Automated Software Engineering (2021) 28:2
https://doi.org/10.1007/s10515-020-00278-3

ALBA: a model-driven framework for the automatic generation of android location-based apps

Mohammadali Gharaat[1] · Mohammadreza Sharbaf[1] · Bahman Zamani[1] · Abdelwahab Hamou-Lhadj[2]

Received: 14 November 2019 / Accepted: 5 October 2020 / Published online: 21 January 2021

Abstract

In recent years, the number of smartphone users has increased dramatically. These users download millions of apps and use them for various services. Due to the significant demand for mobile apps, developers often seek faster development methods and more effective tools and techniques to generate these apps. Many of these apps are location-based apps in which users receive services based on their geographical location. In this paper, we propose a model-driven approach for the automatic generation of Android location-based mobile apps. Our framework, called ALBA, consists of a domain-specific modeling language, a modeling tool, and a plugin which includes model to code transformations. The modeling tool enables a novice designer to model a location-based app. The model is validated against the predefined constraints and the editor prevents creating invalid models. The designer uses the plugin to generate the Android code of the app. The evaluation of our work is two fold. First, to evaluate the generalizability of the ALBA framework, we conducted an experiment which includes the generation of four industrial location-based apps. Second, to evaluate the usability and quality of both the framework and the generated apps, we conducted a case study consists of three experiments. The results of the evaluation are promising both in terms of the applicability of the framework and the quality of the generated apps.

Keywords Model-driven software engineering · Domain specific language · Location-based android apps · Software engineering for mobile apps · Automated software engineering

✉ Bahman Zamani
zamani@eng.ui.ac.ir

Extended author information available on the last page of the article

1 Introduction

Mobile app development is one of the areas of software engineering that has seen important growth in recent years (Clement 2019), due to the increasing number of smartphones [more than 3 billion in 2019 Holst (2019a)]. While Android and iOS are the two most predominant operating systems for smartphones (Okediran et al. 2014), the former has a larger market share. Android market share reached 89% in the second quarter of 2019, while the iOS market share was approximately 10% (Holst 2019b). In the fourth quarter of 2017, the number of downloads from the Google Play store, a repository of Android apps, reached 19 billion (Sydow 2018). This market calls for more apps of different categories, including location-based apps, the focus of this paper (Raveh 2019).

Location-based apps provide users with customized information and services based on their geographical location (Oh et al. 2019). We consider a location-based app as any app that provides location-based services (LBS), which is defined by Ferraro and Aktihanoglu (2011) as follows: *"Typically you could define a location-based service as an information service, accessible with mobile devices through the mobile network and utilizing the ability to make use of the geographical position of the mobile device."* To put it another way, LBS is a service where (1) the user can specify his/her location, (2) the information provided is spatially related to the user's location, and (3) the user is offered dynamic or two-way interaction with the location information or content (Raveh 2019).

Economically, location-based apps play an important role in the global location analytics market size, which is expected to reach USD 22.8 billion by 2024.[1] Therefore, similar to any other software development endeavor, Android developers are looking for faster development paradigms, which can result in a shorter time to market. The problem is that writing an Android app from scratch is a tedious and time-consuming task (Zolotas et al. 2017; Meirelles et al. 2019). To alleviate this problem, in this paper, we resort to the use of model-driven engineering (MDE). MDE promises to cope with the complexity by rising the level of abstraction, and speeding the development process by automatic generation of the code (Brambilla et al. 2017). Several research studies (HoseinDoost et al. 2019; Benouda et al. 2016; Heitkötter et al. 2013; Vaupel et al. 2018) showed that MDE can be applied to the generation of software in different domains, including mobile app development.

To automate the process of creating location-based apps, we propose a model-driven engineering framework, called ALBA (android location-based app generator), which developers can use to automatically generate location-based Android apps in Java. ALBA consists of three components: (1) a domain-specific modeling language (DSML) that supports the concepts of Android location-based apps, (2) a graphical editor that enables developers to model an Android location-based app, and (3) an Eclipse plugin that generates the final app code from the model, based on the predefined transformations.

[1] https://www.marketsandmarkets.com/Market-Reports/location-analytics-market-177193456.html.

Developers use the ALBA modeling editor to design the app model based on predefined requirements. The model is validated against the predefined constraints and the editor prevents creating invalid models. Then, the app code will be automatically generated from the models using the transformations that are embedded in the code generation plugin. As such, we believe that the ALBA framework can facilitate greatly the development of location-based mobile apps for both novice and experienced developers. This is because of the high abstraction level that the ALBA modeling language provides as well as the automatic code generation provided by the framework. ALBA advances the development of location-based apps by benefiting from the advantages of both model-driven and code-centric paradigms, which can reduce the development time and maintenance costs compared to well-known mobile app development frameworks with the same output quality.

To validate our framework, first, we performed a case study to demonstrate the generalizability of our approach for developing real-world location-based apps in different domains. Second, we evaluated the usability and quality aspects of the proposed framework and generated apps by following a case study research, including three experiments. In the first experiment, we assessed the impact of using the ALBA framework on the development of the location-based app in a workshop with 14 participants. In the second experiment, we conducted an online user survey to evaluate the usability of an ALBA generated app. Finally, in the third experiment, we scrutinized the advantages of ALBA framework and quality of its generated apps comparing to the peer frameworks such as React Native[2] and MIT App Inventor.[3] Analyzing the obtained results of this case study shows that ALBA is an effective and efficient approach, which reduces the development time while generating high-quality apps.

The rest of this paper is organized as follows. Section 2 presents the UniFy, as a motivation example of a location-based app. The ALBA modeling language, as a DSML, is discussed in Sect. 4. The ALBA framework and the process of using the framework are covered in Sect. 5. The results of the evaluation, which shows the usefulness of the ALBA framework, are depicted in Sect. 6. Section 7 is dedicated to related work. Finally, we conclude the paper in Sect. 8, followed by future research directions.

2 Motivation

As a concrete example of the need for a location-based app, let us consider the UniFy app, a real case in the University of Isfahan. The University of Isfahan campus occupies about 650 acres and is one of the largest campuses in Iran, with more than 100 buildings, scattered around the campus. Every newcomer, including new students who come to the University of Isfahan for the first time, experience the problem of locating the buildings and classrooms on the campus. In the past, newcomers used

[2] https://facebook.github.io/react-native/.

[3] http://appinventor.mit.edu.

to find their destination on the university campus using a paper map, which was neither useful nor informative. Paper maps are not as handy as mobile apps and do not provide features such as fast searching among locations, classification of locations, and navigating around the campus. This encouraged the University of Isfahan administration to develop a mobile app (university guide app) that locates and gives full detail about more than 200 places inside the campus.

Figure 1 shows the wireframe prepared for the university guide app, which is designed based on the app requirements. The following are the requirements for the University of Isfahan guide app. The user should be able to select a category the same as what is shown in Fig. 1a for filtering a large number of locations. This is also used to prevent putting too many pins on a small area of the map. According to the wireframe in Fig. 1b, each location on the university campus should be pinned on the map. The current location of the user should be shown on the same page that represents the location pins. By selecting a pin on the map, the detailed information about that location is illustrated on another page, as indicated in Fig. 1c.

3 Location-based app development methodology

Before introducing our methodology, it should be noted that in model-driven approaches the goal is to automatically generate the code from the models. Hence, in every model-driven methodology we deal with elements such as models and transformations. The models must conform to corresponding meta-models (i.e., modeling languages). Transformations are written using the transformation languages. The generated code will run on the target platform. Our methodology is inspired by the methodology presented by Brambilla et al. (2017).

Brambilla et al. (2017) proposed a methodology, which is a comprehensive top-down process for system development based on the model-driven software engineering (MDSE) philosophy. In this work, we adopted their methodology to develop our model-driven approach for building Android location-based apps using the ALBA framework. Our methodology is illustrated in Fig. 2.

In Fig. 2, we see the two main aspects of every model-driven approach, i.e., Conceptualization and Implementation, as horizontal and vertical dimensions, respectively. The Conceptualization dimension covers the abstraction aspects. This dimension is divided into three columns in the figure: Application, Application domain, and Meta-Level. The Implementation dimension covers the development aspects. This dimension is divided into three rows: Modeling, Automation, and Realization.

Figure 2 can be scanned column-wise or row-wise. From the one hand, if we look at the figure column-wise, the first column (Application) shows the core flow of MDSE, in which the model of app (App Model) is designed, and then by running the written transformations (Transformation/Code generation), this model is converted into the code of application (Java and XML Files), automatically. Due to the fact that, this column is per se a process for developing apps using the ALBA framework, we will come back to this column again and show it as a process in Sect. 5 (Fig. 6).

The second column (application domain) is in fact one level above the previous column, in terms of abstraction. This is the first meta-level, in which we need to design our modeling language, or our meta-model (ALBA Modeling Language), by which we can design our app model. Also, in this level, the transformations (EGL Template Files) are written to convert a model to the corresponding app. Finally, the platform on which the generated code will be compiled to prepare the APK file, is specified at this level. In our work, the generated code is supposed to be compiled on the Android Studio. This is why Android Studio is indicated at this level.

The third column (meta-level) is more abstract than the previous column. In this level, we specify the underlying languages both for designing our modeling language and for writing our transformations. In ALBA, we designed our modeling language using the Ecore concepts, and we wrote our transformations using EGL transformation language. Clearly, we do not deal with the language in which Android Studio is built, hence, that cell in the figure is left blank.

On the other hand, if we look at Fig. 2 row-wise, we see that the first row (Modeling) shows that in ALBA, the App Model is built using the ALBA Modeling Language, which is per se built based upon the Ecore language. The second row clarifies that the Transformation/Code generation in ALBA is achieved by writing EGL Templates using the EGL transformation language. The third row indicates that the code that is generated by ALBA are JAVA and XML files that will be fed to Android Studio for generating the corresponding APK file.

To summarize, in the following, we discuss the three primary steps to implement an Android location-based app using the ALBA framework based on the methodology depicted in Fig. 2.

1. *Modeling* The first step is modeling the location-based apps; models of these apps should be based on a modeling language, which is "ALBA Modeling Language". This shows that it is essential to have a modeling language (meta-model) which specifies meta-classes, relations, and gives predictable structure to the models. In Sect. 4, the ALBA Modeling Language is discussed. Besides, to facilitate the modeling by the ALBA Modeling Language, a graphical modeling editor is implemented. Section 5.1 provides detailed information about this editor. Modeling editor is part of the ALBA framework, which is discussed in Sect. 5.
2. *Automation* After modeling, the model needs to be transformed into the desired code. In our research, we have implemented various model-to-code transformations to generate all Java and XML layout files automatically from the input model. In Sect. 5.2, we explain the ALBA transformations, as another part of the ALBA framework.
3. *Realization* The output of the ALBA framework is the auto-generated files from their co-responding model. These files can be edited by Android Studio as default IDE for Android development. Section 5.2 gives information about the generated files for our case studies.

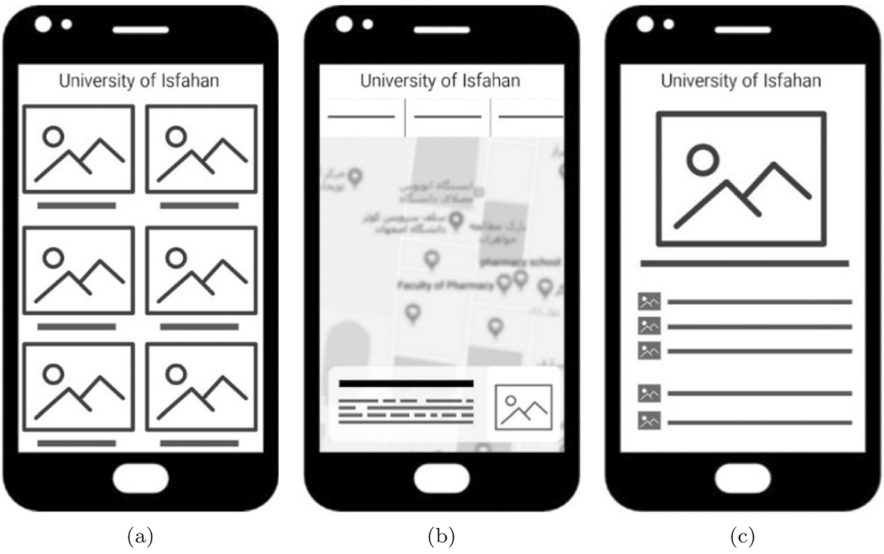


Fig. 1 Wireframe of the UniFy app. (**a**) Categories, (**b**) map, and (**c**) location details

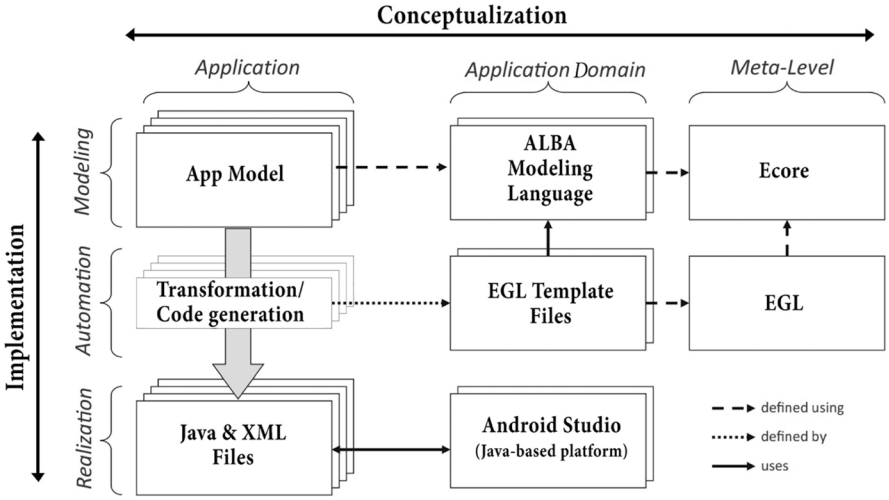


Fig. 2 A model-driven methodology for developing Location-based Apps using ALBA framework. (Adapted from Brambilla et al. 2017)

4 ALBA modeling language

This section presents the proposed modeling language, ALBA DSML, which supports the concepts and features required for location-based Android app development. To define the ALBA DSML, we followed the approach introduced by Mernik

et al. (2005). This approach consists of five phases: decision, analysis, design, implementation, and deployment. In the decision phase, the rationale for developing a new DSML is presented. In the analysis phase, the domain concepts and terminology are discovered. After completing domain analysis, the design phase starts, and the relationships between the concepts are specified. The result is also called the meta-model of the language, in terms of the model-driven concepts. Indeed, the meta-model as a class diagram is used to define the abstract syntax of the language in terms of the model elements and their relationships (Brambilla et al. 2017). In addition, OCL invariants are added to the abstract syntax to express the semantic constraints on the language (Kurtev et al. 2006). In the implementation phase, the language entities and their relationships, i.e., the meta-model, are implemented in a tool. The deployment phase focuses on the use of the DSML on an executable platform. For the ALBA modeling language, these phases are explained in Sects. 4.1 to 4.5, respectively.

4.1 Decision

The rationale for proposing a new DSML for the domain of location-based app development is two-fold. On the one hand, defining a new DSML for a domain will result in a higher level of abstraction, which in turn results in more productivity (Brambilla et al. 2017; Alfraihi et al. 2018). On the other hand, it can help developers satisfy the high demand for mobile apps, in general, and location-based apps in particular (Dehlinger and Dixon 2011). Hence, a new DSML for designing location-based Android apps is considered essential. As it will be discussed later (in Sect. 7), and to the best of our knowledge, the domain of Android location-based app development is not covered yet by any dedicated DSML, particularly considering the level of abstraction that ALBA provides to the user.

The goal of this research is to design a new DSML to be used by the Android programmers. The DSML automates the implementation of code skeleton and map configurations, and generates components of a location-based app, and adds them to the skeleton using MDE techniques.

4.2 Analysis

For the analysis phase, we applied the feature-oriented domain analysis (FODA) approach (Kang et al. 1990). FODA describes a method for classifying features of the applications in a special domain by distinguishing the mandatory and optional features and their relations. To obtain a feature model for the domain of location-based apps, we went through the following steps. First, we selected some of the most popular apps in this domain from GooglePlay. Second, we classified the concepts that are supported by these apps. Finally, we constructed a feature model for the features of the apps in this domain, which are the ones supported by ALBA.

After a thorough analysis of several location-based apps available on the Google Play store, we found five popular apps that are used worldwide. As it is depicted in Table 1, these apps are considered "popular" due to the high number of downloads

(indicated at the third column of table) and also their star ratings (indicated in the second column of table). The selected apps are as follows:

1. *Google Map*[4]: A Google navigation app used for finding the best path between two points and locating different locations.
2. *Booking.com*[5]: A popular app and website to facilitate renting hotels, apartments and cars.
3. *Trip Advisor*[6]: The world's biggest travel site with an average of 455 million unique visitors every month.
4. *Airbnb*[7]: An app providing more than two million locations in 191 countries.
5. *Foursquare*[8]: A popular city guide app and a place recommender based on customer needs.

To extract the domain concepts and terminology, we classified the essential concepts of these apps based on their functionality see Table 2. The names chosen for the concepts in the table are similar to the names of the pages of the apps. This is because the same features are always organized together on a page. We observed that the studied apps use mainly five pages dedicated to the following functionalities: (1) Map, (2) Login and Sign Up, (3) Category Selection, (4) Subcategory Selection, and (5) Location Details. Also, we found that existing apps use two navigation patterns namely "Navigation Drawer" and "Tab Bar", which are also described in Google material design patterns.[9] In the followings, we describe the details of the selected pages and navigation patterns.

- *Map page*: It refers to the pages of these apps with a full-screen map view, representing the positions of the locations on the map. The map used by these apps relies usually on the Google map library. In addition, there are other functionalities that are implemented on this page such as showing the current location on the map.
- *Login and sign up page*: The Login and Sign Up is used in all the studied apps. This helps the companies save histories of user navigation pattern for recommending locations according to their needs.
- *Category selection page*: Since the number of locations could be quite large and it is not possible to show all of them together on the map, all apps classify their locations using a two-level categorization.

[4] https://map.google.com.
[5] https://www.booking.com.
[6] https://www.tripadvisor.com.
[7] https://www.airbnb.com.
[8] https://foursquare.com.
[9] https://material.io.

Table 1 Selected apps for the analysis phase

App name	No. of stars (from GooglePlay)	No. of downloads (from GooglePlay)
Google Map	4.3/5	5,000,000,000+
Booking.com	4.8/5	100,000,000+
Trip Advisor	4.4/5	100,000,000+
Airbnb	4.7/5	50,000,000+
Four Square	4.3/5	10,000,000+

- *Subcategory selection page*: This page is necessary for narrowing down the range of the locations. Some apps use a completely different page for their subcategories; others use pop-up windows.
- *Location details page*: This page is dedicated to showing complete information about the desired location. Moreover, some functionalities such as bookmarking, navigating to the location, leaving a comment, and rating the place could be implemented on this page.
- *Navigation drawer*: This navigation pattern consists of a navigation drawer, which appears on one side of the screen and shows a list of pages that the user wants to jump to directly in the app.
- *Tab bar*: This navigation pattern consists of implementing a bottom tab bar that represents a horizontal list of pages that a user can move to directly. The user can also move to the neighbor pages by swiping the screen. "It is recommended to have a Tab Bar that shows no more than five pages, which are at the same level of importance. If the number of pages is higher than five, then it is better to use the "Navigation drawer."

In addition to the functional features described above, there is more location-based features. Figure 3 shows other features that should also be supported by a location-based app. These include: Device Type, OS Type, User Interface Modes, Geo Services, and Data Management Models.

Device type refers to the type of devices on which the app can be installed. ALBA supports both smartphones and tablets. User interface (UI) modes can be either "Static" or "Dynamic". A static UI means that the content of the UI components is hard coded, while a dynamic UI changes displayed data dynamically based on various settings. ALBA supports customized material theme color palettes[10] and icons. Additionally, ALBA can identify pre-designed cards inside its constant pages, meaning that the user of ALBA can design a card in Android studio independent of ALBA and then specify the name of the file and the card items in the ALBA modeling editor to let ALBA recognize what is within the card view. Then ALBA generates code for connecting those card items to the server API given in the model.

[10] https://material.io/develop/android/theming/color.

Table 2 Classification of the essential concepts of the selected location-based applications

	Pages					Navigation patterns	
Concepts of applications	Map	Login & sign up	Category selection	Sub category selection	Location details	Navigation drawer	Tab bar
Google Map	●	●	●	◐	●	●	●
Booking.com	●	●	●	◐	●	●	○
Trip Advisor	●	●	●	○	●	○	●
Airbnb	●	●	●	○	●	○	●
Foursquare	●	●	●	○	●	○	●

Legend: ● Fully implemented ◐ Partially implemented ○ Not implemented

In the ALBA framework, we mixed the user interface and the app logic for the sake of simplicity and consistency. Using ALBA, the app logic and user interface can be implemented in three ways: (1) By the relationships between classes in the model. For example, when a connection between a Fragment and a TabBar is modeled, the ALBA framework will generate the related logic and user interface. (2) By changing a Boolean attribute in the model. For example, the LocationDetailsFragment meta-class has some Boolean attributes which can add some special functionality to the final app. If we set the "ShowCommentOption" attribute to True, the framework generates the app logic and user interface for showing comments about each location to the app users. (3) By generating a skeleton code for the one who is going to extend the app, ALBA gives a structure for implementing the rest of the requirements that are not supported by ALBA. For instance, the Fragment meta-class can be used for this purpose.

Geo services include a map and a GPS sensor. These are the most important features of any location-based app and are supported by all studied apps. Data management models play an essential role in location-based apps. There are two options for data management: Local and Server. For example, Google Trip was known for its ability to save all of the information locally, a helpful feature when an Internet connection is not available. ALBA supports both data management options, which means that it keeps the data locally and tries to update the data from the server when an Internet connection is accessible. Due to the Google map cache mechanism, the generated apps can be used offline.

As shown in Fig. 3, our framework covers most of the features that we expect of any location-based application. This helps developers to focus on implementing the key features related to the app instead of implementing the common ones.

Finally, despite the fact that all the studied apps are designed for both iOS and Android, for the present time, our framework only supports Android.

4.3 Design

We designed the ALBA DSML based on the concepts that are described in the analysis phase. The design shows the elements that are required in the language, as well as the relationships among these elements. The ALBA DSML was designed using familiar terms since we assume that users of ALBA are not necessarily proficient in programming. We refer to the concepts of the language and the relationships between these concepts as the meta-model of the language (Combemale et al. 2016).

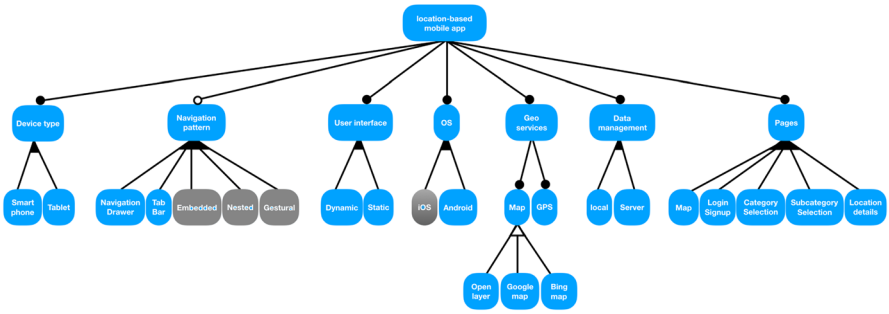


Fig. 3 Feature model for location-based app (blue features are supported by our framework) (Color figure online)

The ALBA meta-model not only supports all the features that are common to the studied apps, but also accepts customization in user interfaces, navigation patterns, and data providers for their content. ALBA can provide model and implement Java code for the communication between the app and the data provider (e.g., web service).

The ALBA meta-model is shown in Fig. 4. There is a root meta-class, called **APP**, that represents the Android app that is modeled. It has three mandatory children: (1) **Configuration**, (2) **Theme**, and (3) **MainActivity**. Every Android app has configurations, e.g., package name, minimum SDK version, and target SDK version, which are represented in the *Build.gradle* file and the *manifest* file in the Android project. These configurations can be specified in the **Configuration** meta-class. Android apps consist of three primary colors, as it is mentioned in the Google material design principles. These colors and the name of the parent theme will be set in the **Theme** meta-class. Android apps need at least one Activity for showing the views and responding to user actions. Almost all activities interact with the user, so the Activity class takes care of creating a window in which the designer can place the UI elements.[11]

The ALBA meta-model consists of other meta-classes, described in the following.

- **MainActivity:** This is responsible for creating a window and producing a response to user actions, so we involved **MainActivity** meta-class in the ALBA meta-model, which controls the transition between different pages; these pages are called fragments.
- **Fragment:** Fragments are considered as predefined pages with unique user interfaces and logic, which are shown to the end-user by the **MainActivity**. There exist five types of fragments in our DSML, including **CategoryFragment**, **LoginFragment**, **LocationsFragment**, **LocationDetailsFragment**, and *EmptyFragment*. We have some predefined fragments with their functionalities as well as an empty fragment, which is only a skeleton for requirements that ALBA

[11] https://developer.android.com/reference/android/app/Activity.html.

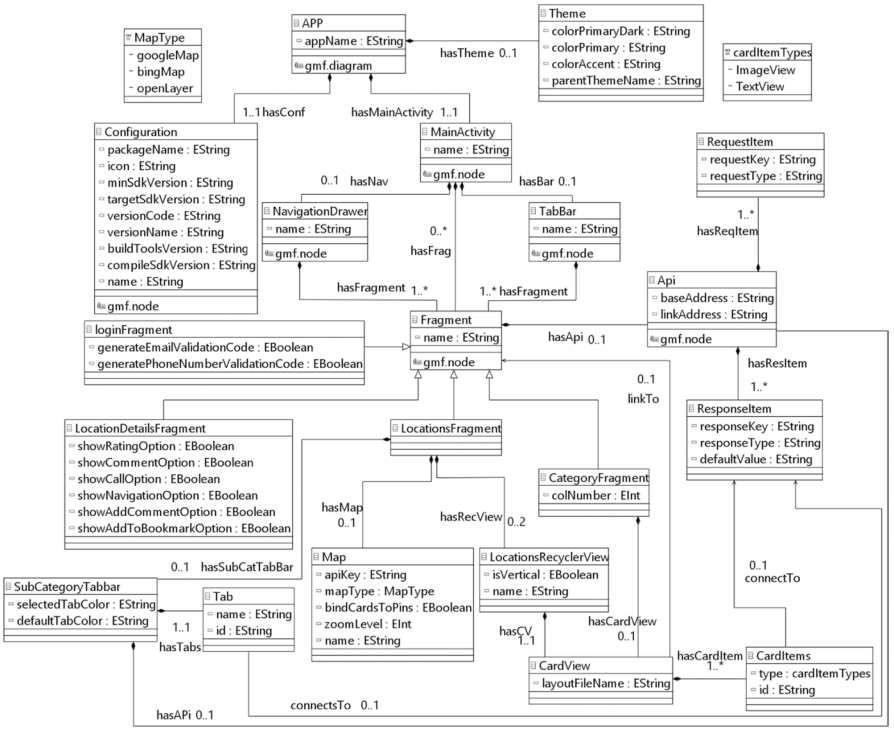


Fig. 4 ALBA meta-model

does not support yet. Each fragment has a primary role in the app; taking the **LoginFragment** as an example, the login and signup screens and some functions like mobile phone and email validation will be implemented due to presence of **LoginFragment** in the app model. The next fragment is **LocationsFragment**, which consists of a **Map** and could have two recycler views[12] named **LocationsRecyclerView** in the meta-model for representing the locations lists in a vertical, and a horizontal recycler view.

- **NavigationDrawer and TabBar:** Navigation between fragments depends on the navigation pattern preferred by the engineering team. It is mainly implemented in two ways (1) **TabBar**, and (2) **NavigationDrawer**. These two concepts are also supported in the ALBA meta-model.
- **Api:** The **Api** requests and responses can be defined in each fragment not only to get the information from the server and show it in a **CardView** but also to store it in the local database for offline use. It is advisable that all of the fragments can communicate with a web service sending parameters to the server and expecting to receive other parameters from the server by features of the **Api**, **RequestItem**, and **ResponseItem**. After retrieving the data from the server, the framework

[12] https://developer.android.com/guide/topics/ui/layout/recyclerview.

shows received information in the **CardItems**. Card items are either (1) **imageView** or (2) **textView**.

ALBA-generated apps need to obtain data from an end-point, which is specified by the **Api** class in the app model. For the sake of simplicity, the ALBA framework forces **Api** responses to have some mandatory fields. For instance, every location must have latitude and longitude. The data model, depicted in Fig. 5, shows the minimum entities that a location-based app requires in its database. It shows structures that data of the app should satisfy in order to be supported by the ALBA. The locations, location categories, and subcategories with their relationships are essential for ALBA.

4.4 Implementation

The ALBA meta-model is implemented using the Eclipse Ecore tool, which is the graphical modeling tool for defining modeling language concepts in the Eclipse IDE. Ecore is the meta-language of the Eclipse Modeling Framework (EMF) (Steinberg et al. 2008). To implement the ALBA meta-model by Ecore, all its meta-classes and their relationships are defined using Ecore, which is enriched with OCL invariants.

4.5 Deployment

The meta-model is, in fact, the abstract syntax of our DSML. To use an abstract syntax, we need a concrete syntax and a modeling editor. In this phase, we build the concrete syntax of the language, i.e., the notations that are selected for each concept as well as the ALBA modeling editor, based upon the concepts that are defined in the abstract syntax. Our editor is built using the Eugenia[13] editor generator. Eugenia is a tool that automatically generates a GMF editor from just an Ecore annotated file (Kolovos et al. 2017). We will go through the details in Sect. 5.1.

5 The ALBA framework

The ALBA framework allows the user to design an Android location-based app using the ALBA editor, then the designed model is transformed into an Android app using the ALBA code generator plugin. The source code of the ALBA framework is available from GitHub[14] under the Apache 2.0 license. Figure 6 shows the process of building an Android app using the ALBA framework. This process is fully aligned with the methodology shown in Fig. 2 of Sect. 3.

Similar to any software development process, the preliminary step is to collect the requirements. Clearly, this step is out of the scope of the ALBA framework.

[13] https://www.eclipse.org/epsilon/doc/eugenia/.
[14] https://github.com/MohamadAli22/ALBA-framework.

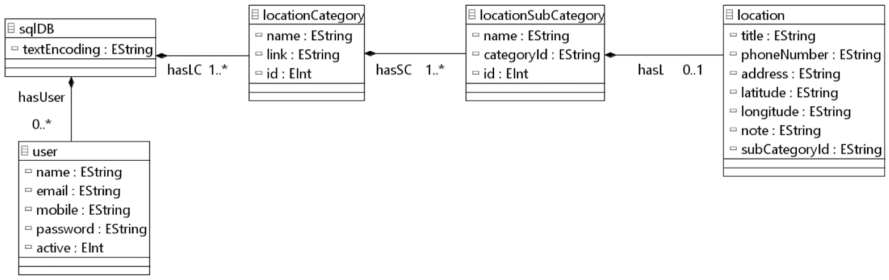


Fig. 5 ALBA data model

Next, as indicated by the dashed box labeled "Modeling", the requirements and user stories should be modeled by the modeler using the ALBA Modeling Editor to describe the structure of the required app. The result is called App Model in the figure. Next, as indicated by the dashed box labeled "Automation," the app model is transformed into the app code using the ALBA Code Generator. Finally, as indicated by the dashed box labeled "Realization," the app code is revised by the user (if required) and then it will be compiled into an APK file using the Android Studio. Since Android Studio is the primary IDE for developing Android apps, ALBA generates the code in a format that is fully compatible with Android Studio.

In the rest of this section, the ALBA Modeling Editor and the ALBA code generator as the main parts of the ALBA framework are further explained.

5.1 ALBA modeling editor

After the meta-model is designed in the Ecore format that is supported by Eclipse modeling tools, the meta-model file is annotated to specify the links between the nodes, where every node can be placed by the user, and what are the visual properties of the node.

The ALBA modeling editor environment is shown in Fig. 7. The editor has three main panes as follows: (1) design pane, (2) objects pane, and (3) connections pane. The design pane is considered as the root node, which corresponds to the **APP** node in the meta-model. The user can take advantage of the editor by selecting an object and dropping it into the design pane. The editor dynamically and automatically prevents the meaningless objects hierarchy or incorrect connections between the objects. As an example, the model of a **"locations fragment"** is depicted in Fig. 7. In the designed model, (4) the given card view has four parts: **title**, **address**, **phone number**, and **image**. Three of them are **text views**, and another item is an **image view** as these **card items** require to be connected to the response **Api** class (5). When the page is being created, the app tries to get the data from the web service, which is specified in the model, modeled as an **Api** class. However, if the data is not available and the local database was created before, the app searches for the information in the local storage instead of requesting data from the end-point.

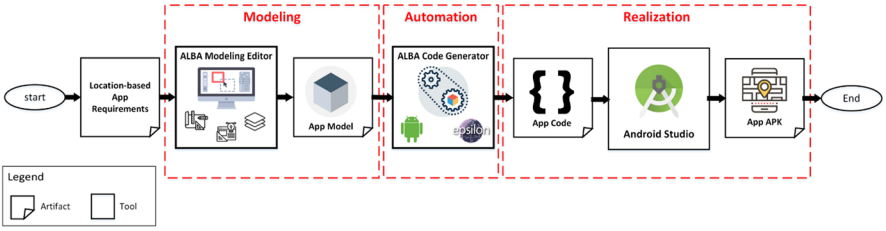


Fig. 6 The process of generating an app using ALBA framework (Aligned with Fig. 2)

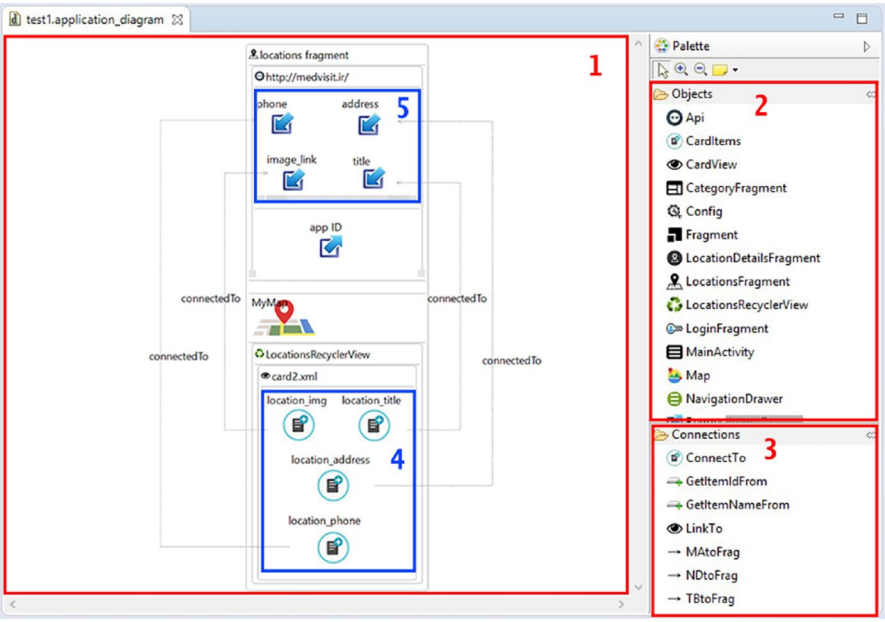


Fig. 7 The ALBA modeling editor environment

5.2 ALBA code generator plugin

The code generator is an Eclipse plugin which is responsible for importing models, and then running a list of Epsilon Generation Language (EGL) template files in coordination with a single EGX file. The EGX files are role-based sub-languages for template execution. They can generate different files based on a user-defined model (Kolovos et al. 2010). The plugin generates code in folders structure, which are handy to be opened by Android Studio. Table 3 shows the number of lines of code implemented in the template files and the number of template files used in the ALBA code generator.

To show more details of the implementations, two samples are explained in the following. Listing 1 reveals the content of an EGX file that is responsible for

Table 3 Statistics about ALBA code generator

Implemented transformations	
Line of code	Number of templates
4000 lines	24 files

executing the EGL files regarding the model. The rules are executed one by one. For example, if the Configuration class is present in the given model, at the run time of the EGX file, rules R0 and R1 will be executed.

It happens because the rule R0 and R1 both are bound to the Configuration in the EGX file of the framework. This means that the Configuration class from the model along with its properties are passed to the *buildGradleModule.egl* and *manifest.egl* file, and they will generate the code of the *build.gradle* file and manifest file of the Android Studio project.

```
1
2     rule R0    transform conf : Configuration
3     {
4        template : "buildGradleModule.egl"
5        target : "generatedCodes/buildGradleModule/build.gradle"
6     }
7
8     rule R1    transform conf : Configuration
9     {
10       template : "manifest.egl"
11       target: "generatedCodes/manifests/AndroidManifest.xml"
12    }
```

Listing 1: A sample EGX file used in ALBA

Listing 2 shows the content of one of the template files that we have used in the ALBA framework that is the *buildGradleModule* file responsible for generating the contents of the *build.gradle* file. The *build.gradle* file holds the Gradle toolkit build settings.[15] The compile setting of the project, based on the configurations made by the user in the model, will be set as shown in Listing 2.

[15] https://developer.android.com/studio/build/index.html.

```
android {
    compileSdkVersion [%=conf.compileSdkVersion%]
    buildToolsVersion "[%=conf.buildToolsVersion%]"

    defaultConfig {
        applicationId "[%=conf.packageName%]"
        minSdkVersion [%=conf.minSdkVersion%]
        targetSdkVersion [%=conf.targetSdkVersion%]
        versionCode [%=conf.versionCode%]
        versionName "[%=conf.versionName%]"
    }

    buildTypes {
        release {
            minifyEnabled false
            proguardFiles
            getDefaultProguardFile("proguard-android.txt"),
            "proguard-rules.pro"
        }
    }
}
```

Listing 2: A sample EGL file used in ALBA

6 Evaluation

This section addresses the generalizability and usability of the proposed approach on the development of Android location-based apps by analyzing both ALBA framework and resulting apps in two main parts. In the first part (Sect. 6.1), the generalizability of ALBA framework has been demonstrated by carrying out an experiment, which consists of creating different real-world location-based apps. The term generalizability refers to how far the generated apps are applicable and valid in the context of location-based apps. To enhance the rigor of this experiment, we follow the guidelines defined by Runeson and Höst (2009), which include the definition of research questions, designing the case study, and discussing about the results. In the second part (Sect. 6.2), we evaluate the usability and quality aspects of the proposed framework and generated apps. For this evaluation, we perform a case study research following the guidelines suggested by Wohlin et al. (2012). The steps recommended by this guideline are as follows: evaluation setup, planning and data collection, and analyzing the results. In the following sections, we go through the details of each activity for both parts of the evaluation. Also, we address the threats to validity for our work in Sect. 6.3.

6.1 Evaluating the generalizability of ALBA framework

In this section, we demonstrate the generalizability of ALBA framework for developing location-based apps on the basis of a variety of real-world location-based

cases. In the following, we describe performed activities based on the guidelines by Runeson and Höst (2009).

6.1.1 Research questions

ALBA allows specifying a location-based app using a graphical modeling editor in which the user can design an app model to satisfy his/her requirements. Based on the app model, Android code are generated automatically using the ALBA code generator. Thus, two essential factors that need to be assessed are the ability to support the requirements of location-based apps and the development effort. Therefore, the research questions for this study are posed as follows.

RQ1. Is ALBA framework an appropriate solution to design and deploy the requirements of location-based apps?
RQ2. How much effort is required to generate a location-based app using ALBA framework?

6.1.2 Case study design

Setup. We chose to analyze the development process of four location-based apps coming from real-world needs. These apps are as follows: (1) UniFy, (2) Baharestan, (3) MedIUM, and (4) Covid-19 Dashboard. In our experiment, we considered the modeling and development of apps based on customer requirements. Each app addresses different functional requirements, where displaying information about locations is a key feature.

The first app is called UniFy, a guide app for the University of Isfahan, which is a typical example of location-based apps. UniFy guides newcomers through the university campus. It was made available to all students of the University of Isfahan from October 2018. Figure 8 shows three main pages of UniFy app, and its requirements were wholly described in Sect. 2.

The second app is Baharestan, which provides information about economics, recreational, and cultural places in the city of Baharestan[16] to the citizens and investors. In addition, this app provides an interactive user interface for receiving comments and suggestions from the user about each location. The Baharestan app was ordered by Baharestan city municipality and has been available to Baharestan residents since 2019. Three pages of this app are shown in Fig. 9.

The third app is MedIUM, which is a guide app that has been generated for the Isfahan University of Medical Sciences.[17] The MedIUM app requirements are mainly similar to the UniFy app. However, compared to UniFy, it has three different requirements, e.g., the News page. The MedIUM app was made available to all students of the Isfahan University of Medical Sciences from April 2019. Figure 10 shows three main pages of the MedIUM app.

[16] https://en.wikipedia.org/wiki/Baharestan,_Isfahan.
[17] http://english.mui.ac.ir/.

The fourth app is Covid-19 dashboard, which is generated due to the coronavirus pandemic to show the latest statistics about the coronavirus outbreak all around the world. In this app, users can find real-time information such as the number of confirmed coronavirus cases and mortality rate for all the countries in the world. Three pages of this app are shown in Fig. 11.

To conduct this experiment, first, we modeled four mentioned apps and their Android code was generated using the ALBA framework. Then, the generated code was reviewed based on the customer requirements, and if needed, the code were manually edited. During the experiment, the time for doing each required task has been independently collected. Therefore, by comparing the data obtained from our experiment, the research questions can be answered appropriately.

Measures. For determining the generalizability of the ALBA framework, we consider quantitative measures that are related to the required effort and the completeness of the requirements.

Firstly, to investigate the required effort, we need the size and development time of the generated apps. The size of apps can be measured in terms of lines of code (LOC). LOC is a general metric for all kinds of software and does not show the effort saved in the automatic code generation using the ALBA framework. Thus, we also need another metric that indicates the extent of automatically generated code compared to the manually-written code. The development time can be computed based on the number of hours spent on the development process. However, for identifying the usefulness of the ALBA framework, we need to determine how much development time has been spent on modeling. Consequently, we define LOC, the percentage of automatically generated code, the overall time consumed for the app development, and the time spent on the modeling, as measures for investigating the size and development time for apps generated by the ALBA framework.

Secondly, to investigate completeness of requirements, we need the number of completely supported requirements comparing to the total number of requirements. A requirement is fully supported by an app if all needed features and its associations are developed correctly. Hence, we define the number of modeled requirements and the total number of requirements as the measures for investigating the completeness of the requirements in the ALBA modeling process.

6.1.3 Results

We gathered data for the mentioned apps during the development process by the ALBA framework. Then, we computed the quantitative measures for the requirements completeness and the cost of the development process for each app. The results of our experiment are depicted in Table 4.

RQ1: Is ALBA framework an appropriate solution to design and deploy the requirements of location-based apps?

As shown in Table 4, for the UniFy app, 20 out of 25 requirements (80%) are supported by modeling capabilities. The five requirements that are not supported by modeling are *creating help for the user*, *adding the Persian language*, *finding the distance to the destination*, and calculating the cost of *Snapp* and *Tapsi* internet taxi services, which are not considered as location-based requirements. Also, the

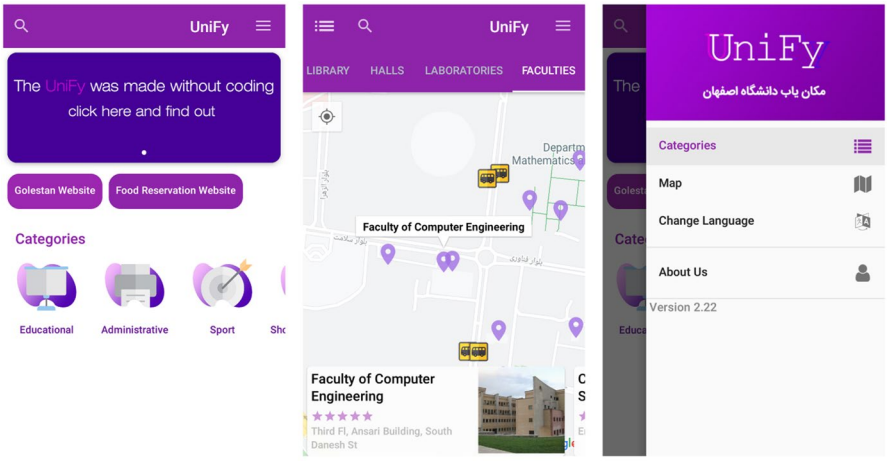


Fig. 8 Three main pages of UniFy App

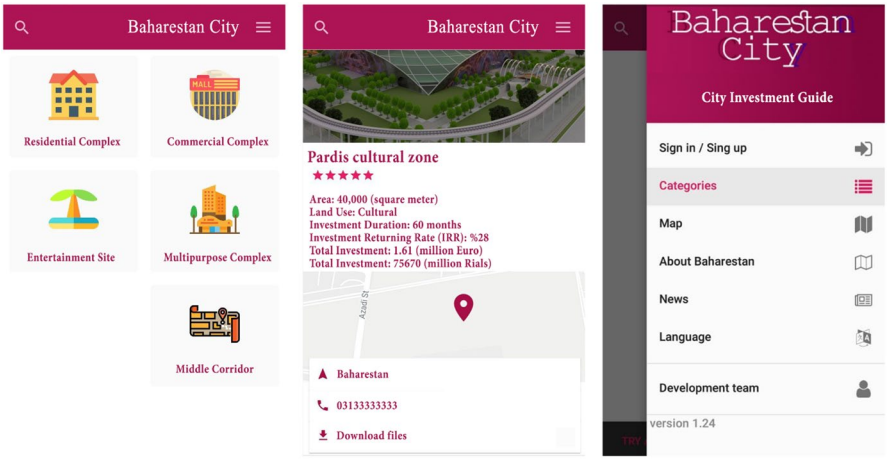


Fig. 9 Three main pages of Baharestan App

percentage of covered requirements for the MedIUM app is about 83%, which is very close to UniFy. This is due to high similarity between the requirements of these two apps.

As a different case, for Baharestan app with 23 requirements, 18 requirements are supported by the modeling capabilities, which shows more than 78% coverage. In the Baharestan app, in addition to *creating help for the user* and *adding the Persian language*, three other requirements, including *adding About*

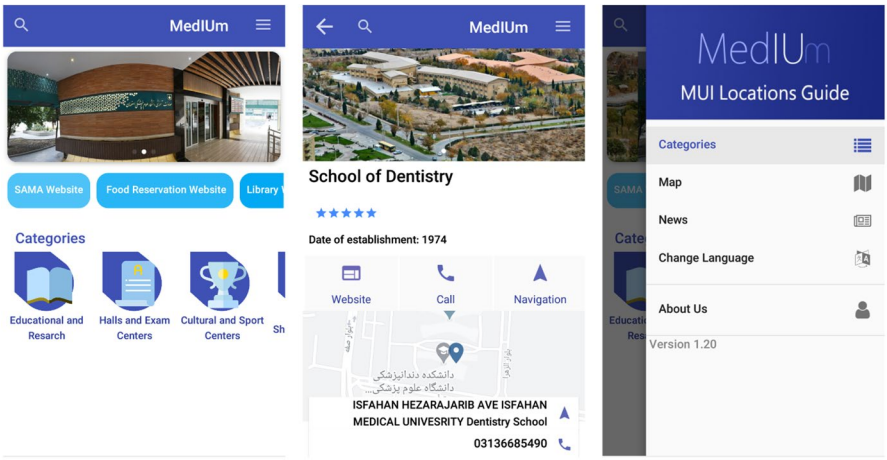


Fig. 10 Three main pages of MedIUM App

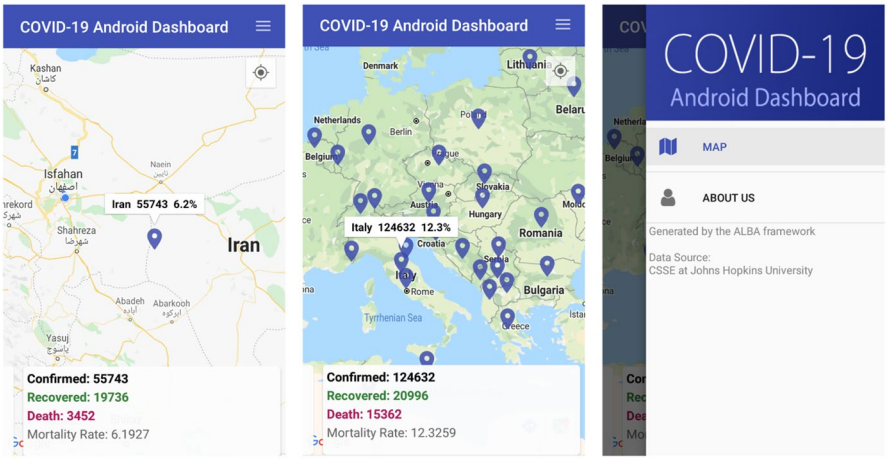


Fig. 11 Three main pages of Covid-19 dashboard App

Baharestan page, *adding News page* and *adding the capability of receiving the user suggestions*, are not entirely supported by the modeling. For these requirements that are not inherently location-based, ALBA creates a skeleton, which needs a little manual coding. In addition, for Covid-19 dashboard app, which is an app with only location-based requirements, we reached 100% coverage using the ALBA modeling capability.

Table 4 Results of evaluating the generalizability of ALBA

App name	# Total requirements	# Modeled requirements	% Requirements (%) covered in model	Modeling time (h)	Overall time (h)	#LOC	% Auto generated code (%)
UniFy	25	20	80	~ 6	~ 10	~ 2000	95
Baharestan	23	18	78	~ 5	~ 22	~ 3000	80
MedIUM	24	20	83	~ 6	~ 14	~ 2400	90
Covid-19	6	6	100	~ 0.5	~ 2	~ 1200	100

In summary, we found that the ALBA framework could support most frequent requirements of location-based apps. Based on the percentage of modeled requirements in Table 4, ALBA covers approximately 80% of the overall requirements of our experiment by the modeling capabilities. Most of the unsupported requirements were not originally location-based requirements. However, the ALBA framework supports the development of such requirements, mostly by providing code skeletons.

RQ2: How much effort is required to generate a location-based app using ALBA framework?

As we mentioned earlier, the first step of the development process in the ABLA framework is modeling. After modeling, the Android code is generated automatically and is modified if required. As shown in Table 4, the overall time for developing the Unify app was 10 h, of which 6 h were spent on modeling. However, the result of modeling time was generating approximately 95% of the total 2000 lines of code of the UniFy app. The modeling time for MedIUM is roughly equal to the time for Unfiy. However, because of similarities between the requirements of MedIUM and Unify, the MdeIUM app model was primarily reused from the UniFy app model and reduced the modeling time by about 1 h. Reusing the previous models to create similar apps is one of the main advantages of the ALBA framework.

As shown in Table 4, the total time required to develop the MedIUM and the Barharestn apps are more than UniFy. However, their modeling times and their total number of requirements are almost the same. This extra required time is due to the lower percentage of automatically generated code for these apps that leads to writing more manual codes. In general, considering the overall time of development, we found out that with the decreasing percentage of automatically generated code from models, the overall development effort spent on manual code witting was increased quickly.

In contrast to other apps, the Covid-19 dashboard app was generated entirely automatically. The overall time for developing the Covid-19 app was 2 h, of which, half an hour was spent on modeling, and the rest was used for preparing APIs and app generation. Furthermore, compared to the manual coding, based on the productivity of average programmers (Dalmasso et al. 2013), the Covid-19 app needs 12 person-day effort to be built and tested. However, using the ABLA framework, we generated it with only 1/4 person-day effort.

6.2 Evaluating the usability and quality aspects

In this section, we evaluate the proposed approach based on the usability and quality aspects. In order to evaluate the usability, the most reliable approach is to conduct a case study with users (Dillon 2001). To assess the quality of the apps generated using the ALBA framework, an option is to perform a case study based on the real-world cases and then compare the results using quantitative measures. To this end, we conducted a descriptive case study, which consists of three experiments, designed based on the guidelines of Wohlin et al. (2012).

6.2.1 Evaluation setup

We define two types of users of the ALBA framework: developers and the end-users. Developers use the framework to create location-based apps, whereas end-users are users of the resulting apps. Note that developers can also be end-users. We use the GQM (Goal Question Metric) paradigm (Basili et al. 1994) to decompose the goals of the evaluation into specific research questions and metrics. We use the evaluation model for mobile apps proposed by Hussain (2013) to assess the effectiveness, efficiency, and satisfaction of ALBA's generated apps. We formulate the following research questions that we aim to answer by this case study.

RQ3. What is the effectiveness, efficiency, and satisfaction of the ALBA framework?
RQ4. What is the effectiveness, efficiency, and satisfaction of the resulting apps generated by the ABLA framework?
RQ5. What perception of satisfaction does the ALBA approach present, compared to the existing approaches?

6.2.2 Planning and data collection

The study was carried out in three experiments. In the first experiment, we designed an empirical study with 14 users to assess the usability of the ALBA modeling editor and its code generator. In the second experiment, we conducted an online user survey for evaluating the usability and quality of the apps generated by the ALBA framework, in which we review UniFy as a real-world app that is generated using the ALBA framework. Finally, in the third experiment, we designed an experiment, including two studies to compare the ALBA framework to the MIT App Inventor and the React Native, two well-known frameworks for developing mobile apps. In the following, we describe the activities that were performed for each experiment.

Experiment 1: ***Workshop to Build the UniFy App Using ALBA***

As introduced earlier, UniFy is the name chosen for the University of Isfahan guide app that was described in Sect. 2. Unify aims to help people finding various places on the university campus. The app has been built using the ALBA framework and is available on our research group web site[18] and in the Cafebazar Android market.[19]

[18] http://mdse.ui.ac.ir/tools/.
[19] https://cafebazaar.ir/app/ir.mohamadaligharaat.mdsegp.eng/?l=en.

It is worth mentioning that more than 95% of the UniFy app code, which represents 2000 lines of code, are generated automatically by the ALBA framework. Considering the productivity of average programmers that enables them to implement 100 lines of code per day (Kung 2013), the UniFy app needs 21 person-day effort to be built and tested. However, we modeled and made 95% line of code of UniFy from the requirements using the ALBA framework with only 3/4 person-day effort.

Figure 12 shows three pages of the UniFy app. Figure 12a shows the selected category fragment in which we have a grid recycler view (indicated by no. 1 in the figure) that handles the card views (no. 2). The number of columns for such pages can be set in the app model. In category fragment node, we have a property "colNumber" that is responsible for the number of columns of this grid view. It must be emphasized that, the developer can use any card view for showing the categories (no. 2) without any limitation. To do this, the developer should design a card view in Android Studio and set its Children (image views and text views) in the model. Hence, when the data is received from the server or local database, they will be fed into the card view to be shown to the user. Figure 12b illustrates the map fragment of the UniFy app, which has two primary views, map view (no. 4) and a horizontal list view (no. 5), which shows the list of the locations on the map. This figure also shows how subcategories are depicted in the app. They are placed in a scrollable tab bar above the map (no. 3). Figure 12c shows the details of each location. There could be six different choices with particular functionalities, namely bookmarking, navigating, calling, comments, and rating. On the figure, only the navigation button (no. 6) and the calling button (no. 7) are represented.

To evaluate the effectiveness, efficiency, and satisfaction of the modeling process of the ALBA framework for the developer, we conducted a user study with 14 participants. The participants are divided into three segments: undergraduate software engineering students, graduate software engineering students, and software developers. Table 5 shows the characteristics of the population used in this study. All three participant segments were divided into four categories regarding their expertise in the fields of MDE and Android app development. Those categories are as follows: (1) modeler, (2) Android developer, (3) both modeler, and Android developer, and (4) none.

The participants were asked to model an app, which is functionally similar to three pages of the UniFy app. Before we started the modeling part of the workshop, we spent 30 min to present the ALBA framework and the tool for the participants, and then we gave the participants the user stories and asked them to model it. User stories consisted of three pages of an app similar to the UniFy with a category selection page. By selecting a category, a map page would be opened which shows the locations that belong to the selected category, on the map. We had prepared the required web services, and their request and response parameters were described in detail.

We prepared a questionnaire consisting of 12 questions and asked the participants to answer the questions and rate features of the ALBA Framework. The questions and the statistics of participants' feedback are shown in Table 6.

The average rating to each question for each segment of participants is depicted in Fig. 13. Two participants were experienced in both MDD and Android app development. As it was expected, they firmly believed that ALBA is successful in the asked challenges, as they did not give anything less than 1 to any question. For the three software developers, the answers were 0 or 1 for all of the questions. Since they were more hesitant that ALBA can be used in the industry. Getting these ratings from them shows that they were convinced about the ALBA framework features. The answers given by the group which were only experienced in the MDD were less than those who were only experienced in Android development.

The drop of ratings from those who were experienced in both MDD and Android app development (blue line) to those who were only experienced in the Android app development (red line) is also worth to mention. This shows that those who are familiar with MDD and Android, have lower expectations from the framework in comparison with those who are only experienced in Android, and as the participants knowledge of coding became less, their rating of the ALBA features became less too.

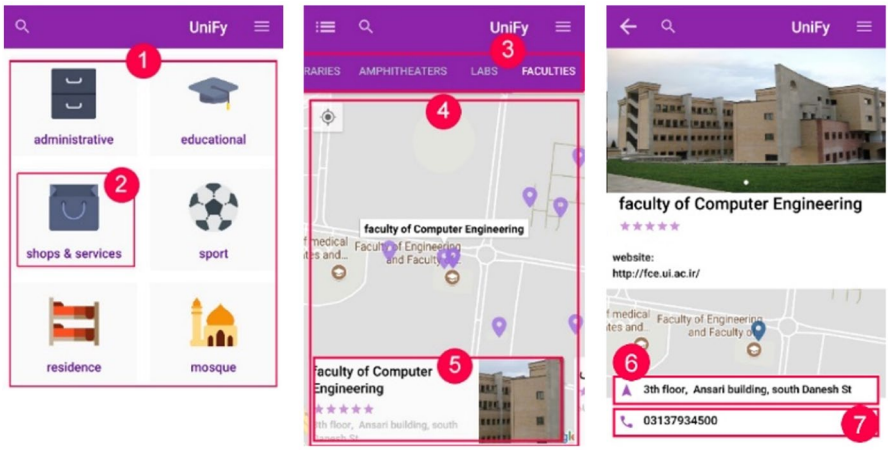
We compared these groups in more detail and figured out that those who were experienced in the Android programming were more likely to implement an app with the framework in less time. Figure 14 shows the time spent by each participant in producing the app using the ALBA framework. The average production time for modeling and implementing the UniFy app based on the requirements using the ALBA framework were about 1.28 h.

Experiment 2: ***Online User Survey on the UniFy App***

Reliable assessment of effectiveness, efficiency, and satisfaction as three quality characteristics of an app, requires to conduct a survey with users. Meanwhile, it is challenging to measure the usability or other quality characteristics for a mobile app in terms of absolute metrics. To handle this issue, we extended the usability evaluation model, proposed by Hussain (2013) for mobile apps, such that it considers the specific characteristics of the location-based apps. The proposed evaluation model for the location-based apps can be found in the "Appendix". This evaluation model, which follows the GQM approach, is designed carefully to assess the three mentioned quality characteristics (effectiveness, efficiency, and satisfaction) by refining the related goals into several questions which are measurable.

According to the questions of the proposed evaluation model, we conducted an online user survey for evaluating the effectiveness, efficiency, and satisfaction of the UniFy app, as the first app generated by the ALBA framework. UniFy is a real-world app with more than 500 active users, which has been used in the University of Isfahan for the last 2 years. We created an online questionnaire[20] and made it available to all users. The participants were asked to assign a value between 1 and 5 to each question. If a participant had no idea about a question, she/he had to select number 6. The ranking was as follows: (1) Strongly disagree,

[20] https://forms.gle/e4gHML8iVTBBWWXJ9.



(a) (b) (c)

Fig. 12 Different fragments of the UniFy App. (**a**) Categories, (**b**) map, and (**c**) location details

Table 5 The characteristics of participants in the evaluation workshop

Segment	No. of modeler	No. of android developers	No. of android developers and modelers	No. of not android developer and not modeler	Sum
Undergraduate students	0	3	1	3	7
Graduate students	3	0	1	0	4
Software engineers	0	3	0	0	3
No. of participants	3	6	2	3	14

(2) Disagree, (3) Neutral, (4) Agree, (5) Strongly agree, and (6) No idea. Twenty people took part in the survey. Table 7 summarizes the results of this survey.

The questions in Table 7 are grouped using horizontal lines. The first five questions designed to determine the effectiveness of the UniFy app in the context of simplicity and accuracy. The second five questions contain questions concerning the efficiency of the UniFy app, which designed mostly based on the location-based capabilities. The last group are questions that aim to evaluate the satisfaction of participants in general, as well as the safety goals.

Experiment 3: ***Comparing ALBA to other frameworks***

In this experiment, we aim to scrutinize the advantages and perception of satisfaction of ALBA framework compared to other mobile app development frameworks. The initial step for designing this experiment was to choose the frameworks to

Table 6 The answers to the questions of the workshop

Q#	Question	Strongly disagree (− 2)	Disagree (− 1)	Neurtral (0)	Agree (1)	Strongly agree (2)
Q1	The 30 min walkthrough was enough for learning all features available in the ALBA modeling editor	0	0	3	5	6
Q2	The terminologies used in the ALBA modeling editor are relevant to their task	0	0	6	6	2
Q3	The models of the apps are clear and understandable	1	1	6	5	1
Q4	ALBA is applicable for designing and implementing location-based Android apps	0	2	2	4	6
Q5	ALBA can make location-based Android app development easier in comparison with other methods	0	1	2	7	4
Q6	ALBA can reduce production time for location-based Android apps	0	1	2	5	6
Q7	The graphics used in the ALBA modeling editor are attractive for designer	1	1	1	8	3
Q8	The graphics used in the ALBA modeling editor are relevant to their task	0	1	2	6	5
Q9	The ALBA modeling editor is user-friendly	1	2	8	0	3
Q10	The ALBA modeling editor is a handy editor for modeling location-based Android apps	1	0	5	6	2
Q11	ALBA is easy to understand for someone without any experience in Android development	1	3	1	4	5
Q12	ALBA is easy to understand for someone without any experience in MDE	0	2	6	2	4

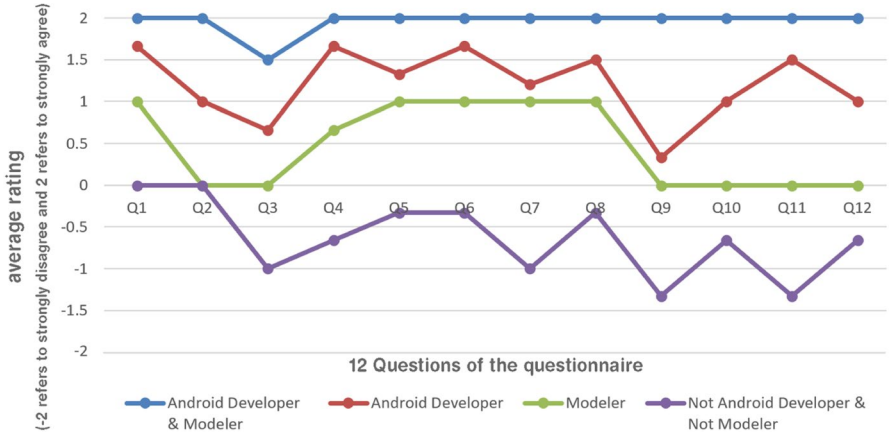


Fig. 13 Average rate to each question by participant segments

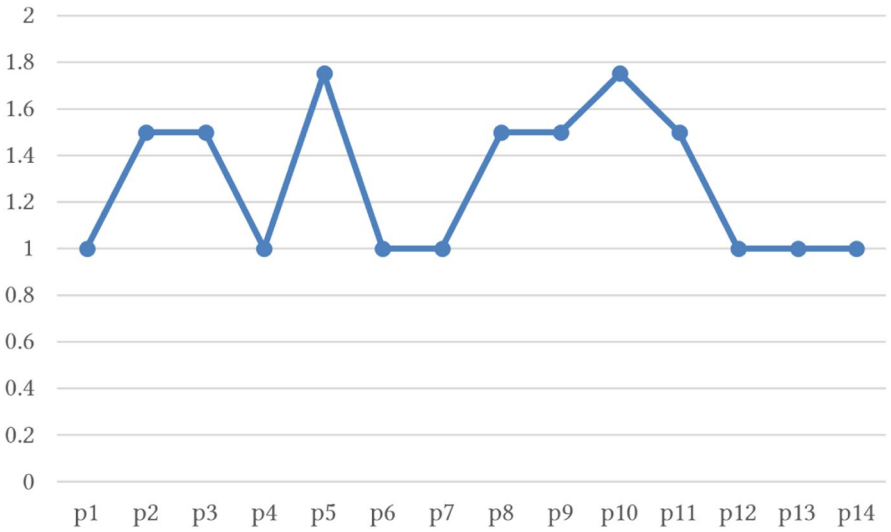


Fig. 14 Production time of each workshop participant

which the ALBA framework should be compared. We decided to go with MIT App Inventor and React Native because of their popularity and maturity (Dabit 2018; Kloss 2012). MIT App Inventor is the most comparable framework to our framework as it also supports the idea of model-based app development. Unlike MIT App Inventor, React Native is a framework that only supports manual coding. However, when comparing our framework with React Native, the comparison would allow us to evaluate the quality of the resulting apps. In the following, first we introduce the React Native and MIT App Inventor frameworks. Then we describe two comparison studies. The first comparison study focuses on the experts to develop a similar app,

Table 7 Results of online user survey to assess UniFy app

Q#	Question	Strongly disagree (1) (%)	Disagree (2) (%)	Neurtral (3) (%)	Agree (4) (%)	Strongly agree (5) (%)	No idea (6) (%)
Q1	How easy is it to install the application?	5	0	5	40	50	0
Q2	Is the application easy to learn?	5	15	15	35	30	0
Q3	Is it simple to find locations?	5	10	10	35	40	0
Q4	Is the application accurate?	5	10	10	45	30	0
Q5	Are many tasks successful at the first use?	0	5	25	35	30	5
Q6	Does the application respond quickly?	0	5	10	45	40	0
Q7	Does the application provide appropriate help?	10	10	10	45	25	0
Q8	Does the application provide appropriate menu?	0	0	15	45	40	0
Q9	Does the application provide location details?	5	5	5	50	35	0
Q10	Are few resources being used by the application?	0	5	25	20	30	20
Q11	Does the user enjoy while using the application?	0	0	5	55	40	0
Q12	Is the application secure to use?	0	10	30	25	15	20
Q13	Is the user happy with the interface?	0	5	15	50	30	0
Q14	Is the user familiar with the user interface?	0	5	5	55	35	0

and the second comparison study trains beginners to develop different real-world apps with all three frameworks. Finally, we report on the results of two comparison studies, in which the two types of participants were asked to quantitatively evaluate whether ALBA advances the quality in comparison to other frameworks?

React native: The React Native framework, with 73,969 stars, and being 11th most stared repository on GitHub, is the first cross-platform general-purpose framework. The React Native is the finest cross-platform framework among Ionic[21] and PhoneGap[22] frameworks (Quazi and Sinha 2018). However, React Native has some significant issues in performance and processing.

React Native does not support parallel multi-threading. Programmers often request Facebook to add this feature to the React Native framework for developers.[23] However, React Native has four threads: one UI thread, one JS thread, one Native Modules thread, and one Render thread (Android 5 and up) while native Android and ALBA enable the developer to implement a multi-thread app completely. Thus, the core components of the Java code used in ALBA are implemented using this multi-thread feature for async tasks for network and database communications. Overall, ALBA provides the programmer with full multi-threading capability due to its reliance on the Java multi-threading mechanism.

Background processing is another issue in React, since it can affect the user experience. This deals with how the framework runs tasks when the app is running in the background. React Native requires some native code for implementing this feature. In this case, React Native cannot help as a cross-platform framework. It is worth to mention that this feature called "headless JS" and is only available for Android release.[24] Developers are asking the React Native community for implementing this feature for iOS generated apps as well.[25]

MIT app inventor: MIT App Inventor is a web-based visual programming environment that facilitates designing both the business logic and GUI layout of Android apps. The objective of the tool is to teach kids and assist inexperienced people in building simple mobile apps. MIT App Inventor has two main parts as follows: (1) *designer* which is for designing the UI of the app, and (2) *blocks* which is a drag-and-drop environment helping the user to implement the business logic of the app using connectable puzzles. While the framework is very helpful for teaching purposes and the target domain of the generated apps is general, yet, it is incomplete for building serious apps. More importantly, the website does not provide its users with the app code. This is while ALBA provides users with app code and the user can benefit from Android Studio features. Overall, if a minor change is required in the app designed by the website and it is not supported, the user will lose all of the effort investigated to implement the app due to the lack of completeness. Take layouts (a view item which orients the child items, e.g., linear layout) as an example,

[21] https://ionicframework.com.
[22] https://phonegap.com.
[23] https://react-native.canny.io/feature-requests/p/parallel-multithreading--workers.
[24] https://facebook.github.io/react-native/docs/headless-js-android.
[25] https://react-native.canny.io/feature-requests/p/headless-js-for-ios.

the MIT App Inventor only supports some Horizontal, Table, and Vertical Arrangements. While layouts which are supported by the Android studio is more than these layouts.

Both MIT App Inventor and React Native are general purpose frameworks, while, ALBA has a narrower domain. To the best of our knowledge there is no DSML for generating location-based apps. Subsequently, we chose these two languages for our comparison study.

Comparison study 1: To identify the advantages of the ALBA framework for developing a location-based app by an expert, we carried out an experiment to compare the apps generated by ALBA with the results of React Native and MIT App Inventor frameworks. The experiment was conducted with the participation of three developers, each of them has expertise in one of the frameworks: ALBA, React Native, and MIT App Inventor. In this experiment, we asked three developers to build the UniFy app. The first developer, who had 1 year of experience with ALBA and 1 year of experience on location-based apps, generated the app using ALBA. The second developer, who had 6 months of experience with MIT App Inventor and 1 year of experience on location-based apps, developed the app using MIT App Inventor. The third developer, who had 4 years of experience with React Native and 1 year of experience on location-based apps, developed the app using React Native.

We gave the participants a list of full requirements of the UniFy real-world app, introduced in Sect. 6.1.2. Then, we asked the participants to develop the UniFy app using a framework that they are most familiar with. Once they accomplished their tasks, they have been asked to fill out a questionnaire. Finally, we installed all three developed apps on one mobile phone to compute the performance metrics for performing an identical scenario on each of these apps. Figure 15 shows three main pages of the developed apps. We also present summaries of the data gathered in our experiment for each framework in Table 8.

In order to find the functional correctness of the apps that were generated in this experiment, we performed different scenarios to test the functionality of each app. In these scenarios, we consider a functionality as correct if it worked without any errors. Table 8 shows that only ALBA has met all the requirements and achieved 100% functional correctness, while React Native and MIT App Inventor have managed to cover 96% and 48% of the requirements, respectively. In addition, the generated app using ALBA has less average response time compared to two other frameworks. React Native is a code-driven approach that has the longest *development time* and highest lines of code *(LOC)*. In contrast, the MIT App Inventor used a model-based approach that has led to the least *development time*. However, the MIT App Inventor only covered half of the requirements. Among the generated apps, the one generated with ALBA had the smallest *APK size* and *App storage*. Also, for a specific scenario, all three apps used the same amount of *memory* and *battery*.

Comparison study 2: To identify the advantages of the ALBA framework in developing location-based apps by the beginners, we carried out another experiment in four working sessions that was conducted with three students of the 8th semester of the Computer Engineering program at the University of Isfahan. They had finished basic courses in computer science, such as Introduction to Java Programming, Object-Oriented Analysis and Design, and Software Engineering. All

three students had enough knowledge in programming and modeling, but had no prior experience in Android development and location-based apps. They also had no prior knowledge about any of the three frameworks, ALBA, MIT App Inventor, and React Native.

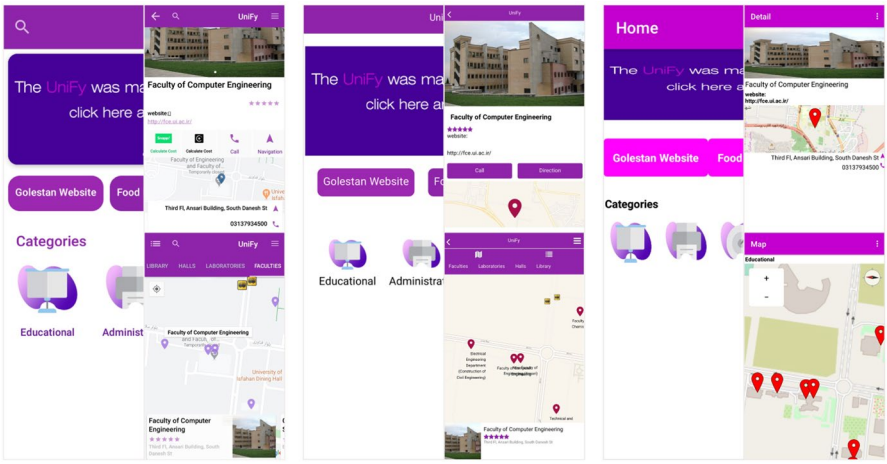
The first session (100 min long) was divided into two stages that focused on the ALBA framework. In the first stage (30 min long), we presented ALBA to the participants and trained them how to model and generate apps using the ALBA framework. In the second stage (70 min long), the participants received a list of requirements for a simplified version of the MedIUM app, which consisted of 10 requirements. Then, they started the modeling process and continued with the code generation process to prepare the corresponding APK file. Afterward, we investigated the correctness of the generated apps considering the list of initial requirements. All the requirements were satisfied, hence, we calculated the functional correctness of 100% for all the participants.

The second session (95 min long) was divided into two stages that focused on the MIT App Inventor framework. In the first stage (30 min long), we presented the MIT App Inventor to the participants and trained them how to generate apps using the MIT App Inventor. In the second stage (65 min long), the participants received a list requirements for the simplified version of the Baharestan app, consisting of 10 requirements. They modeled the app using MIT App Inventor and generated the corresponding APK file. Afterward, we investigated the correctness of the generated apps considering the list of the initial requirements. We calculated the functional correctness of 80% for all the participants.

The third session (285 min long) was divided into two stages that focused on the React Native framework. In the first stage (90 min long), we trained the participants how to develop apps using the React Native framework. In the second stage (195 min long), they received a list of requirements for the Covid-19 dashboard app, which consisted of six requirements. They developed the corresponding APK file, and then we investigated the correctness of the generated apps considering the list of the initial requirements. We calculated the functional correctness of 100% for all the participants.

The fourth session (410 min long) was divided into three stages. In this session, we investigated the generation of the simplified version of the UniFy app consisted of 10 requirements using all three frameworks. In the first stage (60 min long), the participants developed the introduced app using the ALBA framework. Considering the initial requirements, the average functional correctness was 100%. In the second stage (55 min long), they generated the simplified UniFy app using the MIT App Inventor. Two participants had 70% functional correctness and the third one had 80% functional correctness. In this stage, we consider the average functional correctness (73%) as a notable functional correctness. Finally, in the third stage (295 min long), the participants developed the defined app using React Native, and the average functional correctness was 100%.

At the end of this experiment, the participants were asked to answer a questionnaire for any of the three frameworks. The questionnaire focused on the measuring the satisfaction of the participants on working with a framework, concerning a variety of the system usability aspects (Bangor et al. 2009). In the questionnaire, each



(a) (b) (c)

Fig. 15 Three main pages of UniFy developed with (**a**) ALBA; (**b**) react native; (**c**) MIT app inventor

question has a score ranging from 1 to 5, where value closer to 5 indicates higher value of satisfaction. The data collected from this experiment for each framework, including the average satisfaction score, are presented in Table 9.

6.2.3 Analysis of the result

Having obtained the data presented in the previous section, we proceed with analyzing the results of experiments and draw some conclusions for each research question of Sect. 6.2.1.

RQ3: What is the effectiveness, efficiency, and satisfaction of the ALBA framework?

Considering the results obtained from the first experiment (Table 6), we observe that all participants completed the app development process using ALBA. Also, as depicted in Table 6, 9 out of 14 participants agreed that ALBA is easy to understand by developers who are not very experienced in Android development (Q11). In addition, about 80% of the participants indicated that ALBA makes location-based Android app development easier. According to these answers, we may safely draw the conclusion that ALBA is effective for developing location-based Android apps. Also 8 out of 14 participants confirmed that ALBA editor is a handy editor for modeling, and around 80% of them stated that ALBA could reduce productivity time for location-based apps. Thus, the efficiency of ALBA framework is predictable. Based on the answers of questions Q7 and Q8 in Table 6, approximately 79% of the participants indicated that graphical elements that are used in ALBA are attractive and satisfactory. However, only 3 out of 14 participants agreed that ALBA modeling editor is user-friendly (Q9). More precisely, to investigate the satisfaction of ALBA framework, only two participants reported that models of app are not clear(Q3), and 10 out of 14 participants confirmed that ALBA is applicable for designing and

Table 8 Results of the first comparison study to develop the UniFy app by experts in different frameworks

Framework	% Covered req. (funct. correctness)	Avg. resp. time (ms)	Dvlpmt. time (h)	#LOC	APK size (MB)	Storage (MB)	Memory (MB)	% Battery usage (%)
ALBA	(25/25) 100%	~ 720	~ 10	~ 2000	4.59	25	96	< 1
React Native	(24/25) 96%	~ 810	~ 40	~ 3400	9.61	28	97	< 1
MIT App Inventor	(12/25) 48%	~ 940	~ 7	–	5.37	26	96	< 1

Table 9 Results of the second comparison study to develop simplified version of the UniFy app by beginners in different frameworks

Framework	% Covered requirements (average functional correctness) (%)	Training time (min)	Average development time (min)	Average satisfaction score (# of 5)
ALBA	100	30	~ 60	~ 3.97
MIT app inventor	73	30	~ 55	~ 1.95
React native	100	90	~ 295	~ 3.03

implementing location-based Android apps (Q4). Overall, the participants were satisfied with working with the ALBA framework.

RQ4: What is the effectiveness, efficiency, and satisfaction of the resulting apps generated by the ABLA framework?

The results of this experiment are shown in Table 7. As it is indicated in questions Q1 and Q2, 90% of the participants agreed that UniFy is installed easily, and 65% of them reported that Unify has been easy to learn and use. Regarding the answers to questions Q3 and Q4, we realize that UniFy, as a real-world result of ALBA was recognized as an effective app by 75% of the participants. Also, 65% of the participants stated that UniFy successfully did many tasks at the first use (Q5). These answers confirm that UniFy, as a resulting app of ALBA, was an effective app for end-users. Based on the answers of the participants to question Q6, 85% indicated that UniFy responds efficiently to their needs. As it is indicated in questions Q8 and Q9, 85% of the participants confirmed that the UniFy app provide appropriate menu and location details, whereas 75% of the participants were satisfied with the provided help for app (Q7), only 50% of the participants stated that the app used a few resources (Q10). Nevertheless, these results are comprehensible, because few people know about resource consumption by apps. Around 90% of the participants reported that they are familiar with UniFy user interface (Q14), and 80% indicated that they are happy with the app interface (Q13). The fraction of participants, who consider the app as a secure app, is only 40% (Q12), although 95% stated that they enjoy while using UniFy (Q11). We believe that the reason for the low number 40% (Q12) is the user uncertainty about the information non-disclosure. As a result, we conclude that the participants were satisfied with the UniFy app; however, the relative safety criteria as a necessary but non-functional feature for location-based app, was low for this study and should be considered in the extensions of the ALBA framework.

RQ5: What perception of satisfaction does the ALBA approach present, compared to the existing approaches?

As depicted in Table 8, when experts on three different frameworks tried to develop the same location-based app, only ALBA was able to cover 100% of the requirements. ALBA advances the development of location-based apps by benefiting from the advantages of both model-driven and code-driven approaches. ALBA increases productivity four times more than React Native, a purely code-centric platform, and covers more requirements than MIT App Inventor, a model-based framework. In addition, ALBA generated apps with a small size, more functional

correctness rate, and faster response time, while maintaining the same memory and battery usage as the other two frameworks, which indicate better quality for the ALBA outputs. Furthermore, Table 9 shows that for generating location-based apps, the ALBA framework is a better choice for beginners who had no prior knowledge in location-based app development. Regarding the average satisfaction score, ALBA obtained a higher level of satisfaction of the participants, where they were able to successfully develop all the app requirements. The average functional correctness for React Native was similar to ALBA. However, the average development time of ALBA was about 20% of the required time for React Native. Also, more training was necessary to learn how to generate location-based apps with React Native. In addition, MIT App Inventor had lower level of satisfaction and functional correctness rate. As a result, we conclude that ALBA has obtained an acceptable level of satisfaction of participants with similar knowledge and training. It was accepted among the participants and proposed as the best solution in comparison to other frameworks that can be used for generating location-based apps.

6.3 Threats to validity

In this section, we explain threats to validity of this study.

Construct validity: Construct validity threats concern the accuracy of the observations with respect to the theory. We designed ALBA domain-specific modeling language to support the features required for location-based Android app development. To define the ALBA DSML, we followed the approach introduced by Mernik et al. (2005). This approach consists of five phases: decision, analysis, design, implementation, and deployment. Thus, we argue that there is no threat to the construct validity.

Internal validity: Internal validity threats concern the factors that might influence our results. The selection of the five apps used to define ALBA's features is one possible threat. We may have missed relevant features. We mitigated this threat by examining apps that are widely used. Two of the authors also reviewed the features of the selected apps thoroughly to ensure that they fit this study. Another threat concerns the way we implemented ALBA using Eclipse modeling tools. An error may have occurred during implementation. To mitigate this threat, the first two authors tested and reviewed the code. We also made ALBA available online for the community to use it and report bugs.

Reliability validity: Reliability validity concerns possibility of replicating this study. We studied five location-based apps but we cannot claim that these are representative of all existing apps. Based on the review that we conducted, we argue that these apps are representative of existing location-based apps. Moreover, we put online the ALBA meta-model and implementation to allow other researchers to reproduce this study by generating other location-based apps.

External validity: External validity is related to the generalizability of the results. Firstly, we evaluated ALBA by automatically generating a location-based app for the campus of the University of Isfahan and by conducting a user study. While the results show promising results, in order to generalize these results, we selected and

generated apps for three more real-world cases from different domains. This demonstrates the ALBA framework can be used in the development of different location-based apps from different domains. However, we need to conduct additional user studies, preferably with experienced app developers to further assess the usability and learnability of ALBA and its effectiveness in improving productivity.

7 Related work

The study on cross-platform tools for developing mobile apps has been an active research topic in recent years (Benouda et al. 2016; Bernaschina et al. 2017; Usman et al. 2017). Among 21 tools surveyed by Tufail et al. (2018) and our survey, we found that MDD tools for automatic generation of mobile apps have been used in many domains such as management, games, e-commerce, field force automation, and so on. However, to the best of our knowledge, there is no MDD framework that is tailored to the automatic generation of Android location-based apps. These apps require working with GPS sensor battery efficiently, configuration of map libraries, and integration of map with other parts of the app.

Parada and De Brisolara (2012) proposed a tool for developing Android apps using their previous tool called (Parada et al. 2011). The proposed tool can generate Java code from UML class diagrams and sequence diagrams. This is fine in implementing structures of Java Android classes since the UML diagrams are familiar for domain experts. Their tool requires extensive effort modeling an Android app behavior using sequence diagrams. In contrast, based on our evaluation results, the ALBA framework can automatically generate approximately 95% of the Unify app, which contains most of the domain concepts in a very short time, because most of the implementation details are encapsulated and they are not visible to the framework user.

Heitkötter et al. (2013) proposed an approach named MD^2 for model-driven cross-platform app development. MD^2 is developed in close collaboration with industry, and the language is designed based on the MVC (Model-View-Controller) pattern. However, it is not clear if the tool supports any operation other than Create, Read, Update, and Delete (CRUD) operations. MD^2 generates the server-side code, limiting the user to servers that are required to Java EE app, which is the only platform supported by MD^2. The MAML framework designed by Rieger and Kuchen (2018) is based on MD^2, which represented a graphical DSL that solves the trade-off between technical complexity and graphical oversimplification. ALBA focuses on location-based features such as integration of the business logic with map libraries (e.g., Google Map) that go beyond simple CRUD operations. In addition, with ALBA, the user can specify the endpoint for receiving data from server and communicate with web services implemented by any server-side language (e.g., PHP, Microsoft .net, Python, Ruby, Java, Scala) and not only Java EE.

Vaupel et al. (2018) presented a modeling language that is considered as a native Android and iOS app generator with the support of role-based app variability. They proposed three meta-models for modeling the app layout, data, and app behavior. The layout meta-model is incomplete, and some user interface components are not

supported (e.g., Navigation drawer or Tab bar). Moreover, obtaining mobile location from the GPS sensor is not supported, while ALBA supports it while being cautious about energy efficiency.

Applause[26] is another cross-platform toolkit for generating native mobile apps for major mobile platforms (iOS, Android, Windows Phone 7). The toolkit uses a DSL implemented by Xtext on Eclipse to transform app models to apps. Thanks to Xtext, the Eclipse plugin has an IDE features, e.g., auto-completion and error-highlighting. The next version of this toolkit, Applause 2, is under development, while its last update occurred in 2015 (Gaouar et al. 2015). It is not clear whether Applause supports GUI generation, while ALBA generates all of the model, controller, and view classes together.

Mobl, introduced by Hemel and Visser (2011), is an open-source language used for modeling Android and iOS apps using a DSL, which can be compiled to JavaScript, HTML, and CSS code. This language has a good IDE support. Similarly, ALBA modeling editor prevents user errors in the modeling phase. Mobl does not provide native code generation, which can lead to slow app performance in location-based apps.

Among the hybrid frameworks for developing mobile apps, the most popular MDD approach for generating mobile apps is Ionic Creator.[27] The Ionic Creator is a website designed for generating Android and iOS apps using the Ionic framework. It is useful for developing static apps using a website with a drag-and-drop environment. However, implementing dynamic apps that obtain their information from an end-point or a sensor requires the modeler to write JavaScript code. Similar to ALBA, navigation patterns are supported, namely the navigation drawer and tab bar. But there is no option for communication with the server except implementing the code in the website code pane, while ALBA supports complete code generation to connect to any server API using the ALBA modeling tool without writing a single line of code. Also, Ionic, like other cross-platform technologies, face efficiency issues (Hemel and Visser 2011).

MIT App Inventor is a website for designing both business logic and views of Android apps. The tool has a drag-and-drop environment for modeling the app logic. Commands are puzzle shaped, and the user can understand which command to choose based on the free slot in the puzzles. This tool is handy for students wish to learn Android programming. The main limitation of MIT App Inventor is that it covers a few number of Android view items. The ALBA framework is dedicated to the location-based app domain and offers more capabilities than the MIT App Inventor. ALBA DSML is more abstract and can assist users to achieve their goal faster. Moreover, ALBA provides the user with the app source code, which is something that MIT App Inventor does not support.

Interaction Flow Modeling Language (IFML[28]) is a model-driven tool for designing web app user interfaces that respond to user actions with respect to a control flow. The tool has become a standard for integrating the front-end design in models science 2015 by Object Management Group[29] (OMG). It is not clear whether the

[26] https://github.com/applause/applause.
[27] https://creator.ionic.io/.
[28] https://www.omg.org/spec/IFML.
[29] https://www.omg.org.

access to device-specific features such as GPS sensors is guaranteed by IFML, but ALBA generates native code for communicating with GPS sensors.

8 Conclusion and future work

In this paper, we introduced a framework called ALBA, which consists of a DSML, a modeling tool, and a code generator plugin for Eclipse. DSML supports many of the common features of the popular location-based apps. The modeling tool has been created in order to help the users model their apps. The plugin transforms the models to Java native code as logic and XML layout code as view.

Having a framework that generates the app code in the structure of the Android projects, gives a boost to the development of these app types. As we surveyed, there is no DSML in the field of location-based apps. We showed how ALBA could enhance the development process of Android location-based apps. We evaluated our framework in the workshop from the audience perspective and compared the code of a similar app generated with best practices of other approaches. The participants of the evaluation workshop consisted of developers, model-driven engineers, and early bachelor computer engineering students. Generally, they were satisfied with the framework and were also surprised to generate an Android app in less than 2 h. We also conducted another case study research in which we assessed the usability of the ALBA framework and its generated app with real users. Furthermore, we investigated the development of UniFy app using ALBA in comparison to React Native, MIT App Inventor frameworks. The results of the evaluation are promising both in terms of the applicability and usability of the framework and the quality of the generated apps.

Going forward, first, we plan to define more features in the DSML and add compiling feature to the ALBA framework. Second, we aim to support iOS platform and implement iOS transformation code such that the iOS users can take advantage of the generated apps. Although some changes regarding advancement in software development in Android is needed in the future, while, we keep supporting the previously designed models and let them be valid. Finally, we need to conduct further studies to see how ALBA can be used in the context of complex apps with diverse application logic and in which location-based services are not the primary focus (e.g., TripAdvisor and Expedia).

Appendix: The usability evaluation model for location-based apps

Table 10 presents the usability evaluation model for location-based mobile apps based on the GQM approach. In this table, effectiveness, efficiency, and satisfaction are defined as three main quality characteristics to focus on evaluating the usability based on ISO 9241-11. According to quality characteristics, six goals are established. Then, the questions are described to assess each goal. Finally, a set of metrics based on the location-based features are explained to collect information to answer related questions.

Table 10 Evaluation model for location-based mobile applications. (Adapted from Hussain 2013)

Quality characteristics	Goals	Questions	Metrics
Effectiveness	Simplicity	How easy is it to install the application?	Satisfaction with the installation process
			Satisfaction with help provided
			Time taken to install
			The number of interactions while installing the application
		Is the application easy to learn?	Satisfaction with help provided
			Number of mistakes while learning
			Time taken to learn
		Is it simple to find locations?	Number of errors while searching the location of interest
			Time taken to find a location
			Satisfaction with search UI
	Accuracy	Is the application accurate?	Satisfaction with output
			Number of errors
		Are many tasks successful at the first use?	Time taken to complete the task
			Numbers of tasks successful at the first use

Table 10 (continued)

Quality characteristics	Goals	Questions	Metrics
Efficiency	Time Taken	Does the application respond quickly?	Time taken to start the application
			Time taken to load map
			Time taken to respond location search
			Time taken to show location details
			Time taken to connect to the network
	Features	Does the application provide appropriate help?	Satisfaction with help provided
		Does the application provide appropriate menu?	Satisfaction with tabs, navigations, and categories
		Does the application provide location details?	Satisfaction with location details
			Number of detail fields provided for a location
		Are few resources being used by the application?	Percentage of battery used during the installation
			Percentage of battery used per hour
			Percentage of CPU used per hour
			Percentage of RAM used per hour

Table 10 (continued)

Quality characteristics	Goals	Questions	Metrics
Satisfaction	Safety	Does the user enjoy while using the application?	Stress
			Enjoymen
			Satisfaction with contents
		Is the application secure to use?	Satisfaction with user authentication
			Satisfaction with user information non-disclosure
			Safety while walking
			Safety while driving
	Attractiveness	Is the user happy with the interface?	Easy to find help
			Satisfaction with hints
			Satisfaction with the map
			Satisfaction with text
			Satisfaction with system navigation
		Is the user familiar with the user interface?	Satisfaction with interface graphics
			Satisfaction with interface arrangement
			Satisfaction while learning

References

Alfraihi, H., Lano, K., Kolahdouz-Rahimi, S., Sharbaf, M., Haughton, H.: The impact of integrating agile software development and model-driven development: a comparative case study. In: International Conference on System Analysis and Modeling, pp. 229-245. Springer, Copenhagen (2018)

Bangor, A., Kortum, P., Miller, J.: Determining what individual SUS scores mean: adding an adjective rating scale. J. Usability Stud **4**, 114–123 (2009)

Basili, V.R., Caldiera, G., Rombach, H.D.: The goal question metric approach. In: Encyclopedia of Software Engineering, pp. 528–532 (1994)

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

Bernaschina, C., Comai, S., Fraternali, P.: IFMLEdit. org: model driven rapid prototyping of mobile apps. In: Proceedings of the 4th International Conference on Mobile Software Engineering and Systems, pp. 207–208. IEEE Press, Buenos Aires (2017)

Brambilla, M., Cabot, J., Wimmer, M.: Model-Driven Software Engineering in Practice, 2nd edn. Morgan & Claypool, San Rafael (2017)

Clement, J.: Number of apps available in leading app stores (2019). https://www.statista.com/statistics/276623/number-of-apps-available-in-leading-app-stores/. Accessed 11 Nov 2019

Combemale, B., France, R., Jézéquel, J.M., Rumpe, B., Steel, J., Vojtisek, D.: Engineering Modeling Languages: Turning Domain Knowledge into Tools. Chapman and Hall/CRC, London (2016)

Dabit, N.: React Native in Action. Manning Publications Company, New York (2018)

Dalmasso, I., Datta, S.K., Bonnet, C., Nikaein, N.: Survey, comparison and evaluation of cross platform mobile application development tools. In: 9th International Wireless Communications and Mobile Computing Conference, pp. 323–328. IEEE, Sardinia (2013)

Dehlinger, J., Dixon, J.: Mobile application software engineering: Challenges and research directions. In: Workshop on Mobile Software Engineering, pp. 29–32. Lille (2011)

Dillon, A.: The Evaluation of Software Usability. Taylor and Francis, London (2001)

Ferraro, R., Aktihanoglu, M.: Location-Aware Applications. Manning Publications Co, New York (2011)

Gaouar, L., Benamar, A., Bendimerad, F.T.: Model driven approaches to cross platform mobile development. In: Proceedings of the International Conference on Intelligent Information Processing, Security and Advanced Communication, pp. 19–23. ACM, Batna (2015)

Heitkötter, H., Majchrzak, T.A., Kuchen, H.: Cross-platform model-driven development of mobile applications with MD^2. In: Proceedings of the 28th Annual ACM Symposium on Applied Computing, pp. 526–533. ACM, Coimbra (2013)

Hemel, Z., Visser, E.: Declaratively programming the mobile web with Mobl. In: Proceedings of the 2011 ACM International Conference on Object Oriented Programming Systems Languages and Applications, pp. 695–712. ACM, Portland (2011)

Holst, A.: Smartphone users worldwide 2016–2021 (2019a). https://www.statista.com/statistics/330695/number-of-smartphone-users-worldwide/. Accessed 11 Nov 2019

Holst, A.: Global market share held by leading smartphone vendors from 4th quarter 2009 to 3rd quarter (2019b). https://www.statista.com/statistics/271496/global-market-share-held-by-smartphone-vendors-since-4th-quarter-2009/. Accessed 11 Nov 2019

HoseinDoost, S., Adamzadeh, T., Zamani, B., Fatemi, A.: A model-driven framework for developing multi-agent systems in emergency response environments. Softw. Syst. Model. **18**(3), 1985–2012 (2019)

Hussain, A.: A metric-based evaluation model for applications on mobile phones. J. Inf. Commun. Technol. **12**, 55–71 (2013)

Kang, K.C., Cohen, S.G., Hess, J.A., Novak, W.E., Peterson, A.S.: Feature-oriented domain analysis (FODA) feasibility study (No. CMU/SEI-90-TR-21). Carnegie-Mellon Univ Pittsburgh Pa Software Engineering Inst (1990)

Kloss, J.H.: Android Apps with App Inventor: The Fast and Easy Way to Build Android Apps. Addison-Wesley, Boston (2012)

Kolovos, D., Rose, L., Paige, R., García-Domínguez, A.: The Epsilon Book. Eclipse (2010)

Kolovos, D.S., García-Domínguez, A., Rose, L.M., Paige, R.F.: Eugenia: towards disciplined and automated development of GMF-based graphical model editors. Softw. Syst. Model. **16**(1), 229–255 (2017)

Kung, D.: Object-Oriented Software Engineering: An Agile Unified Methodology. McGraw-Hill Higher Education, New York (2013)

Kurtev, I., Bézivin, J., Jouault, F., Valduriez, P.: Model-based DSL frameworks. In: Companion to the 21st ACM SIGPLAN Symposium on Object-oriented Programming Systems. Languages, and Applications, pp. 602–616. ACM, New York (2006)

Meirelles, P., Aguiar, C.S., Assis, F., Siqueira, R., Goldman, A.: A students' perspective of native and cross-platform approaches for mobile application development. In: International Conference on Computational Science and Its Applications. Lecture Notes in Computer Science, vol. 11623, pp. 586–601. Springer, Cham (2019)

Mernik, M., Heering, J., Sloane, A.M.: When and how to develop domain-specific languages. ACM Comput. Surv. CSUR **37**(4), 316–344 (2005)

Oh, Y.J., Park, H.S., Min, Y.: Understanding location-based service application connectedness: model development and cross-validation. Comput. Hum. Behav. **94**, 82–91 (2019)

Okediran, O.O., Arulogun, O.T., Ganiyu, R.A., Oyeleye, C.A.: Mobile operating systems and application development platforms: a survey. Int. J. Adv. Netw. Appl. **6**(1), 2195–2201 (2014)

Parada, A.G., Siegert, E., De Brisolara, L.B.: Generating Java code from UML class and sequence diagrams. In: Brazilian Symposium on Computing System Engineering, pp. 99–101. IEEE, Florianopolis (2011)

Parada, A.G., De Brisolara, L.B.: A model driven approach for Android applications development. In: Brazilian Symposium on Computing System Engineering, pp. 192–197. IEEE, Natal (2012)

Quazi, F.U.R., Sinha, N.: Android-platform based determination of fastest cross-platform framework. Int. J. Comput. Sci. Mob. Comput. **7**(9), 1–12 (2018)

Raveh, J.: Use of location-based services in 2019 (2019). https://www.theneura.com/use-of-location-based-services-in-2019/. Accessed 11 Nov 2019

Rieger, C., Kuchen, H.: A process-oriented modeling approach for graphical development of mobile business apps. Comput. Lang. Syst. Struct. **53**, 43–58 (2018)

Runeson, P., Höst, M.: Guidelines for conducting and reporting case study research in software engineering. Empir. Softw. Eng. **14**(2), 131–164 (2009)

Steinberg, D., Budinsky, F., Merks, E., Paternostro, M.: EMF: Eclipse Modeling Framework. Pearson Education, London (2008)

Sydow, L.: Record levels of app downloads and app store consumer spend in Q4 2017 (2018). https://www.statista.com/statistics/330695/number-of-smartphone-users-worldwide/. Accessed 11 Nov 2019

Tufail, H., Azam, F., Anwar, M.W., Qasim, I.: Model-driven development of mobile applications: a systematic literature review. In: 9th Annual Information Technology. Electronics and Mobile Communication Conference, pp. 1165–1171. IEEE, Vancouver (2018)

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

Vaupel, S., Taentzer, G., Gerlach, R., Guckert, M.: Model-driven development of mobile applications for android and iOS supporting role-based app variability. Softw. Syst. Model. **17**(1), 35–63 (2018)

Wohlin, C., Runeson, P., Höst, M., Ohlsson, M.C., Regnell, B., Wesslén, A.: Experimentation in Software Engineering. Springer, Berlin (2012)

Zolotas, C., Diamantopoulos, T., Chatzidimitriou, K.C., Symeonidis, A.L.: From requirements to source code: a model-driven engineering approach for RESTful web services. Autom. Softw. Eng. **24**(4), 791–838 (2017)

Authors and Affiliations

Mohammadali Gharaat[1] · Mohammadreza Sharbaf[1] · Bahman Zamani[1] · Abdelwahab Hamou-Lhadj[2]

Mohammadali Gharaat
mohamadali.gharat@mehr.ui.ac.ir

Mohammadreza Sharbaf
m.sharbaf@eng.ui.ac.ir

Abdelwahab Hamou-Lhadj
wahab.hamou-lhadj@concordia.ca

[1] MDSE Research Group, Department of Software Engineering, University of Isfahan, Isfahan, Iran

[2] Department of Electrical and Computer Engineering, Concordia University, Montreal, QC, Canada