ISSN 1816-6075 (Print), 1818-0523 (Online)
Journal of System and Management Sciences
Vol. 14 (2024) No. 5, pp. 1-13
DOI:10.33168/JSMS.2024.0501

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

Ayoub Korchi, Mohamed Karim Khachouch, Younes Lakhrissi

SIGER Laboratory, Faculty of Science and Technology, Sidi Mohamed Ben Abdellah University, Fez, 36000, Morocco

ayoub.korchi@usmba.ac.ma, mohamedkarim.khachouch@usmba.ac.ma, younes.lakhrissi@usmba.ac.ma

Abstract. This paper proposes an automatic code generation approach for multi-platform mobile apps based on the Model Driven Architecture (MDA) methodology. A UML-based metamodel encompassing common mobile application constructs like views, controls, resources, and events is presented. The Platform Independent Model (PIM) conforming to this metamodel can be transformed into platform-specific source code using the Acceleo code generator. The approach aims to reduce cost and effort compared to native development for each platform since it allows to develop once and deploy in each platform. A case study demonstrates Android user interface generation from a sample PIM model. Further enhancements like supporting more target platforms, integration of sensors and advanced UI elements have been identified.

Keywords: MDA, Cross-platform, Mobile-development

1. Introduction

Multi-platform mobile development constitutes a real challenge for application providers due to the diversity of devices, operating systems, and development environments. One of the most serious obstacles is to develop an application that respects the unique design principles, user experiences, and performance capabilities of existing mobile platforms. Each mobile platform Android, iOS, Windows Phone, and potentially others have its own set of tools, APIs, and development languages, requiring developers to have experiences on multiple codebases and adapt to varying tools and frameworks to develop a native application for each platform, following a normal development cycle.

Maintaining synchronization between updates, features, and bug fixes across different platforms further amplifies the complexity. Moreover, to optimize apps to run on different mobile platforms adds another layer of intricacy. These challenges lead to the existence of cross-platform concept which allows to develop once and deploy everywhere. This concept helps to reduce effort and development cycle time.

The following figure compare the difference between normal development cycle and cross-platform cycle that we have just talked about.

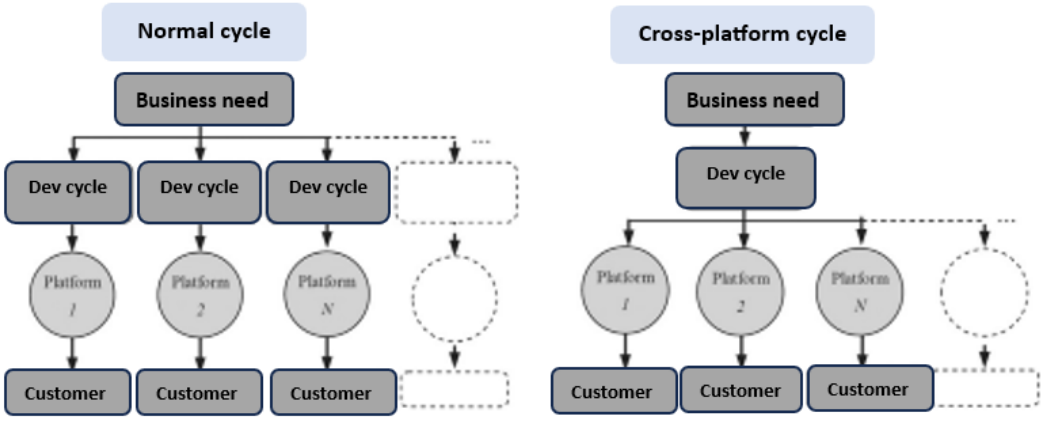

Fig. 1: Different development cycles

Since its apparition, many approaches have been proposed to serve the cross-platform purpose such as: compilation, component-based, interpretation, cloud-based, merged, and Model-Driven Architecture approaches. Each one of them presents several advantages, but almost all of them are limited by technical aspects and specifications that needs a lot of learning efforts, high quality resources and consequently increases time and cost of the project as we presented in our previous works (Korchi et al. 2020; Khachouch et al. 2020). However, the Model Driven Architecture (MDA) remains the only approach not really limited by technical specifications since it focuses on abstract models and automated transformations. MDA approach aims to address this limitation by emphasizing the creation of a platform-independent-model (PIM) that facilitates more systematic and consistent code generation across multiple platforms.

Our solution is based on Model-Driven Architecture approach benefiting from its advantages, since it allows to facilitate the development process by emphasizing models as primary artifacts throughout the development lifecycle. Many researches have been done based on the same solution emphasizing the importance of creating high-level models, using automated transformations to generate platform-specific code, and highlighting the benefits of abstraction, reusability, and maintainability achieved in mobile app development. Most of these research's lack of simplicity since they propose manual model-to-model transformation or use some additional APIs for this transformation which add more effort to generate the result code. On the other hand, some researchers focus on simplicity without limiting the metamodel with only mobile components which could generate some dumps. Our solution consists of generating platform-specific code directly from a platform-independent-model (PIM) which allows to skip model-to-model transformation between the platform-independent-model and the platform-

specific-model. Our solution uses the Acceleo generator tool that transform each PIM component to its equivalent in platform-specific code using templates that contains transformation rules. Our solution consists also of limiting components that could be used in the PIM model by proposing a source metamodel that regroups a lot of mobile UI elements.

This article is structured as follow: Section 2 will be dedicated to literature review presenting some existing solutions and giving limits of each one of them. Section 2 will also give more details about the background of our works such OS and cross-platform approaches. Research methodology is presented in section 3 with 3 subsections. The first one gives a description of the MDA approach with its advantages, the second one presents our proposal metamodel, and the third one presents the Acceleo tool. In section 4, we present a case study demonstration of an Android user interface. We end up with a conclusion and some perspectives in section 5.

2. literature Review

Cross-platform solutions appeared due to the diversity among mobile operating systems, each with its specific tools, development languages, and IDEs. Android, iOS, and Windows Phone are the three key operating systems discussed in the context. Android dominates the mobile OS landscape, capturing an impressive 71.95% market share in 2023 with around 2.7 million apps available in Google Play, as reported by (Eralda et al. 2023). iOS follows with a 26.27% market share according to (Garg, 2021). Despite its lower market share, Windows Phone is noted for its exemplary user interface. Android's market dominance is attributed to various factors. These include the accessibility of applications in the Play Store with free downloads, a wide diversity of available applications, even beyond the confines of the Play Store, and other factors contributing to its widespread adoption and popularity among users. Table 1 regroups some parameters and specifications of each OS as a comparative study.

Table 1: Summary comparative of the OS

	Android	iOS	WP
FOUNDER	Google	Apple	Microsoft
KERNEL	Linux	Unix	Windows CE
DEV. LNAGUAGE	Java/Kotlin	Objective c/ Swift	C#
UI	XML Files	Cocoa touch	XAML Files
VIRTUAL MACHINE	Dalvik	---------	CLR
IDE	Android Studio	Xcode	Visual Studio
APP. MARKET	Play Store	App Store	WP Store
LAST VERION	Version 14	Version 16.6	Version 10
LAST UPDATE	2023	2023	2020
FOUNDER	Google	Apple	Microsoft
KERNEL	Linux	Unix	Windows CE

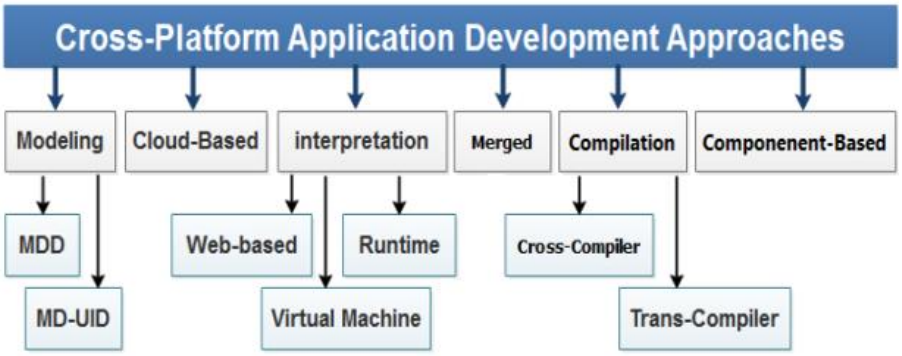

Fig. 2: Classification of the cross-platform approaches

The diverse nature of mobile platforms has prompted the development of cross-platform solutions, leading researchers to explore various approaches since 2008. In a study by (Korchi et al. 2024), six cross-platform approaches were classified: Compilation, Component-based, Interpretation, Cloud-based, Merged, and MDA approaches.

Compilation, interpretation, and component-based approaches primarily focus on code manipulation, allowing generation of code for multiple platforms from a codebase written in a specific programming language. In contrast, the MDA approach prioritizes creating models representing an essential phase before application development begins. The Cloud-based and Merged approaches, while viable, face limitations. The MDA approach stands out as it can generate applications accessible in offline mode, unlike the Cloud-based approach. Moreover, MDA-based development remains relatively simpler compared to the complexity of the Merged approach, which combines multiple cross-platform methodologies, demanding more time and effort for learning and development. Table 2 provides a comparative analysis, outlining the strengths and weaknesses of each cross-platform approach.

Table 2: Comparative summary of cross-platform approaches.

Approach	Pros	Cons
Compilation approach	Reuses existing code.	Focuses on common elements, API. Difficulties in mapping between source and target languages.
Component-based approach	Simplifies the support of the new platforms.	Difficult task for developer who must learn how to use the component interfaces. Focuses on common functions belonging to all supported platforms.
Interpretation approach	Easy to use as an approach. Low storage. Lower downloads time.	Miss Look and feel. Performance degradation. Absence of virtual machine in some platforms.
Cloud-based approach	Storage. Application backend stored in server. Computation and processing executed in a distant server.	High speed network.
Merged approach	Combines two or more approaches (Ettifouri et al., 2017).	Learning efforts.
MDA approach	Reduces time cost. Reduces time. Increases productivity. Based on models. No interest to technical aspects.	Metamodel for complicated interfaces. Metamodel for some new aspects such as sensors especially the newest ones.

The initial five approaches have drawbacks related to technical code aspects, such as mapping between codes, performance degradation due to additional layers or virtual machines, and focusing on common elements across different platforms. The MDA approach circumvents these issues, emphasizing its primary advantage of not being affected by such technical implementation problems (Korchi et al., 2024). This predominant advantage serves as a strong motivation for us to select this approach as the foundation for our work.

Numerous researchers have explored MDA approaches in their work. We aim to present these

solutions chronologically, outlining the weaknesses of each to underscore the rationale behind our proposed solution.

Authors in (Sabraoui et al., 2012), present a methodology involving three key steps. Initially, they create the application's GUI within a Platform-Independent Model (PIM) using UML object diagrams aligned with a suggested meta-model for the mobile application. Following this, they convert this model into a simplified XMI file utilizing the JDOM API. Subsequently, they transform it into a Platform-Specific Model (PSM) using platform-specific meta-models for mapping, ATL languages for rule definition, ultimately leading to the generation of the GUI code. However, this approach incorporates an additional API for model-to-model transformation between PIM and PSM, necessitating more time and adaptations which constitute the main limitation of this solution.

In (Lachgar & Abdali, 2014), authors implement the MDA approach to create graphical user interfaces. They start by employing a GUI Domain-Specific Language (DSL) model as the Platform-Independent Model (PIM), developed using the Xtext solution. They then proceed with M2M transformation from the DSL Platform-Independent-Model to a Platform-Specific-Model (PSM), adhering to predefined generation rules. Finally, they generate source code by employing M2T transformation with a template created using Xtend. However, this approach involves using a complex language for modeling the PIM and another complex language, based on Java, for transforming the PSM into target code, demanding considerable effort to learn these languages and their associated libraries.

In (Benouda et al., 2016), authors propose an MDA-based solution to generate graphical user interfaces for Android applications. Their method involves two transformations: firstly, a Model-to-Model transformation employing the QVT language to convert a Platform-Independent Model (PIM) to a Platform-Specific Model (PSM). Secondly, a Model-to-Text transformation using the Acceleo tool to translate the PSM into code, aiding in Android app development. However, this approach allows unrestricted use of mobile elements in a PIM, potentially leading to issues with unknown elements and resulting in system errors. Additionally, it focuses solely on UI generation, lacking support for resources, embedded sensors, event management, and the creation of intricate user interfaces.

In (Benouda et al., 2017), authors propose a method to generate mobile phone applications, utilizing the same source meta-model as their work in (Benouda et al., 2016). They create corresponding Ecore models for both the source and target meta-models, followed by implementing a transformation algorithm using the QVT language. Their approach employs a "Translationist" method, translating a Platform-Independent Model (PIM) into the final code via a sophisticated code generator, using the Platform-Specific Model (PSM) as an intermediate step during code generation. However, this article lacks specific details or demonstrations regarding this code generator, and it retains the issue of limiting mobile elements used in the PIM, similar to their prior work in (Benouda et al., 2016).

Authors in (Salma et al., 2018), introduce the TimPhoneGenerator, a cross-platform solution based on the MDA approach. This method involves five key steps: initially defining the target platform, followed by an analysis of fundamental features associated with each platform, encompassing data storage, software architecture, user interface, access to phone data, inter-application communication, and phone feature accessibility. The third step involves defining a Platform-Independent Model (PIM) to generate a Platform-Specific Model (PSM) in the subsequent phase. Finally, they generate mobile code using the Eclipse Modeling Framework (EMF) and Acceleo tool. However, the solution's PIM comprises six PIMs addressing data storage, cross-app communication, and software architecture without detailed mention of the remaining three. Furthermore, this solution suffers from some shortcomings, notably manual transformation from PIM to PSM, requiring additional manual adjustments in the generated code.

Authors in Sebastián et al., 2020 presents a comprehensive analysis of the use of Model Driven Architecture (MDA) in the process of code generation. The study involved a systematic review of

literature in software engineering from 2008 to 2018, resulting in the identification of 50 primary studies relevant to the analysis. The study highlights the increasing interest in MDA, with a significant number of publications in various journals, conferences, and workshops. It also identifies the main application areas of MDA, such as web development, security, testing, simulation, e-learning, and IoT. The study focuses on the use of Unified Modeling Language (UML) as the most widely used modeling language for MDA, followed by Domain Specific Languages (DSL) and SysML. It also discusses the potential research opportunities in the field of software engineering, including the integration of CIM (Computationally Independent Model) in the MDA architecture and the exploration of new application domains for MDA. In conclusion, the study provides a detailed analysis of the use of MDA in code generation, highlighting current trends, gaps, and research opportunities in the field of software engineering. It also addresses the validity threats and limitations of the study, providing a structured understanding of the state-of-the-art of code generation using MDA research in software engineering.

Authors in (Shamsujjoha et al., 2021) explores the application of model-driven development (MDD) for mobile app creation. It discusses the advantages of MDD, such as enhanced productivity, reduced errors, and improved maintainability. However, the review also addresses several limitations of MDD. These limitations include the complexity of transforming models into efficient code tailored for specific mobile platforms, potential performance issues due to the automatic code generation process, and the adaptability of MDD to rapidly evolving mobile technologies, which may pose challenges in keeping the generated code up to date.

Each of the previous solutions has specific shortcomings. Some rely on multiple transformations, requiring additional effort and adjustments for each transformation. Others lack constraints on elements used in the source meta-model, leading to inconsistencies in the platform-independent model and affecting the coherence of the target code. Additionally, some solutions involve complex languages, demanding more effort to learn and expertise to develop generation templates.

In our work, we aim to circumvent these limitations by developing a solution that involves creating a source meta-model. We then construct a Platform-Independent-Model (PIM) exclusively using elements proposed in that source meta-model. Finally, we generate the target code directly from this refined PIM using Acceleo templates, streamlining the process and ensuring coherence by adhering strictly to the source meta-model's elements.

3. Research methodology

Since our solution is based on MDA approach, the first subsection will be devoted to this approach, giving an overview, and talking about its pros on cons. We will also present our proposal source meta-model which constitute the important step of our work. This source meta-model must be respected while developing the platform-independent-model. Then, we will present the Acceleo tool that we used to generate the native code directly from the platform-independent-model.

3.1. MDA approach

Model Driven Architecture approach separates the business and the technical aspects of the application, every aspect is represented by a set of models. MDA approach is based on three main model levels such as: the Computational Independent Model (CIM) which helps to represent the system’s functionalities. Then, the Platform Independent Model (PIM) which interests into the application design. And the Platform Specific Model (PSM) which interests into the technical implementations on each platform (Charkaoui et al., 2014; Jia et al., 2018), and it leads to generate code for each platform, as illustrated

in Figure 3.

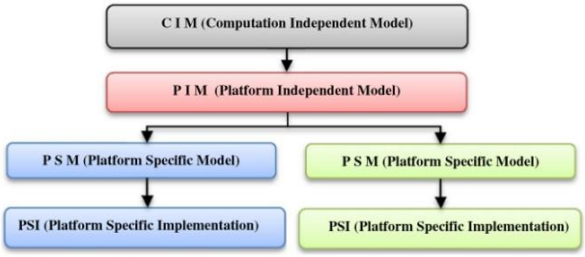

Fig. 3: MDA process

According to the MDA, a CIM will be transformed to a PIM, which will be transformed in its turn to a PSM, in order to generate a source code for each platform. These transformations of each entity of a model involve two steps (Lachgar, & Abdali ,2015). First, it identifies correspondences between source and target model concepts. Then it executes a program called transformation engine or execution, in order to generate the source target model automatically from the source one, using the QVT language (Lachgar & Abdali, 2014; Benouda et al., 2017), as shown in Figure 4.

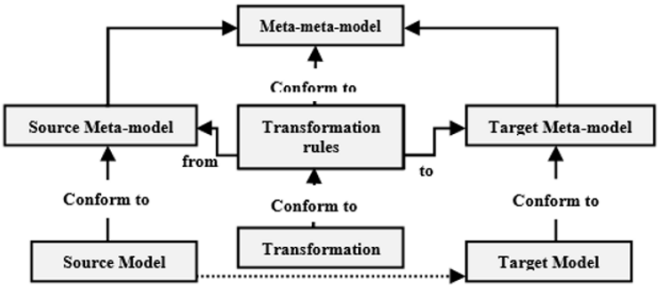

Fig. 4: MDA transformation process

MDA approach has more interest into models and functional aspects, without considering technical specifications, which reduce fragmentation problems. It could generate native code for each specific platform in order to increase productivity and decrease time. MDA capitalizes on similar concepts, since all the mobile platforms benefits from almost the same mobile concepts such as View, Controller, Events, etc. These concepts cand be modelized once and generate native code for each platform. MDA reduces the ownership charges which constitutes the main advantage. It reduces cost by reducing time and efforts since it focuses more on functional conception regardless of the platform's technical specifications.

3.2. Source Meta-model

Our solution consists of developing a platform-independent-model (PIM) respecting the source proposed metamodel. So, we limit dumps and bugs that could happen during the execution of transformation programs. We also limit the generation of unknown component which is the lack of solutions presented by authors in (Benouda et al., 2016; Benouda et al., 2017) since they choose the UML metamodel as a source metamodel.

Moreover, our solution consists of generating native code source of each platform directly from the PIM, so we skip the model-to-model transformation' step contrary to what it proposed by authors in (Lachgar & Abdali, 2014; Sabraoui et al., 2012; Salma et al.; 2018). Our solution allows to skip this intermediate step by using of Acceleo tool, which allows to automatically generate code from a PIM model.

Our solution consists of using fewer intermediate tools and techniques of model-to-model transformations and model-to-text transformations which is the lack of these solutions (Sabraoui et al., 2012; Lachgar & Abdali, 2014).

In this section, we present our proposal source metamodel that must be respected while developing platform-independent-model, to generate UI source code for each mobile platform using a transformation language. In other words, each platform-independent-model must conform to this source meta-model.

We will use the UML class stereotypes to design our source meta-model benefiting from the advantages of this language since it is the most used, the high recognized, and the most understood modeling language.

In our source meta-model, we gather all the common elements of mobile's UI: such as the Layout, label, button, input, checkbox, radio button, list box as stereotype classes. These classes represent the UI components that will be used in a platform-independent-model and that can be transformed to a platform-specific source code. Our source meta-model also regroups some enumerations such as: ViewType, InputType, LayoutType, ResourceEnum, and The OrientationEnum. They contain values of some classes' attributes such: the container type, the Input Type, etc., These enumerations allow to control user choice and avoid dumps that could be caused using unknown values.

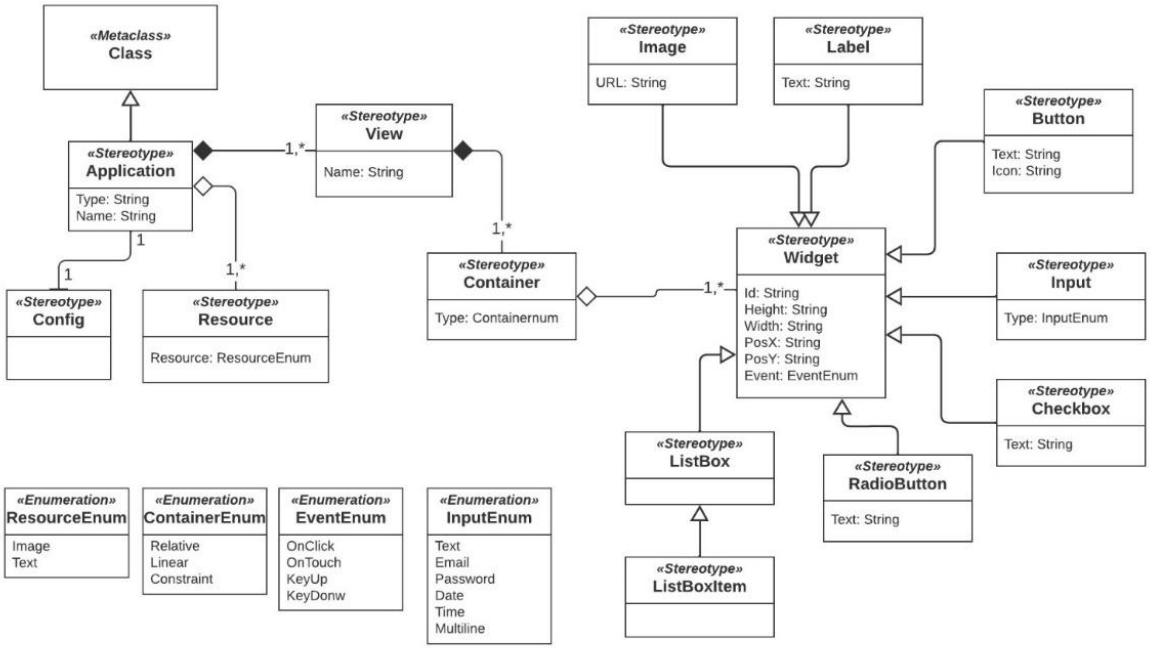

Fig. 5: Proposed source metamodel.

3.3. Acceleo tool

Acceleo is a code generation framework attached to Eclipse as a plugin and it is developed in java. It has its own development language called Acceleo Query Language (AQL) and benefits from the same advantages as the other languages such as syntax highlighting, auto-completion, error detection, quick fixes, a debugger to visualize the program's execution step by step and many others.

Acceleo has been defined by Object Management Group (OMG) after years of research and development by Obeo company. It is based on model to text transformations, supports Object Constraint Language (OCL) syntax, and allows to create source code from any input model respecting template's queries that represent the model to text transformation rules. In a few words, the template is a function composed of many queries, takes any source model that respects Eclipse Modeling Framework (EMF) format as an input parameter, and generate a source code as the output parameter.

Fig. 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.

3.4. Results and discussion

In this section, we will give step-by-step demonstration of our solution using a case study. We choose for the case study an Android interface to add a new book to the library database. This interface is composed by a title, label and input text for the book name, label and input text for the author's name, label and list box for the editor's name, label, and input number for the number of copies. It contains also two button to validate or cancel the form. The following figure represents the platform-independent-model regrouping UI elements mentioned above for creating this Android interface that allows to add a new book. This platform-independent-model conforms well to the meta-model presented above. Each element will be transformed to each equivalent android component using Acceleo template.

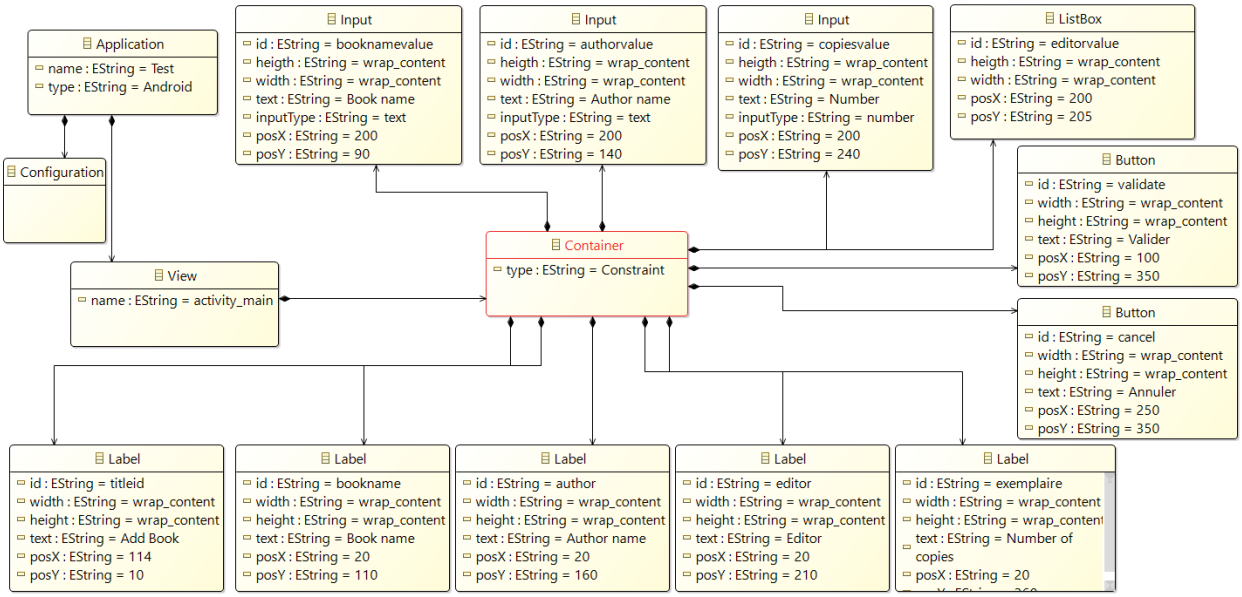

Fig. 7: Case study PIM

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 to navigate and query 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 figure 8 shows a part of the template code.

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()/]"
```

Fig. 8: 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 figure 9.

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>

```

Fig. 9: Generated Manifest

The template also generates the XML file of Android UI. The following figure 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.

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"
```

Fig. 10: Generated UI code

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 figure 11 shows the result interface containing elements modeled in

the platform-independent-model and transformed to android code using the Acceleo template.

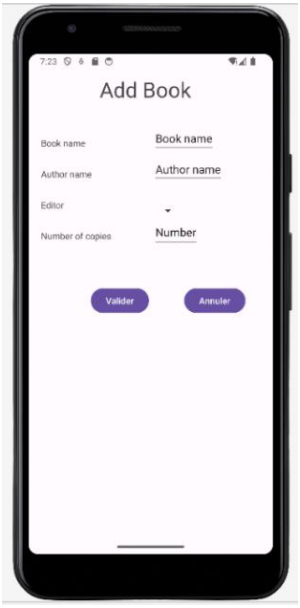

Fig. 11: Android interface

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. Till 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.

4. Conclusion and Perspectives

This paper presented an MDA-based approach to automatically generate native source code for mobile apps from a Platform Independent Model. A UML metamodel capturing core mobile concepts was proposed to guide development of consistent, reusable PIMs. The Acceleo model-to-text transformer was utilized to generate Android interface code from the PIM. The sample case study highlights the potential of this approach to reduce cost and accelerate multi-platform mobile development, compared to platform-specific coding. We chose an Android mobile interface that allows to add a new book to the existing database. We created a PIM that respects our proposal metamodel and gave it as an input parameter to an Acceleo template in order to generate the android source code of this interface. So, we generated native code source directly from PIM model and skip model-to-model transformation from PIM to PSM which enables to decrease time and efforts.

Future work will expand the metamodel with advanced constructs like transitions, sensors, access device APIs, and generate complete application code encompassing business logic and data layers for platforms like iOS and Windows.

References

Benouda, H., Azizi, M., Esbai, R., & Moussaoui, M. (2016). MDA approach to automate code generation for mobile applications. In Mobile and Wireless Technologies 2016 (pp. 241-250). Springer Singapore.

Benouda, H., Azizi, M., Moussaoui, M., & Esbai, R. (2017, December). Automatic code generation within MDA approach for cross-platform mobiles apps. In 2017 First International Conference on Embedded & Distributed Systems (EDiS) (pp. 1-5). IEEE.

Charkaoui, S., & Adraoui, Z. (2014, October). Cross-platform mobile development approaches. In 2014 Third IEEE International Colloquium in Information Science and Technology (CIST) (pp. 188-191). IEEE.

Eclipse Foundation. (2021). GENERATE ANYTHING FROM ANY EMF MODEL. Available online: https://eclipse.dev/acceleo/

El-Kassas, W. S., Abdullah, B. A., Yousef, A. H., & Wahba, A. M. (2017). Taxonomy of cross-platform mobile applications development approaches. Ain Shams Engineering Journal, 8(2), 163-190.

Eralda C., Vijayan., S. (2023, October) Classification and security assessment of android apps. In 2023 Discover Internet of Things 3, 15

Ettifouri, E. H., Rhouati, A., Berrich, J., & Bouchentouf, T. (2017, July). Toward a merged approach for cross-platform applications (web, mobile and desktop). In Proceedings of the 2017 International Conference on Smart Digital Environment (pp. 207-213).

Garg, S., & Baliyan, N. (2021, May). Comparative analysis of Android and iOS from security viewpoint. In 2021 Computer Science Review 40, 100372.

Jia, X., Ebone, A., & Tan, Y. (2018, May). A performance evaluation of cross-platform mobile application development approaches. In Proceedings of the 5th International Conference on Mobile Software Engineering and Systems (pp. 92-93).

Khachouch, M. K., Korchi, A., Lakhrissi, Y., & Moumen, A. (2020, June). Framework Choice Criteria for Mobile Application Development. In 2020 International Conference on Electrical, Communication, and Computer Engineering (ICECCE) (pp. 1-5). IEEE.

Korchi, A., Khachouch, M. K., Lakhrissi, Y., & Moumen, A. (2020, June). Classification of existing mobile cross-platform approaches. In 2020 International Conference on Electrical, Communication, and Computer Engineering (ICECCE) (pp. 1-5). IEEE.

Korchi, A., Khachouch, M K., Lakhrissi, Y., El Mazrouki, N., Moumen, A. & El Mohajir, M. (2024). Classification of existing mobile cross-platform approaches and proposal of decision support criteria, Int. J. Information and Communication Technology, 24(1), 86-112.

Lachgar, M., & Abdali, A. (2014, October). Generating Android graphical user interfaces using an MDA approach. In 2014 Third IEEE International Colloquium in Information Science and Technology (CIST) (pp. 80-85). IEEE.

Lachgar, M., & Abdali, A. (2015, November). Dsl and code generator for accelerating ios apps development. In 2015 Third World Conference on Complex Systems (WCCS) (pp. 1-8). IEEE.

Sabraoui, A., El Koutbi, M., & Khriss, I. (2012, November). Gui code generation for android applications using a mda approach. In 2012 IEEE International Conference on Complex Systems (ICCS) (pp. 1-6). IEEE.

Salma, C., Abdelaziz, M., & Issam, A. (2018). Cross-Platform Mobile Development Framework Based on MDA Approach. International Journal of Technology Diffusion (IJTD), 9(1), 45-59.

Sebastián, G., Gallud, J. A., & Tesoriero, R. (2020). Code generation using model driven architecture: A systematic mapping study. In 2020 Journal of Computer Languages, 56, 100935.

Shamsujjoha, M., John, G., Li L., Hourieh, K., Qinghua, L., (2021, December). Developing Mobile Applications Via Model Driven Development: A Systematic Literature Review. In 2021 Information and Software Technology, 140, 106693