field for the author's name, a label and a text field for the publisher's name, and a label and a number field for the number of copies. It will also contain two buttons to either submit or cancel the form. Based on this test case description, the platform-independent models must conform to the previously presented metamodel. Then, each element will be transformed into its equivalent in the Android-specific models using the ATL language, and finally, source code will be generated using Acceleo templates.

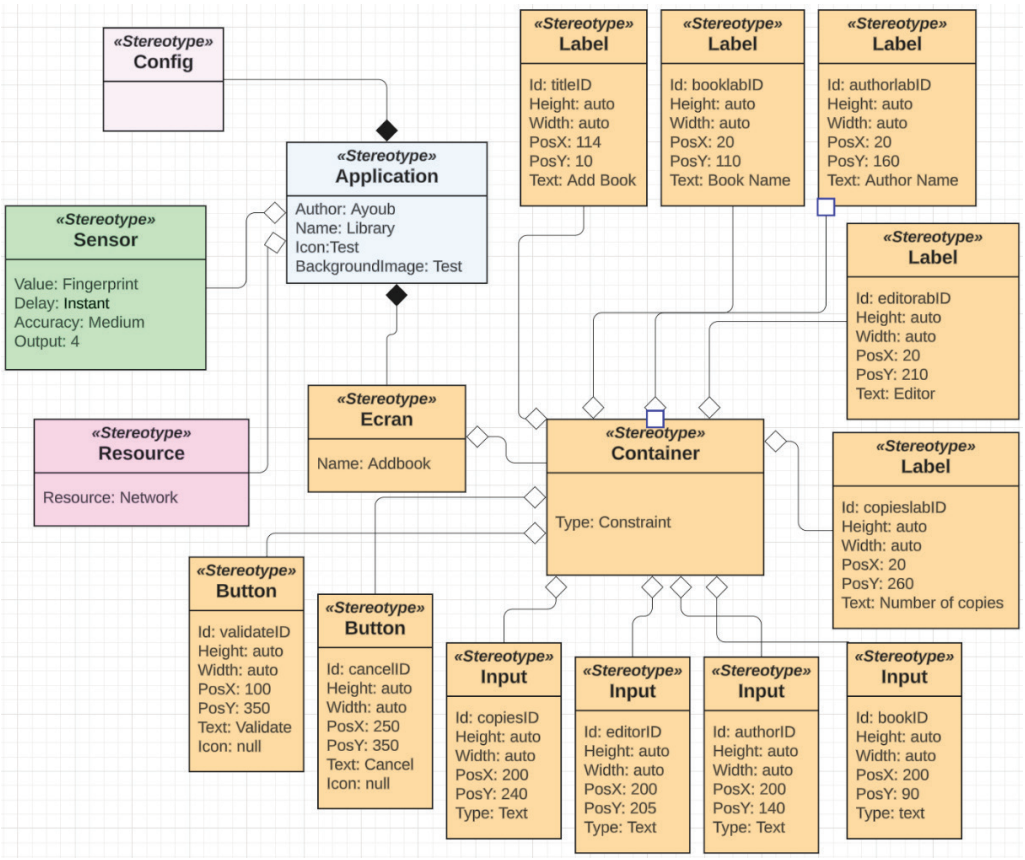

Figure 7 PIM of Add book interface

The platform-independent model of this interface will consist of a title, a label and a text field for the book's name, a label and a text field for the author's name, a label and a text field for the publisher's name, as well as a label and a number

field for the number of copies. It also includes two buttons to either submit or cancel the form. The following figure represents the platform-independent model incorporating the aforementioned user interface elements to create this Android interface for adding a new book. This platform-independent model conforms to the previously presented metamodel.

The following model represents the PSM (Platform-Specific Model) of the Add book interface for the Android platform, which was generated from the PIM (Platform Independent Model) shown above through a transformation performed using the ATL (Atlas Transformation Language). Here is a detailed description of the elements that make up this model.

Using an Acceleo template, we generate the AndroidManifest file, which is the configuration file of the Android application. We also generate the view XML file by transforming each component of this platform-independent-model to each equivalent in the android user interface.

The code is written in Acceleo Query Language, which allows navigating and querying input model within templates defined by Acceleo (Eclipse Foundation, 2021). As we mentioned above, Acceleo is a powerful tool for generating code from models, streamlining development processes, and ensuring consistency across projects, based on templates. The template takes the platform-independent-model as input. It transforms each element and its attributes to their android equivalent component. For example, this template transforms the label element to the TextView component, it transforms the width attribute in the PIM to the layout width in the Android code. It also affects the values of each element' attribute to its Android equivalent.

The following Fig. 9 shows a part of the template code.

Once the template is executed, two files are generated, the AndroidManifest and the XML view files, following the transformation rules defined in the Acceleo template as it shown in the figures above. This file represents a model-to-text transformation between Configuration class modeled in the platform-independent-model and the AndroidManifest.xml file with its contents as it shown in the Fig. 11.

The template also generates the XML file of Android UI. The following Fig. 10 shows an extract of the XML view file containing android component generated from the platform-independent-model presented above such as: the constraint layout which corresponds to the container in the PIM, the EditText which corresponds to the text Input in the PIM, etc.

In order to visualize our solution result, we create an application on Android Studio. We drug & drop these two files: android manifest and xml view file in their corresponding folders, before running the application to visualize the result. The following Fig. 12 shows the result interface containing elements modeled in the platform-independent-model and transformed to android code using the Acceleo template.

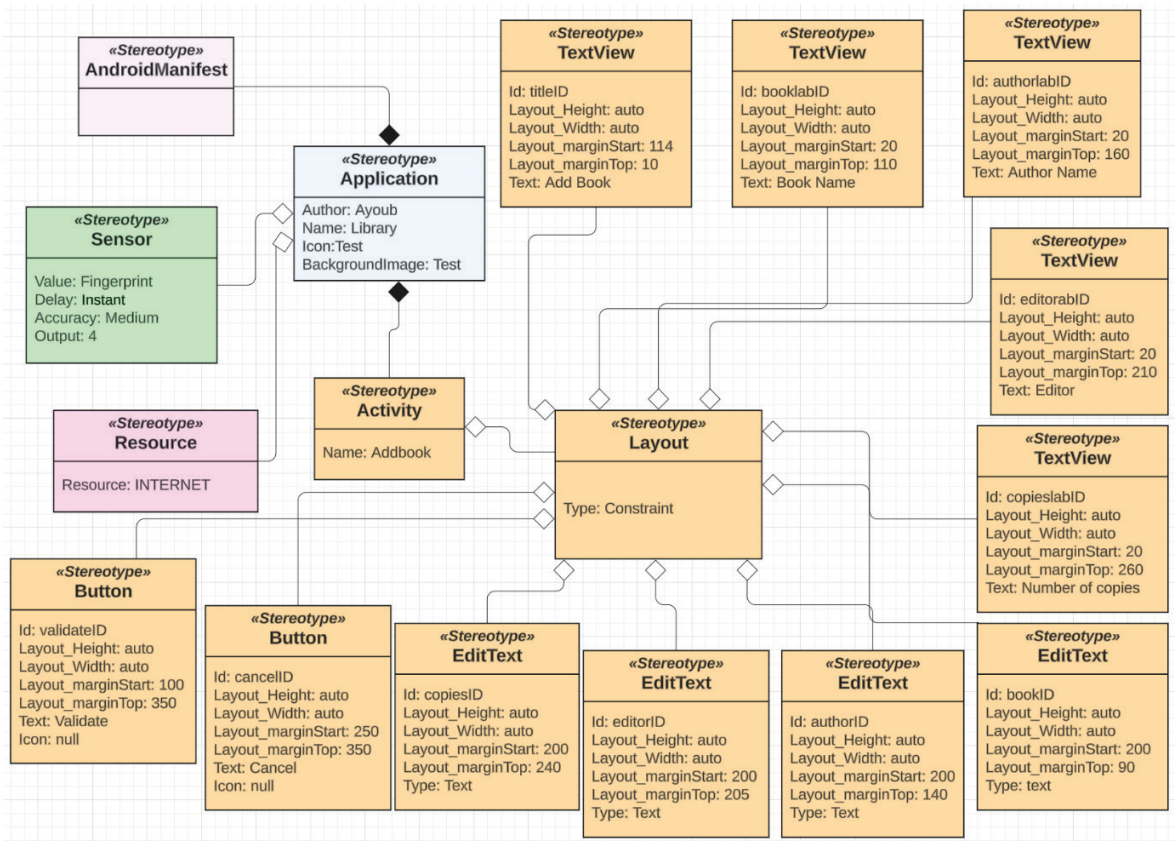

Figure 8 PSM generated from the PIM

generate.mtl × AndroidManifest.xml activity_main.xml class diagram

```
[/if]
[if (anEClass.name.equalsIgnoreCase('Label'))]
    [file ('activity_main.xml', true, 'UTF-8')]
<TextView
    android:id="@+id/[anEClass.eAllAttributes->at(1).defaultValue.toString()/]"
    android:layout_width="[anEClass.eAllAttributes->at(2).defaultValue.toString()/]"
    android:layout_height="[anEClass.eAllAttributes->at(3).defaultValue.toString()/]"
    android:text="[anEClass.eAllAttributes->at(4).defaultValue.toString()/]"
    android:layout_marginStart="[anEClass.eAllAttributes->at(5).defaultValue.toString()/]dp"
    android:layout_marginTop="[anEClass.eAllAttributes->at(6).defaultValue.toString()/]dp" />
    [/file]
[/if]
[if (anEClass.name.equalsIgnoreCase('Input'))]
    [file ('activity_main.xml', true, 'UTF-8')]
<EditText
    android:id="@+id/[anEClass.eAllAttributes->at(1).defaultValue.toString()/]"
    android:layout_width="[anEClass.eAllAttributes->at(2).defaultValue.toString()/]"
    android:layout_height="[anEClass.eAllAttributes->at(3).defaultValue.toString()/]"
    android:layout_marginStart="[anEClass.eAllAttributes->at(6).defaultValue.toString()/]dp"
    android:layout_marginTop="[anEClass.eAllAttributes->at(7).defaultValue.toString()/]dp"
    android:inputType="[anEClass.eAllAttributes->at(5).defaultValue.toString()/]"
    android:text="[anEClass.eAllAttributes->at(4).defaultValue.toString()/]" />
    [/file]
[/if]
[if (anEClass.name.equalsIgnoreCase('ListBox'))]
    [file ('activity_main.xml', true, 'UTF-8')]
<Spinner
    android:id="@+id/[anEClass.eAllAttributes->at(1).defaultValue.toString()/]"
    android:layout_width="[anEClass.eAllAttributes->at(2).defaultValue.toString()/]"
    android:layout_height="[anEClass.eAllAttributes->at(3).defaultValue.toString()/]"
    android:layout_marginStart="[anEClass.eAllAttributes->at(4).defaultValue.toString()/]dp"
    android:layout_marginTop="[anEClass.eAllAttributes->at(5).defaultValue.toString()/]dp" />
    [/file]
[/if]
[if (anEClass.name.equalsIgnoreCase('Button'))]
    [file ('activity_main.xml', true, 'UTF-8')]
<Button
    android:id="@+id/[anEClass.eAllAttributes->at(1).defaultValue.toString()/]"
    android:layout_width="[anEClass.eAllAttributes->at(2).defaultValue.toString()/]"
```

Figure 9 Template code

activity_main.xml × generate.mtl AndroidManifest.xml class diagram

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.coordinatorlayout.widget.CoordinatorLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:fitsSystemWindows="true"
    tools:context=".MainActivity">

<androidx.constraintlayout.widget.ConstraintLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent">

<TextView
    android:id="@+id/titleid"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Add Book"
    android:layout_marginStart="114dp"
    android:layout_marginTop="10dp"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent" />

<EditText
    android:id="@+id/booknamevalue"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginStart="200dp"
    android:layout_marginTop="90dp"
    android:inputType="text"
    android:text="Book name"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent" />

<TextView
    android:id="@+id/bookname"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Book name"
```

Figure 10 Generated UI code

AndroidManifest.xml × activity_main.xml class diagram

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
        xmlns:tools="http://schemas.android.com/tools">
    <application
        android:allowBackup="true"
        android:dataExtractionRules="@xml/data_extraction_rules"
        android:fullBackupContent="@xml/backup_rules"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/Theme.Test"
        tools:targetApi="31">
        <activity
            android:name=".MainActivity"
            android:exported="false"
            android:label="@string/title_activity_main"
            android:theme="@style/Theme.Test" />
         </application>

</manifest>

```

Figure 11 Generated Manifest

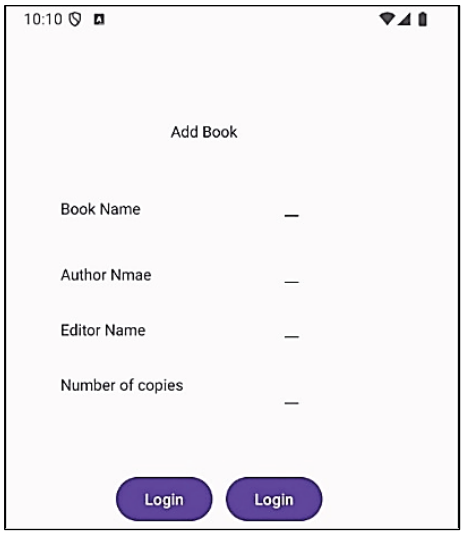

Figure 12 Android interface

This case study uses a PIM composed of multiples mobile components such as, View, Configuration, Container, Label, button, ListBox, input text and input number. This PIM respects our proposed source metamodel. Using Acceleo tool, we generate an AndroidManifest file, which is the configuration file of android applications and constitutes a model-to-text transformation between configuration class and the generated manifest file. We also generate an Android view code by transforming each component in the PIM to its equivalent android component. The layout tag refers to container class, TextView tags refer to Label classes, EditText tags refer to input classes, Spinner tag refers to ListBox class, and Button tags refers to Button classes. Until now, our solution allows to generate UI interface and not a mobile application that can be executed in different platforms, which needs some manual adaptations to create a mobile application and drug & drop the result code in the appropriate folder of the mobile application. In our future works, we will expand our source meta-model to respect mobile app creation with different folders.

3.6 Analysis of Generated Code

After executing the Acceleo template on the Platform-Independent Model (PIM), the generated code includes the AndroidManifest and XML layout files. Below is an in-depth analysis of the code:

- AndroidManifest.xml: This file specifies essential application metadata, permissions, and components, including Activities. The generated AndroidManifest aligns well with Android standards, specifying the main activity for launching the application and permissions for network access if specified in the PIM. The automatic generation of this file ensures consistency, as updates to the PIM automatically reflect changes here. However, additional permissions and services, such as location services or background processing, may require manual inclusion as these aspects are not fully automated in the current version of our model.
- XML Layout Files: The XML layout accurately represents the user interface described in the PIM. Each component is mapped to its Android equivalent:

Container to Layout Mapping: Containers defined in the PIM translate directly to layout components such as ConstraintLayout or LinearLayout in Android.

Widget to View Mapping: Each Widget (e.g., Labels, Buttons) is generated as its corresponding Android component (e.g., TextView, Button, EditText). Attributes like text, layout_width, layout_height, and event listeners such as onClick are defined according to the template's transformation rules.

The quality of the generated code is high, with a clear, organized structure that adheres to Android's recommended layout hierarchy. However, the generated code currently only supports basic UI components and lacks some advanced features like nested views or custom styles. Developers may need to make manual adjustments for more complex layouts or additional styling.

3.7 Comparison with Traditional Development Methods

Table 2 MDA vs Tradition development methods

Aspect	Traditional Development	Model-Driven Approach
Development Time	Requires manual coding of UI components, configuration, and repetitive tasks for layout and manifest files.	Significantly reduced due to automated generation from PIM, especially for simple UIs.
Consistency	High risk of inconsistencies due to manual updates across multiple files.	Maintained across the application since modifications are made directly in the PIM.
Adaptability	Changes in design require updates in multiple places (XML, Java/Kotlin files).	Adaptations are easier as updates to the PIM are automatically reflected in generated files.
Learning Curve	Requires knowledge of Android development tools and languages like Java/Kotlin.	Lower learning curve with Acceleo, especially for developers familiar with UML and MDA concepts.
Scalability	Manual coding becomes more time-consuming and error-prone as application complexity increases.	Model-driven approach may face challenges with complex applications due to limitations in model representation.

The model-driven approach generally accelerates the development process for basic UIs, saving time on manual coding and minimizing human error. However, for complex

applications with custom animations, extensive styling, or device-specific features, the model-driven approach may require substantial template customization or manual adjustments post-generation.

3.8 Challenges and Limitations with Complex Applications

While this approach is effective for simple to moderately complex applications, applying it to more sophisticated mobile applications present several challenges:

- Advanced UI Components: For complex user interfaces involving custom views, animations, and dynamic content, the current model may lack sufficient granularity. Extending the metamodel to support these components could make the process cumbersome and reduce template reusability.
- Integration with Native Android Features: Certain Android-specific features, such as fragments, services, and broadcast receivers, are not fully automated in the current version of the metamodel. Customizing the metamodel and transformation templates to support these features would increase the initial setup time and require in-depth knowledge of both Android and Acceleo's transformation language.
- Performance Optimization: Generated code may not always be optimized for performance, especially in terms of layout complexity and memory usage. In traditional development, developers can manually refine layout hierarchies to reduce memory footprint, which might not be as straightforward in a generated environment.
- Maintenance Over Time: As Android evolves, the model-driven approach needs frequent updates to the metamodel and transformation templates to stay compatible with the latest APIs and best practices. This adds a layer of ongoing maintenance that must be managed alongside the application development lifecycle.

4 CONCLUSION AND PERSPECTIVES

In this paper, we introduced an approach based on Model-Driven Architecture (MDA) to automatically generate native source code for mobile applications, beginning with a Platform Independent Model (PIM). We developed a UML metamodel that encapsulates key mobile application elements, ensuring that PIMs remain consistent and reusable across projects. By utilizing Acceleo, a model-to-text transformation tool, we were able to generate Android user interface code directly from the PIM. Through a case study, we demonstrated how this approach can streamline mobile development, reducing both the cost and time required compared to traditional platform-specific coding.

Our case study focused on an Android interface for adding a new book to a library database, where a PIM adhering to the metamodel was used as input for an Acceleo template to generate corresponding Android source code. By bypassing the typical intermediate transformation to a Platform-Specific Model (PSM), our approach simplifies the development process, reducing both effort and potential for error.

Broader Impact on Mobile Development Practices. This approach has the potential to significantly impact mobile development practices, especially in environments that require multi-platform support and rapid development cycles. By abstracting the application design process from platform-specific details, developers can focus more on the core functionality and user experience, rather than on managing multiple codebases. This abstraction can streamline the onboarding process for developers and enhance collaboration within teams, as the PIM provides a unified framework that aligns with widely understood UML standards.

Moreover, as mobile applications continue to evolve to incorporate more complex features—such as advanced sensors, machine learning integrations, and real-time data processing—automating code generation can help developers manage the increasing complexity. The MDA approach allows for rapid prototyping and iteration, as modifications to the PIM can be propagated through to the generated code, ensuring alignment between high-level design and implementation. This capability could lead to faster time-to-market for mobile applications, as well as improved code quality through standardized model-driven practices.

Planned Extensions to the Metamodel and Code Generation Capabilities. To further enhance the versatility and effectiveness of our MDA approach, we plan to extend the current metamodel to include additional features:

- Device-Specific API Access: To maximize cross-platform functionality, we aim to provide built-in support for accessing device-specific APIs, including camera functionality, storage access, and network communication. By defining these features within the metamodel, the PIM can be transformed to include calls to platform-native APIs, such as Android's Camera or iOS's CoreLocation. This will facilitate seamless cross-platform code generation, enhancing the usability of generated applications across different devices.
- Business Logic and Data Management: Currently, the focus is on generating user interface components. Future iterations will include business logic elements and data management structures within the metamodel. For instance, we will enable the generation of basic CRUD operations and database interactions, which will support applications with more complex data needs. Our goal is to eventually provide end-to-end code generation, covering not only UI but also the underlying logic and data layers, supporting platforms such as iOS and Windows.
- Multi-Platform Support: In addition to Android, we plan to extend support to other platforms like iOS and Windows. By generalizing the transformation rules and utilizing platform-specific templates, the PIM can be translated into PSMs that are compatible with multiple operating systems. This extension will allow developers to leverage a single metamodel to generate fully functional applications across a variety of platforms with minimal manual adjustments.

These planned enhancements aim to expand the scope and applicability of the MDA approach, enabling developers to create richer, more sophisticated mobile applications with reduced manual effort. By continuing to refine and extend the metamodel, our approach will support a broader range of mobile development needs, making model-driven development a practical and efficient choice for modern application creation.

5 REFERENCES

[1] Castelain, Q. (2015). Elaboration d'un outil d'analyse d'alignement IT-Business adapté aux spécificités des PMEs. (in France)

[2] Štimac, H., Vajda, T. & Franjkovic, J. (2019). Lifecycle of Smartphones: Brand Representation and Their Marketing Strategy.

[3] Market Share: Smartphones, Worldwide, 4Q07 and 2007. https://www.gartner.com/en/documents/619509 Accessed: July 19, 2024.

[4] Smartphone sales worldwide 2007-2023 | Statista. https://www.statista.com/statistics/263437/global-smartphone-sales-to-end-users-since-2007/ Accessed: July 19, 2024.

[5] Hui, N. M., Chieng, L. B., Ting, W. Y., Mohamed, H. H. & Hj Mohd Arshad, M. R. (2013). Cross-platform mobile applications for android and iOS. *The 6th Joint IFIP Wireless and Mobile Networking Conference (WMNC2013)*, 1-4. https://doi.org/10.1109/WMNC.2013.6548969

[6] Dickson, P. E. (2012). Cabana: a cross-platform mobile development system. *Proceedings of the 43rd ACM technical symposium on Computer Science Education, in SIGCSE'12*. New York, NY, USA: Association for Computing Machinery, 529-534. https://doi.org/10.1145/2157136.2157290

[7] Truyen, F. (2006). *The fast guide to model driven architecture the basics of model driven architecture*. Cephas Consulting Corp.

[8] Prabowo, Y. D., Kristijantoro, A. I., Warnars, H. & Budiharto, W. (2021). Systematic literature review on abstractive text summarization using Kitchenham method. *ICIC Express Letters, Part B: Applications, 12*(11), 1075-1080.

[9] Luque, S., Maréchal, D. & de Thomas Sanz, C. (2014). Modélisation des compartiments phyto-écologiques des milieux ouverts d'altitude par la théorie des graphes et les modèles de distribution d'espèces. (in France)

[10] Sabraoui, A., El Koutbi, M. & Khriss, I. (2012). GUI code generation for android applications using a MDA approach. *IEEE International Conference on Complex Systems (ICCS2012)*, 1-6. https://doi.org/10.1109/ICoCS.2012.6458567

[11] Lachgar, M. & Abdali, A. (2014). Generating Android graphical user interfaces using an MDA approach. *Third IEEE International Colloquium in Information Science and Technology (CIST2014)*, 80-85. https://doi.org/10.1109/CIST.2014.7016598

[12] Tufail, H., Azam, F., Anwar, M. W. & Qasim, I. (2018). Model-driven development of mobile applications: A systematic literature review. *The 9th IEEE Annual Information Technology, Electronics and Mobile Communication Conference (IEMCON2018)*, 1165-1171. https://doi.org/10.1109/IEMCON.2018.8614821

[13] Benouda, H., Azizi, M., Esbai, R. & Moussaoui, M. (2016). MDA approach to automate code generation for mobile applications. *Mobile and Wireless Technologies 2016*, Springer, 241-250. https://doi.org/10.1007/978-981-10-1409-3_27

[14] Benouda, H., Azizi, M., Moussaoui, M. & Esbai, R. (2017). Automatic code generation within MDA approach for cross-platform mobiles apps. *First IEEE International Conference on Embedded & Distributed Systems (EDiS2017)*, 1-5. https://doi.org/10.1109/EDIS.2017.8284045

[15] Parada. A. G. & De Brisolara, L. B. (2012). A model driven approach for android applications development. *IEEE Brazilian Symposium on Computing System Engineering 2012*, 192-197. https://doi.org/10.1109/SBESC.2012.44

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

[17] Korchi, A., Khachouch, M. K. & Lakhrissi, Y. (2024). A Model-Driven Architecture Solution for Multi-Platform Mobile App Development. *Journal of System and Management Sciences, 14*(5), 1-13. https://doi.org/10.33168/JSMS.2024.0501

[18] Bali, S. et al. (2023). A framework to assess the smartphone buying behaviour using DEMATEL method in the Indian context. *Ain Shams Engineering Journal*. https://doi.org/10.1016/j.asej.2023.102129

[19] Lachgar, M. & Abdali, A. (2014). Modélisation et Génération des Interfaces Graphiques Android et JSF 2.2 selon une approche MDA. (in France)

[20] Korchi, A., Khachouch, M. K., Lakhrissi, Y., Marzouki, N. E., Moumen, A. & Mohajir, M. E. (2024). Classification of existing mobile cross-platform approaches and proposal of decision support criteria. *International Journal of Information and Communication Technology, 24*(1), 86-111. https://doi.org/10.1504/IJICT.2024.135305

[21] El-Kassas, Wafaa S. et al. (2017). Taxonomy of Cross-Platform Mobile Applications Development Approaches. *Ain Shams Engineering Journal, 8*(2), 163-190. https://doi.org/10.1016/j.asej.2015.08.004

[22] Korchi, A., Khachouch, M. K., Lakhrissi, Y. & Moumen, A. (2020). Classification of existing mobile cross-platform approaches. *International Conference on Electrical, Communication, and Computer Engineering (ICECCE2020)*, Istanbul, Turkey, 2020, 1-5. https://doi.org/10.1109/ICECCE49384.2020.9179222

[23] Khachouch, M. K., Korchi, A., Lakhrissi, Y. & Moumen, A. (2020). Framework Choice Criteria for Mobile Application Development. *International Conference on Electrical, Communication, and Computer Engineering (ICECCE2020)*, 1-5. https://doi.org/10.1109/ICECCE49384.2020.9179434

[24] Lachgar, M. & Abdali, A. (2015). DSL and code generator for accelerating iOs apps development. *Third IEEE World Conference on Complex Systems (WCCS2015)*, 1-8. https://doi.org/10.1109/ICoCS.2015.7483269

Authors' contacts:

Ayoub Korchi
(Corresponding author)
SIGER Laboratory, Faculty of Science and Technology,
Sidi Mohamed Ben Abdellah University,
Fez, 36000, Morocco
ayoub.korchi@usmba.ac.ma

Mohamed Karim Khachouch
SIGER Laboratory, Faculty of Science and Technology,
Sidi Mohamed Ben Abdellah University,
Fez, 36000, Morocco
mohamedkarim.khachouch@usmba.ac.ma

Younes Lakhrisi
SIGER Laboratory, Faculty of Science and Technology,
Sidi Mohamed Ben Abdellah University,
Fez, 36000, Morocco
younes.lakhrissi@usmba.ac.ma