The Journal of Systems and Software 161 (2020) 110489

Contents lists available at ScienceDirect

The Journal of Systems and Software

journal homepage: www.elsevier.com/locate/jss

A model-driven approach for the development of native mobile applications focusing on the data layer

Manuel Núñez[a,*], Daniel Bonhaure[a], Magalí González[a], Luca Cernuzzi[a]

Department of Electronics and Computer Science Engineering (DEI), Catholic University "Nuestra Señora de la Asunción", Tte. Cantaluppi y G. Molinas, 1366 Asunción, Paraguay

ARTICLE INFO

Article history:
Received 18 February 2019
Revised 24 November 2019
Accepted 3 December 2019
Available online 4 December 2019

Keywords:
Model-Driven Development
Model-Driven Mobile Development
Mobile applications
Data persistence
Data provider
Data layer

ABSTRACT

The data layer access design is a critical task for mobile applications which need constant access to remote data, making them available offline in case of network connectivity problems. Moreover, the variety of mobile operating systems and platforms (fragmentation phenomenon) that handles data storage differently affects the portability of mobile applications. This issue suggests the use of Model-Driven Development (MDD) approach that may ease the smartphone application development for different platforms. Therefore, we propose an extension of the Model-Oriented Web Approach (MoWebA) for the development of native mobile applications focusing on the data layer. MoWebA Mobile (as we name the proposal) covers data persistence concepts to achieve offline applications in case of network connectivity problems. It defines meta-models and the Architecture-Specific Model (ASM) for data persistence and data provider to design the data sources of mobile applications. As well, we propose transformation rules for the generation of Android and Windows Phone applications. The MoWebA Mobile process was illustrated by means of modeling, design, and development of a typical mobile application. Despite the learning curve of the approach, the first evaluation suggests that MoWebA Mobile has the potential for supporting the mobile design applications considering the persistence of data.

1. Introduction and motivation

The development of mobile applications implies taking account of the usually available limited resources like memory, battery, network connectivity, and others (Balagtas-Fernandez and Hussmann, 2008; Barnett et al., 2015c). Most of the mobile applications are in constant access to remote data. The domain of business applications includes those applications, defining them as form-based and data-driven applications interacting with back-end systems (Majchrzak et al., 2015; Heitkötter et al., 2015). In this scenario, not considering the network connectivity dimension could lead to applications with insufficient usability, limited or unusable applications in offline mode. In this study, we emphasize the case of the network connectivity instability manifested in two significant concerns: (i) mobile devices, especially smartphones, that have a constant variation of network connectivity (3G, 4G, and wireless networks, among others), almost imperceptible to the user; and, (ii) the unexpected loss of connection.

To deal with the mentioned concerns, Barnett et al., 2015c and the *Microsoft Application Architecture Guide* (MAAG) (Patterns, 2009) recommend using data caching. They describe it as data persistence in memory or on disk that enhances the application's performance and its User Interface (UI) *responsiveness*[1], making it manipulable in offline mode. According to MAAG (Patterns, 2009), in a multi-layered architecture[2], it is possible to implement the data caching in the presentation layer or the data layer (more details in Section 2.1). The data layer, aside from the data persistence handling, defines data sources – i.e., data providers (Ribeiro and Silva, 2014) – for the mobile application[3] Thus, in this study, we propose to cover data persistence concepts, while considering part of

* Corresponding author.
E-mail addresses: manuel.nunez@uc.edu.py (M. Núñez), daniel.bonhaure@uc.edu.py (D. Bonhaure), mgonzalez@uc.edu.py (M. González).

https://doi.org/10.1016/j.jss.2019.110489
0164-1212/

[1] In this article, we refer that an application feels responsive (*responsiveness*) when it delivers data and feedback to users in a timely manner. Researchers have found the most important factor in determining user satisfaction is the systemâs responsiveness (or the system's ability to keep up with users, keep them informed about its status, and not make them wait unexpectedly) (Johnson, 2014).

[2] *Architecture* focuses on how the principal elements and components within an application are used by, or interact with, other significant elements and components within the application. Furthermore, *layers* describe the logical groupings of the functionality and components in an application (Patterns, 2009).

[3] For the rest of this article, in our approach definition, we refer *data sources* as *data providers*.

MAAG's data layer guidelines to achieve offline mobile applications. In particular, taking into account different data storage options, as well as different data providers, and also providing helpers and utilities that facilitate the data persistence handling.

A complementary issue in regards to the data layer is that each mobile operating system (mobile OS) handles data storage differently. Data caching or store usually requires considering different implementations for each data persistence mechanisms in each mobile platform (more details in Section 2.1). This fragmented environment caused by the variety of mobile platforms (i.e., **fragmentation** phenomenon), is one of the main challenges for mobile developers (Marinho and Resende, 2015; Ribeiro and Silva, 2014). The amount of different mobile OS is not substantial, but the different versions of the same mobile OS impact the mobile application development process. Taking this into account, the native mobile application development effort is highly affected, increasing almost linearly in the face of such profound differences. Moreover, this phenomenon affects the portability of mobile applications, having to develop each of them practically from scratch, which increments the development time and the maintenance cost of the applications (Marinho and Resende, 2015). Therefore, this study focuses on such multi-platform native mobile application development issues.

In summary, the goal of the present study is to support the data persistence handling in the domain of mobile business applications in case of network connectivity problems. Moreover, taking into account the fragmentation issue and that the source of data could imply different data providers with their respective constraints.

For this purpose, an interesting opportunity is an adoption of the Model-Driven Development (MDD) approach, which may ease the smartphone application development for different platforms (Ribeiro and Silva, 2012). MDD permits stepping back and recompile the Platform Independent Model (PIM) to distinct technology platforms using different Platform-Specific Model (PSM) and compilers. This approach allows reusing diverse aspects of the mobile application design (interface, business logic, data access), ensuring portability and reducing the native mobile application development cost, effort and time for different platforms (Muñoz Riesle et al., 2015).

Among other proposals, the Model-Oriented Web Approach or MoWebA (González et al., 2016b) presented a methodological MDD approach for the development of web applications that adopt the Model-Driven Architecture (MDA) standard, and it expresses appropriate option upon the portability problem in the development of mobile applications (Sanchiz et al., 2017) (more details in Section 2.2).

Being MoWebA a general approach, one of its main characteristics is the adoption of an intermediate layer between the PIM and the PSM, called Architecture-Specific Model (ASM), to capture architectural requirements and their specifications maintaining the PIM more independent. It is worth to mention that a previous work (Sanchíz et al., 2018) presents an extension of MoWebA focused on network communication between mobile applications and their functions in the cloud. Complementary, this study aims to propose an extension of the MoWebA oriented explicitly to the development of native mobile applications with offline support in case of network connectivity problems. The approach is called **MoWebA Mobile**, and it covers data persistence handling and data providers' definition. The main contributions of this study are the definition of: (i) a Meta-Model for the mobile ASM; (ii) the rules of transformation from PIM to ASM; and, (iii) the rules for automatic code generation for two different platforms (i.e., Android and Windows Phone). Moreover, this study presents a first evaluation of the MoWebA Mobile proposal.

Subsequent sections of this paper are organized as follows. Section 2 clarifies necessary concepts to the scope of this work, like the data layer and the mains aspects of MoWebA. Section 3 discusses related work. Section 4 presents the actual proposal: MoWebA Mobile, including its main original contributions. Then, Section 5 focuses on the first evaluation experience. Section 6 analyzes and discusses the results of the first evaluation and outlines some future research directions. Finally, Section 7 concludes with a summary of this work.

2. Background

In the following, we first define the data layer, its elements, and the impact of the fragmentation phenomenon on data management. Then, we introduce the major aspects of MoWebA.

2.1. The data layer definition and the fragmentation phenomenon

As already mentioned, according to MAAG (Patterns, 2009), in mobile multi-layered architecture, it is possible to implement the data caching in the presentation layer or the data layer. On the one hand, the amount of cached data in the presentation layer is usually limited due to limited client resources. On the other hand, the data layer provides access to data hosted within system boundaries and to data exposed by other networked systems. This data is later used by other layers, as the business logic, and perhaps the presentation. For mobile applications in constant access to remote data, the data layer stands for the foundations in the construction of a usable and robust application.

In Fig. 1, we observe the multi-layered mobile application architecture of MAAG, where the data layer, aside from the data persistence handling, defines *Data Sources* for the mobile application. Thanks to the *Data Access Components*, the logic to access the underlying data sources is abstracted. Furthermore, it is possible to identify common data access logic in separate reusable helper (*Data Helpers*) or *Utility* data access components (*Utilities*). Finally, when is required access to data provided by an external service, *Service Agents* implement data access components that abstract the different requirements for calling services from the application. These agents may also provide additional services like caching, offline support, and mapping between the format of data types in a data exchange service - application (Patterns, 2009).

A complimentary issue in fragmentation regards the different types of data that could be saved on the phone using different data storage mechanisms. Typical examples of these data types are images from the device camera, user configuration preferences, files backup, documents, and others. These data might come from external sources (e.g., remote server), or the same phone (e.g., sensors) (Marinho and Resende, 2015). In this context, Mahmoud et al. (2012) present different data storage options as: (i) HTML5 (W3C (0000)), with its specific persistence methods for web applications; (ii) internal storage with files and integrated databases like SQLite (SQL); and, (iii) external storage with Cloud Storage Services (e.g., Dropbox (Pag) and Google Drive (MyD)), and external storage devices (e.g., SD memory card).

Interested in the data caching management in different mobile OSs, in the first place, we carried out an analysis of the comparative study of Grønli et al. (2014) and Mahmoud et al. (2012) about different mobile platforms[4] Then, we gathered information on each of the mobile OS official documentations (Android (And), iOS (App), and Windows Phone (UWP)). As a result, we categorized different data storage mechanisms in local storage (key-

[4] A mobile platform comprises the underlying hardware of the device, the architecture on which it is based, the operating system, the Software Development Kit (SDK) of the provider, and the standard libraries (Sommer and Krusche, 2013).

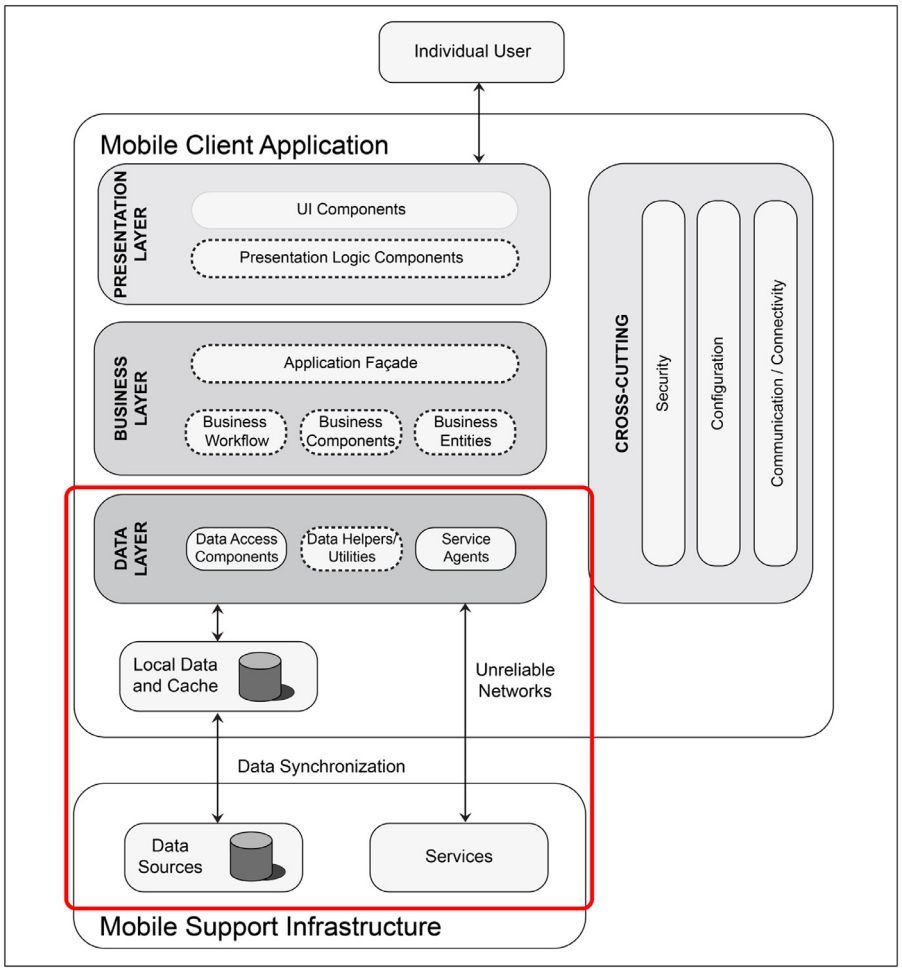


Fig. 1. Multi-layered mobile application architecture of MAAG. The scope of this work is framed in the data layer (indicated by the red rectangle)– *source: MAAG (Patterns, 2009)*. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

Table 1
Summary of data persistence mechanisms implementation in Android, iOS and Windows Phone platforms.

Mobile OS	Key-values	File management	Database	External storage support
Android	SharedPreferences, Bundles	Java Files Stream	Content Provider, SQLite	✓
iOS	NSUserDefaults, Property lists, NSCoding	NSFileManager	Core Data, SQLite	✓
Windows Phone (WP)	Isolated Storage Settings	Isolated Storage File System	Isolated Storage, SQLite	✓

value pairs, files and documents, integrated databases and external storage), and remote or cloud storage (hosting services offered by companies, or database hosted on remote servers). In particular, this study specialized in data persistence on the mobile side, considering local data storage (i.e., device-specific storage Mahmoud et al. (2012)) mechanisms[5] and data providers.

In Table 1, we present a summary of these data persistence mechanisms implementation in different mobile OSs, noting that each OS handles data storage differently, making difficult the development process in this fragmented environment.

All the mentioned above set the basis for our architectural approach for the data layer definition (presented in Section 4.1).

[5] For the rest of this article, we refer to these *local storage mechanisms* simply as *data persistence mechanisms*, knowing that it concerns the mobile local-side data management.

2.2. A glance to MoWebA

Fig. 2 presents the dimensions of MoWebA that cover its modeling and transformation processes. MoWebA adopts the MDA approach by identifying three different phases related to modeling and transformation activities (the *Phases* dimension): (i) The problem space, covered by CIM (Computational-Independent Model), and PIM; (ii) the solution space, covered by ASM, and PSM; and, (iii) the source code definition, covered by the ISM (Implementation-Specific Model) and manual code adjustments. The *Levels* dimension deals with complementary perspectives to be considered in every phase (content, business logic, navigation, presentation, users). Finally, the *Aspects* dimension addresses structural and behavioral considerations for each perspective (González et al., 2016b).

To develop applications, MoWebA defines two main complementary processes: the first related to modeling activities and the

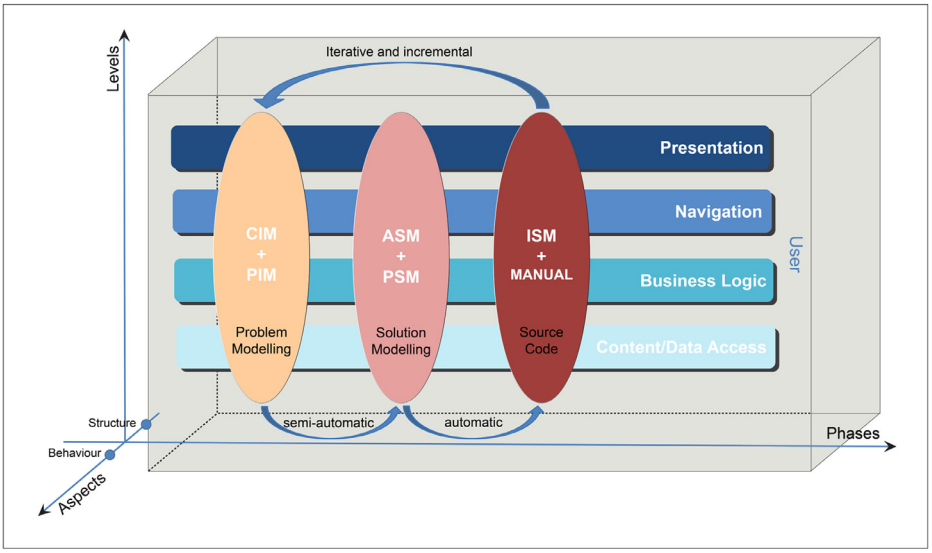


Fig. 2. Dimensions of MoWebA that cover its modeling and transformation processes.

second one related to transformation activities. To formalize the modeling and transformation processes, it adopts the MOF language (Met) for the definition of the abstract syntax, and the UML (Wel) profile extension for a precise definition of the modeling language (i.e., concrete syntax).

The ASM allows details related to the architecture to be moved out from the problem space (PIM) to an intermediate model in the solution space (ASM), avoiding the PIM from containing such details and contributing to its so desired portability. Moreover, the ASM model can result in different PSM models, one per each implementation platform. Therefore, the ASM model enriches previous models with additional information related to the system architecture (e.g., Rich Internet Application (RIA), REST, mobile, among others).

In this line of thought, MoWebA can be extended through the definition of ASM, used to complement the PIM with information about a specific architecture. In previous work, MoWebA has already been extended to support other architectures like RIA and SOA (Nuñez et al., 2018; González et al., 2016a), but these extensions have not been oriented to the mobile data persistence.

In order to use an ASM model, we need to define it first. This definition encompasses the specification of the corresponding meta-model, among other steps. Brambilla et al. (2012) have recommended a process for defining an abstract syntax, and MoWebA follows it to define an ASM meta-model. Furthermore, MoWebA complements the suggested process with additional steps that go from the definition of the concrete syntax until the generation of the final code of an application (González et al., 2016a). The steps of this process are synthesized below:

1. Define the ASM meta-model using MOF.
2. Define the corresponding UML Profile.
3. Specify the mapping rules from PIM elements to ASM elements.
4. Define transformation rules from PIM to ASM using standard transformation languages such as Atlas Transformation Language (ATL) or Query/View/Transformation (QVT).
5. Define transformation rules from ASM to PSM and PSM to code, or from ASM to code, using Model-to-Model (M2M) and Model-to-Text (M2T) – e.g., Acceleo (Acc) – transformation languages, respectively.

As it can be seen in the steps of the previous process, the MoWebA approach requires some additional effort as a counterpart of improving the portability of the PIM and facilitating the architectural evolution of applications. This additional effort includes the need for the specification of ASM meta-models and the definition of the corresponding transformation rules, to achieve automatic transformations on the proposed architecture. In any case, it should be noticed that steps 1, 2, and 3 of the process will only be executed once when targeting a new architecture for the first time.

Once the ASM for a specific architecture is defined, it can be used to develop an application for the selected architecture following these steps:

1. Define the MoWebA CIM and PIM diagrams following the modeling process.
2. Apply transformation rules to obtain the first version of the ASM model.
3. Make manual adjustments (if necessary) to complete the ASM model.
4. Generate the PSM models and the final code, applying transformation rules.
5. Include manual adjustments, if necessary.

3. Related work

Several works in the literature deal with the cross-platform development, comparing different frameworks through surveys with a variety of emphases, some focused on the industry's perspectives and opinions (Biørn-Hansen et al., 2019b), others focused on performance characteristics of mobile applications developed with different frameworks (Jia et al., 2018), others trying to select the best framework (Botella et al., 2016), others covering technical and overarching aspects (Biørn-Hansen et al., 2018). However, few of them contemplate MDD approaches in their analysis. Most of the time, the reason for not considering MDD is either its poor adoption by the industry (Biørn-Hansen et al., 2018, p. 65) or the fact that it serves mostly academic purposes (Biørn-Hansen et al., 2018, p. 65), or both. Among the few available, Biørn-Hansen et al. have published an interesting survey that covers MDD (Biørn-Hansen et al., 2019a). In their survey, they propose a taxonomy for cross-platform development approaches (namely, 1. the Hybrid approach, 2. the Interpreted approach, 3. the Cross-Compiled approach, 4. the Model-Driven approach, 5. the Progressive Web Apps approach (PWA)) and show the state of research for each, focusing on multiple core concepts like user experience, device features, performance, and security. Another relevant study proposes an interest-

ing evaluation framework for mobile cross-platform development approaches (Rieger and Majchrzak, 2019) that must be considered for any further study.

Although these works regarding cross-platform development show significant progress, native apps continue to prevail in app stores (Biørn-Hansen et al., 2019b; 2019a, p. 2), primarily because of users' rates and preferences (Botella et al., 2016). In this context, development approaches that generate native code (generative approaches like model-driven approach, cross-compiled approach, and transpiling approach) outweigh development approaches with a web approach (hybrid approach, interpreted approach, and PWA). Considering generative approaches to cross-platform development, MDD offers several advantages (Rieger and Kuchen, 2019). MDD based approaches lead cross-compiled approaches due to its direct access to native device features (Biørn-Hansen et al., 2019a, p. 8) and transpiling approaches because there is more to app development than just the technical equivalence of code (Rieger and Kuchen, 2019, p. 7433).

Despite the fact that our work deals with some extent with cross-platform development, our main goal is to analyze the data persistence modeling and management from MDD proposals in the context of mobile applications' development. For this purpose, we conducted a study of the available literature following, in a non-strict manner, some guidelines proposed for the Systematic Mapping Studies (Keele, 2007):

- The search was done in digital libraries as IEEE Xplore (IEE), ACM (ACM), SpringerLink (Hom), ScienceDirect (Hom), and ResearchGate (Hom). In the same way, we used Google Scholar (Goo).
- Our search chains were based on aspects of our interest: MDD, model-driven development, mobile, and persistence.
- The exclusion criteria applied to the reviewed works were: (i) They have nothing to do with MDD or mobile development; (ii) year of publication as from 2005; (iii) discontinued approaches; (iv) approaches without peer review.

Table 2
Comparison of MDD proposals for mobile application development. In gray color are highlighted our features of interest in this study. While in green color, those proposals with more similarities to ours. **Table symbols**: (-) it does not correspond, (✗) it does not apply, (✓) it does apply, and (*Not specified*) value represents that the information found in the paper was not enough to confirm or deny an attribute. For space reasons, we abbreviate Windows Phone as WP. (For interpretation of the references to colour in this table legend, the reader is referred to the web version of this article).

Proposal	Year	MDA	Output platforms	Native Code GEN	Modeling aspects	Data persistence modeling	Offline support	External server connection
Kramer et al. (MobDSL) [44]	2010	✗	Android, iOS	-	Behavior, GUI	✗	✗	✗
Min et al. [45]	2011	✗	-	-	Data, Behavior, GUI	✓	✗	✓
Ko et al. [46]	2012	✗	-	-	Data, Behavior, GUI	✓	✗	*Not specified*
Sabraoui et al. [47]	2013	✓	Android, Blackberry	✓	GUI	✗	✗	✗
Botturi et al. [48]	2013	✓	Android, WP	✓	Data, Behavior, GUI	✗	✗	✗
Le Goaer and Waltham (Xmob DSL) [49]	2013	✓	Android, iOS, .NET	✓	Data, GUI	✓	✗	✓
Heitkötter et al. (MD2) [3, 4, 50]	2013	✗	Android, iOS	✓	Data, Behavior, GUI	✓	✓	✓
Brambilla et al. (IFML Mobile) [51, 52]	2014	✓	Android, iOS	✗	Data, Behavior, GUI	✗	✗	*Not specified*
Usman et al. (MAG) [53]	2014	✗	Android, WP	✓	Behavior, GUI	✓	*Not specified*	*Not specified*
Jones and Jia (AXIOM) [54, 55]	2014	✓	Android, iOS	✓	Data, Behavior, GUI	✓	✗	*Not specified*
Ribeiro and da Silva (XIS-Mobile DSL) [7]	2014	✗	Android, iOS, WP	✓	Data, Behavior, GUI	✓	*Not specified*	✓
Geiger-Prat et al. (MIM-DSL) [56]	2015	✗	-	-	GUI	✗	✗	✗
Francese et al. [57]	2015	✗	Android, iOS	✗	GUI	✗	✗	✗
Barnett et al. (RAPPT DSL) [58, 59]	2015	✗	Android	✓	Data, Behavior, GUI	*Not specified*	*Not specified*	✓
Acerbis et al. (WebRatio Mobile) [60, 61]	2015	✗	Android, iOS, Web	✗	Data, Behavior, GUI	✓	✓	✓
Freitas and Maia (JustModeling) [62]	2016	✗	Android	✓	Data, Behavior, GUI	✓	✗	✗
Channonthawat and Limpiyakorn [63]	2016	✓	Android	✓	GUI	✗	✗	✗
Benouda et al. [64]	2017	✓	WP	✓	Data, Behavior, GUI	✓	✗	✗
Lachgar and Abdali [65]	2017	✓	Android, iOS, WP	✓	Data, Behavior, GUI	✓	✗	✗
Veisi and Stroulia [66]	2017	✗	Android	✓	Data, Behavior	✓	✗	*Not specified*
Vaupel et al. [67]	2018	✗	Android, iOS	✓	Data, Behavior, GUI	✗	✓	✓
Sanchez and Florez (MPML) [68]	2018	✓	-	-	Data, Behavior	✓	✗	✓
Rieger and Kuchen (MAML) [36, 69]	2018	✗	Android, iOS	✓	Data, Behavior, GUI	✓	✓	✓

As a summary of our study, Table 2 shows the collected proposals, ordered by year since 2010, from least to greatest. In total, we analyzed and compared 23 MDD solutions for mobile application development. In particular, determined to review concepts and implementations of the data layer by examining the use of persistence in these proposals, we outline our features of interest in: (i) Data persistence aspects considered in the modeling; (ii) support of connection with external servers; and, (iii) offline support. Other features also considered as the adoption of MDA, output platforms, the native code generation, and the modeling aspects considered in respective proposals are analyzed next.

The majority of the proposals present comparatives with at least two output platforms, but in some cases, the approaches were described in only one platform (Barnett et al., 2015b; 2015a; Freitas and Maia, 2016; Channonthawat and Limpiyakorn, 2016; Benouda et al., 2017; Veisi and Stroulia, 2017). Android and iOS were the first and second most selected option as output platforms for case studies and evaluations. Windows Phone appears in third place.

Most of the applications generated are native and data-oriented. Between those approaches that do not generate code, some consider only a modeling language (Geiger-Prat et al., 2015; Sanchez and Florez, 2018), or present a UML meta-model proposal (Min et al., 2011; Ko et al., 2012), or need additional libraries to interpret the modeling for different platforms (i.e., MobDSL (Kramer et al., 2010)). Other proposals generate hybrid code for PhoneGap framework (Brambilla et al., 2014; Bernaschina et al., 2018; Francese et al., 2015; Acerbis et al., 2015a; 2015b). The remaining works generate native code.

Just a few proposals contemplate only one aspect when modeling the mobile applications, in particular, modeling only the Graphical User Interface (GUI) of the applications (Sabraoui et al., 2013; Geiger-Prat et al., 2015; Francese et al., 2015; Channonthawat and Limpiyakorn, 2016). The rest of the proposals in the majority, contemplate GUI, Data, and even behavior modeling of the mobile application.

Most of the projects generate the application taking into account the structure of the project according to the corresponding Integrated Development Environment (IDE) of the chosen platforms. Through this, a compilation can be carried out with their respective SDK and thus obtain the final application. Nevertheless, a different approach was found in MD2 (Heitkötter et al., 2013; 2015; Majchrzak et al., 2015) and the MAML framework (Rieger and Kuchen, 2018; 2019). MD2 allows compiling and generating the final application without modifying the generated code, which includes the project's files and settings, even static libraries. MAML is closely related to MD2 because it uses MD2 for the generation of the final code (Rieger and Kuchen, 2019, p. 7433). Therefore, we assume that, like MD2, MAML also allows compiling and generating the final application without modifying the generated code.

Regarding our features of interest considered in Table 2, we start analyzing different data persistence mechanisms implemented by the different proposals. As we mentioned above, some approaches focus only on **modeling mobile GUI** (Sabraoui et al., 2013; Geiger-Prat et al., 2015; Francese et al., 2015; Channonthawat and Limpiyakorn, 2016), so they do not consider other aspects as data persistence. Thus, they do not support the offline mode. Other proposals **do not consider data persistence in modeling** (Kramer et al., 2010; Botturi et al., 2013; Brambilla et al., 2014; Bernaschina et al., 2018; Vaupel et al., 2018), and among these, there are not enough details to evaluate the offline support. It is important to remark, even though data persistence is not defined in modeling, Vaupel et al., 2018 describe a data model that is mapped thanks to its code generator. This model uses different technologies like a file-based system (e.g., storing data in XML/XMI format (Abo)) or a database system (e.g., using SQLite or MySQL), covering the mandatory data management feature different in Android and iOS platforms. Next, we summarize some interesting approaches detected for data persistence.

From the above, some modeling approaches included **specific platform API** for data storage (Min et al., 2011; Ko et al., 2012; Veisi and Stroulia, 2017). Min et al., 2011 describe in their UML profile for Windows Phone the stereotype *"IsolateStorage"*, which is an independent storage space supported by Windows Phone 7. Ko et al., 2012 also consider the *ContentProvider*, an Android-specific abstraction for data access. This abstraction allows access to the underlying application data, having as possible values: *SharedPreferences, InternalStorage*, among others. In the same way, Veisi and Stroulia, 2017 implement a data storage component using Android content providers. The proposed meta-model provides the user with the possibility to set an SQLite database, defining the database columns, tables, and information about the content provider. It is worth to highlight that the SQLite mobile database is widely chosen between those proposals that generate or contemplate database design and generation.

Other proposals include only **database stereotypes** in their meta-models (Benouda et al., 2017; Lachgar and Abdali, 2017). Benouda, Azizi, Moussaoui, Esbai, 2017 describe a meta-model for Windows Phone applications, where beside GUI elements, a database stereotype is defined, with which the user can model the database model for the application, and the generated code includes Database Helpers, Entity Models or DAOs (Data Access Objects), and CRUD functionalities (i.e., create, read, update and delete). Lachgar and Abdali (2017) propose a meta-model that allows using a database associated with an application, defining its name and version, but with some limitations, according to the authors. The database configuration files are partially generated, not being able to generate the database access layer completely.

Two further proposals, AXIOM (Jones and Jia, 2014; 2016) and JustModeling (Freitas and Maia, 2016), present data persistence **based on annotations**. Both approaches decorated models with these annotations, allowing the user to define columns, table names, and other database related info. These annotations bring the approach the possibility to choose and generate later the persistence framework.

Persistent stereotypes are presented on Ribeiro and Silva (2014) and Usman et al. (2014) proposals (XIS-Mobile and MAG, respectively). On the one hand, MAG assigns a stereotype *"Persistence"* for data storing/retrieving, together with an attribute *"Persistencetype"* to indicate the type of persistence. The authors mapped the *"Persistence"* stereotype to a specific API platform (e.g., *ContentProvider* in Android and *External Storage* in Windows Phone). *"Persistencetype"* seems to be fundamental since it can contemplate different types of persistence. On the other hand, XIS-Mobile establishes the definition of a persistent entity (*Xis-Entity*) using a *"Persistent"* parameter to state if it should be stored or not, but it does not provide more details about the mechanisms for persistent data.

Other proposals differentiate the concepts about **local or remote storage** (Goaer and Waltham, 2013; Sanchez and Florez, 2018; Rieger and Kuchen, 2018; 2019). First, in a modeling example, authors of the MAML framework (Rieger and Kuchen, 2018; 2019) describe a data source that specifies the data type of the manipulated objects and whether they are only saved locally on the device or managed by the remote back-end server. Then, Sanchez and Florez (2018) describe MPLM (Model Peripheral Modeling Language), and one of the concepts MPML includes is data storage. *DataStorage* models two elements, the *LocalStorage* of the device and the Rest Services, with which the user can model the CRUD (Create, Read, Update and Delete) operations. Finally, in a data model called Xmob-data, Goaer and Waltham (2013) detail if the connection refers to a local data source (*localsource*, specifying the

input and output data type) or to a remote data source (*remote-source*, specifying the URL connection).

Another feature of interest for analysis is the external server connection. This server connection involves the exchange of data between the device and a remote database, which could imply the need for storing data locally. As we can see in Table 2, on the one hand, few proposals consider an external connection. Some approaches included network stereotypes in its meta-model (Min et al., 2011), or URL definition of an external data source (i.e., Xmob DSL (Goaer and Waltham, 2013)), or API calls modeling (i.e., RAPPT DSL (Barnett et al., 2015b; 2015a)), and the use of REST services (i.e., MPML (Sanchez and Florez, 2018)). On the other hand, MD2 (Heitkötter et al., 2013; 2015; Majchrzak et al., 2015), MAML framework (Rieger and Kuchen, 2018; 2019), and WebRatio Mobile Framework (Acerbis et al., 2015a; 2015b) allow the user to create back-end services. Moreover, XIS-Mobile (Ribeiro and Silva, 2014) defines data providers, allowing the modeling of communication among the final application with external servers.

The last feature of interest is the offline mode support. This feature can be done by saving data on the phone (e.g., database and files) to use them in case there is no network connection available (data caching). A few proposals affirm their approaches present offline mode support (i.e., MD2 (Heitkötter et al., 2013; 2015; Majchrzak et al., 2015), MAML framework (Rieger and Kuchen, 2018; 2019), WebRatio Mobile (Acerbis et al., 2015a; 2015b), and Vaupel et al. (2018)).

Next, we highlight some common aspects adopted by most of the proposals in this study:

- Most of the applications generated are native and data-oriented.
- The majority of the proposals present comparatives with at least two output platforms.
- Most of the projects generate the application taking into account the structure of the project according to the corresponding IDE of the chosen platforms. Through this, a compilation can be carried out with their respective SDK and thus obtain the final application.
- Most of the proposals consider the modeling of data structure, GUI, and even behavior of the application.
- Some data persistence modeling approaches found were: (i) Specific platform API device-storage mechanisms like Min et al. (2011) and Ko et al. (2012); (ii) database stereotypes in their meta-models, with which the user can describe the database model for the application (Benouda et al., 2017; Lachgar and Abdali, 2017); (iii) representing data persistence based on annotations (basically, decorating models with annotations for database definition); (iv) persistent stereotypes, as a selector of what data should be persisted or not; and, (v) local or remote storage, as a selector of what should be saved locally on the device or managed by the remote back-end server.
- SQLite mobile database is widely chosen between those proposals that generate or contemplate database design and generation.

With the clear intention to address the data persistence layer, we developed MoWebA Mobile as an extension of MoWebA. We chose to extend MoWebA instead of another MDD approach thanks to its ASM and to those characteristics that make it a possible option facing the portability problem. Firstly, as already mentioned before, the ASM constitutes a level of abstraction between the PIM and the PSM, encapsulating details related to the architecture the developed software will have. This encapsulation makes possible to improve both the generalization of the modeling and its understanding (by separating concepts), aspects that contribute to the development of the code generator, and, therefore, maximize the possibilities of extending the approach to more target platforms. The goal of facilitating the extension of the approach to more target platforms is something that was also contemplated by other authors. In this regard, we highlight the proposal of Evers et al. (2016), later refined by Ernsting et al. (2016a), which consists of a reference architecture[6] that governs modeling and is agnostic regarding the MDD approach. Among other things, this reference architecture facilitates the understanding of the structure's archetype, that is, the understanding of meta-models and their instances (Ernsting et al., 2016a). Secondly, MoWebA can be an appropriate option upon the portability problem in the development of mobile applications (Sanchiz et al., 2017).

Our primary focus is on the data layer modeling. In doing so, it is relevant to consider the MAAG's data layer guidelines to achieve offline mobile applications (Patterns, 2009). In particular, taking into account different data storage options, as well as different data providers, and also providing helpers and utilities that facilitate the data persistence handling. On the one hand, for the data persistence modeling, and a difference of those which include specific platform API, we include architecture-specific persistent types (based on Table 1), like a database, files, and key-value pairs. These elements are later mapped to platform-specific API through transformation rules for code generation. Moreover, inspired by MAG (Usman et al., 2014) and XIS-Mobile (Ribeiro and Silva, 2014), we used persistent stereotypes to indicate which data should be stored. On the other hand, inspired by XIS-Mobile, and following the guidelines of MAAG (Patterns, 2009), we also consider data providers, allowing the modeling of incoming data from different sources.

Among those proposals which do consider data persistence modeling, MD2 (Heitkötter et al., 2013; 2015; Majchrzak et al., 2015), MAML framework (Rieger and Kuchen, 2018; 2019), and WebRatio Mobile (Acerbis et al., 2015a; 2015b) take into consideration offline support, and external server connection as well (i.e., REST as a uniform and portable communication interface adopted by WebRatio and MD2 (Sanchíz et al., 2018)). As we can see, these proposals cover all our features of interest. To summarize, we remark the main differences in our proposal with MD2, MAML, and WebRatio Mobile.

A first aspect to mention is the use of OMG's standards, specifically the adoption of MDA. In this context, MoWebA Mobile, by extending MoWebA (see Section 2.2), adopts the MDA approach. Moreover, MoWebA introduces the ASM as part of the solution modeling, which is an interesting intermediate level for assuring more independence for different mobile-architecture specific platforms (Sanchiz et al., 2017).

More specifically, MoWebA Mobile specializes in the data layer, while the mentioned proposals contemplate the development of a complete layered application (business, data, and GUI). Nevertheless, our data layer specialization supports more data layer concept coverage. Specifically, while MD2, MAML, and WebRatio Mobile, consider only local database storage, MoWebA Mobile allows to define other data persistence mechanisms as file management and temporal persistence through key-value pairs (or dictionaries). Furthermore, being the data providers (or data sources, see Fig. 1) an important concept considered in this layer, MoWebA Mobile adopts three types of providers (see Section 4.1). This adoption offers the capacity to consider mobile device sensors as well as other applications, as additional data sources for an application. On the other hand, and in our knowledge, MD2, MAML, and WebRatio Mobile contemplate only external providers, i.e., external connection with remote servers or back-ends.

[6] Disambiguation: in the ASM, the term *architecture* refers to the software architecture to be developed, while in the reference architecture, the term *architecture* refers to the architecture of the MDD approach itself, that is, the architecture that structures the models generated with the approach.

More systematic and experimental comparison analysis between the mentioned proposals and MoWebA Mobile remain for future studies.

Considering the differences, the common characteristics, and the similarities with other approaches mentioned above, we point out some additional aspects for which our proposal presents alternatives for solutions:

- Limited coverage of the concepts surrounding the modeling of the data layer in the development of mobile applications. Considering this limitation, we aim to conceptually cover the data persistence in the modeling of mobile application development through specific elements to consider different data persistence mechanisms, even in offline mode, and considering the connection with different data providers.
- More adoption of the OMG's (OMG) standards. None of those proposals with more similarities to our approach adopt MDA. In this sense, our approach follows the MDA guidelines by extending MoWebA (González et al., 2016b).
- Meanwhile, all the similar proposals consider all modeling aspects, MoWebA Mobile specializes in the data layer.
- Android and iOS as the most selected output platforms for case studies and evaluations. In this regard, we considered interesting to have more studies with Windows Phone, and not only with Android and iOS.

The next section presents in detail MoWebA Mobile: the extensions made to MoWebA, the development process, and the ASM definition.

4. An extension of MoWebA: MoWebA mobile

In this section, we present an extension of MoWebA for the development of native mobile applications focused on the data layer, called MoWebA Mobile. As already introduced, the proposal includes: (i) A Meta-Model for the mobile ASM; (ii) the rules of transformation from PIM to ASM; and, (iii) the rules for automatic code generation for two different platforms (i.e., Android and Windows Phone).

Indeed, we start this section with a general overview of the ASM proposal. Next, we present the MoWebA Mobile ASM definition process for the development of mobile applications considering the data persistence and data provider design of mobile applications. Finally, through an example, we present the transformation rules from PIM to ASM and from ASM to code to build the final mobile application code from the defined models. The final application is instantiated for both platforms, Android and Windows Phone.

The tools defined for the MoWebA Mobile approach, which are the meta-models, UML profiles, PIM-ASM, and ASM-code transformation rules, are available on the *MDD+ project* website (Nuñez, 2017).

4.1. An overview: MoWebA ASM for mobile data persistence

Our main aim in this study is to present an approach for the development of mobile applications that require constant access to a network. The domain of business applications includes those applications, defining them as form-based and data-driven applications interacting with back-end systems (Majchrzak et al., 2015; Heitkötter et al., 2015).

After the analysis of proposals in Section 3, we learned some common aspects adopted by these proposals that we consider in our approach, as well as some highlighted concerns to which we try to present alternative solutions with our approach. On this basis, we have concluded that we chose to extend MoWebA thanks to its ASM.

Some of the main features of MoWebA are (González et al., 2016b): (i) Well-defined layered structure (achieves a clear separation of concepts); (ii) modeling centered in a hierarchical function-oriented navigation (allows a more appropriate design of a user interaction based navigation); and, (iii) enrichment of existing models considering aspects related to the final architecture of the system (e.g., RIA, SOA, REST, among others). Considering the well-defined layered structure and our main aim, we focus on MoWebA's Data Access level. Here, the Entity Diagram allows defining the structure and static relationships between the classes in the problem domain at the PIM level (González et al., 2016b). Thus, this diagram was our clear choice to extend in order to cover more conceptual elements to the structural type, which might facilitate the modeling of mobile applications data persistence. Moreover, for data provider modeling, we extended a UML class. In Section 4.2.1, we will present more details about the definition of these models.

The well-defined layered structure of MoWebA motivated us to follow this software architecture pattern. Thanks to the benefits of a well-defined layered-structure, it is possible to evolve our design over time, allowing us to focus on one area of concern[7] at the time (Patterns, 2009). Engaging in the mobile application architecture definition, we found MAAG (Patterns, 2009). MAAG presents a series of guidelines about software architecture pattern design. By following the processes described in this guide for mobile application architecture design, and using the information it contains, we were able to define our area of concern: the data layer.

Based on the MAAG's guidelines for designing the data layer (Patterns, 2009), in Fig. 3, we represent which elements from these guidelines were adopted to establish the architecture for our approach.

As we mentioned in Section 2.1, the data layer provides access to data hosted within system boundaries and to data exposed by other networked systems. Furthermore, this layer, aside from the data persistence handling, defines data providers for the mobile application. Regarding data persistence, and based on Table 1, the data persistence mechanisms we considered are databases, files, and key-value pairs. These mechanisms are implemented in different ways on each platform. Currently, there are different mobile databases, but for reasons of this work, we opt for SQLite as a database due to is likely the most widely deployed and used database engine, and used extensively in major mobile platforms (Mos). About data providers, and according to XIS-Mobile (Ribeiro and Silva, 2014) profile specifications, we could identify that mobile applications can receive data from three types of sources. (i) External data providers, such as servers and remote databases. In this context, we chose REST, as a uniform and portable communication interface, adopted in extension in mobile application network communication approaches (Sanchíz et al., 2018). (ii) Internal data providers, such as sensors like the gyroscope and accelerometer, and device-specific hardware resources, such as camera and microphone. (iii) Through interoperability with other applications, to exchange different data types like File, Image, and more. It is worth to mention that the list of elements defined in data providers were results of finding common elements supported by each platform. The above applies particularly to internal providers, and interoperability with other applications. For this, we consulted each main platform documentation (Android, iOS, Windows Phone).

In our work, our output platforms are Android and Windows Phone. On the one hand, we decided to develop for Android because it is the most adopted mobile operating system, according

[7] Software Architecture is focused on organizing components to support specific functionality, and the organization of this functionality is often referred to as grouping components into areas of concern (Patterns, 2009).

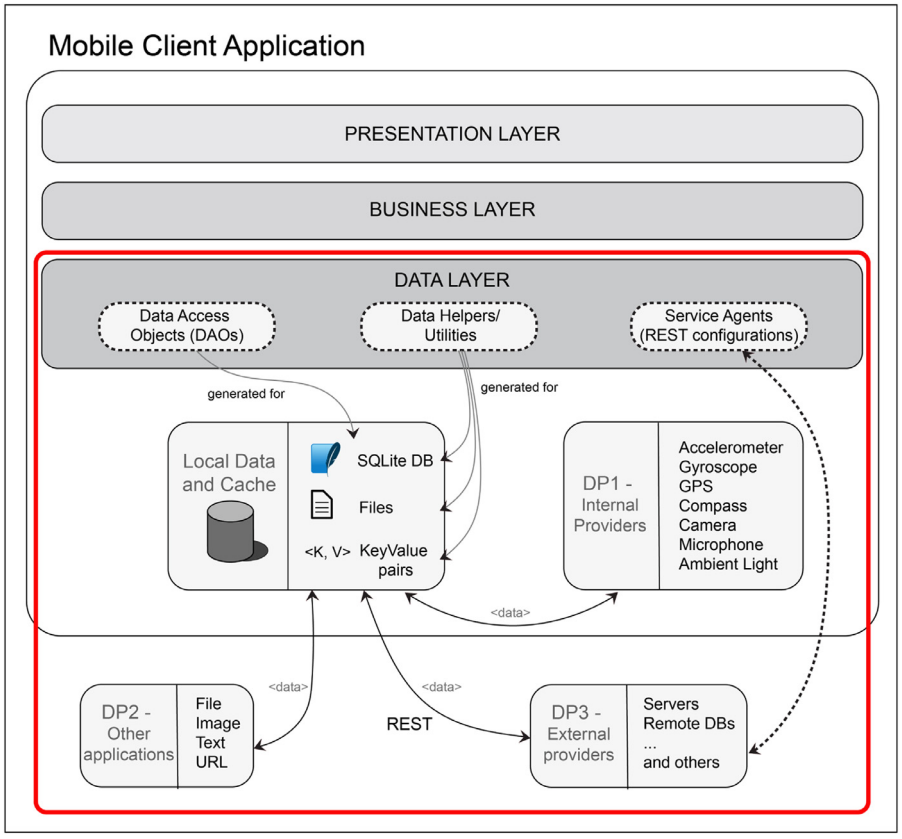


Fig. 3. MoWebA Mobile architecture scheme based on MAAG's guidelines (Patterns, 2009). The scope of this work is framed in the data layer (indicated by the red rectangle).. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

to studies carried out by the International Data Corporation (IDC) (IDC). On the other hand, in Section 3, we found it interesting to have more studies with Windows Phone, since most of the available approaches have been tested with Android and iOS.

This generated application contains most of the tools that would help developers as a starting point when working with data-oriented applications, avoiding as far as possible, creating an application from scratch. First of all, all the data persistence designed and modeled are already implemented in the code (e.g., database configuration, tables creation, files i/o functionalities, key-value management). Second, sensors and device-specific hardware (e.g., camera) are already implemented, as well as a REST API skeleton is provided. So far, all these implemented functionalities can be later modified or extended by the developers in the case is needed. Furthermore, utilities classes and helpers are also generated, with standard functionalities already defined. Third, to test and get first sight of the generated functionalities, a set of basic and general screen are generated. This user interface shows the defined elements like data persistence functions by creating CRUD forms, uses the defined sensors and hardware device to demonstrate read values, and receives values from other applications.

It is important to note that, considering several studies analyzing the characteristics and requirements of the mobile business applications domain (Majchrzak et al., 2015; Heitkötter et al., 2015; Ernsting et al., 2016b), MoWebA Mobile is suitable for different scenarios. In general, MoWebA Mobile generates a basic form-based data-driven application, can communicate with a back-end server (where the business logic resides, almost exclusively (Majchrzak et al., 2015)), provide database access, and support offline work. Incoming data from a server or other data provider can be modeled and stored using specific data persistence mechanisms. This data can be modeled as well as output data (via web-services) using the data provider model. Nevertheless, certain aspects need improvement, or they are not contemplated yet.

On the one hand, MoWebA Mobile can access business logic located in a server (via web-services), and provide database access. The offline support is also possible, but limited. This is because an automatic data synchronization between the application and the server, as soon as the Internet connection is available, is not available yet. The device-specific hardware access (e.g., GPS), and sensors (e.g., accelerometer) are available. It is possible to define data types and CRUD operations for corresponding data, locally the device (data persistence modeling), as well as on a server (data provider modeling). In this context, it is necessary to improve the implementation of data bindings and input validation. On the other hand, a fundamental and general form-based user interface is generated. As we are not dealing with the presentation layer, it is not possible to model considering layouts and UI (User Interface) components (i.e., widgets). Not dealing with the presentation layer also limited us over the definition of the user navigation and sequence of UI views, as well as react to an event and state changes. Nevertheless, incoming data from other applications is considered (data interoperability with other applications).

From the mentioned above, it has to be kept in mind that these requirements are generic, being possible that certain business apps could have more rigorous or more open requirements (Majchrzak et al., 2015, p. 5). In Section 4.2.4, we present a modeling example of typical data-driven business applications to improve the understanding of the development scope of the approach.

In Fig. 4, we observe the development process with MoWebA Mobile. We start modeling with MagicDraw (Mag), which is a general-purpose modeling tool, using the UML profiles of MoWebA. Then, M2M transformation is made from PIM to ASM using ATL language with ATL EMFTVM (ATL EMF Transformation Virtual Ma-

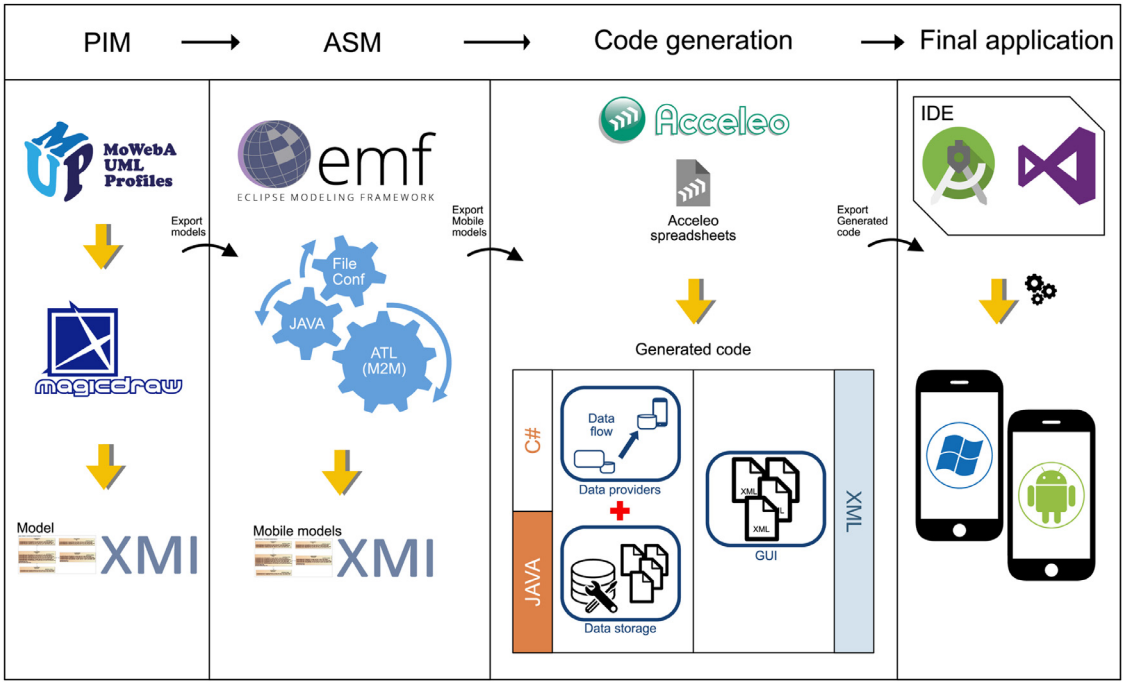


Fig. 4. Process for the design and the generation of mobile applications with MoWebA Mobile. The PIM to ASM transformation process is explained in Section 4.2.2, the ASM to Code transformation process in Section 4.2.3, and the final application generation is shown through a modeling example in Section 4.2.4.

chine). The result of this step is the ASM Model, which has to be adjusted manually. Then, to generate code from the performed models, we export these in XMI v2.1 format (XML of Metadata Exchange) and import them in Acceleo. Using defined transformation rules (Acceleo spreadsheets), we generate the source code of the mobile application data layer implementation. We define transformation rules to generate this implementation, both for Android (Java) and Windows Phone (C#). Besides the generated code from the M2T transformation rules, we have developed a set of user interfaces to show the data layer functionalities. Finally, to obtain a native application for both platforms, the generated code must be compiled in their respective IDEs: Android Studio (Dow) and Visual Studio (Sro).

In the following sub-sections, we explain how the ASM process definition is.

4.2. Moweba mobile: ASM process definition

As previously mentioned, in this work, we use MoWebA (González et al., 2016b) as a starting point and propose an extension for Mobile's functionalities at the architectural modeling level. The definition of the mobile ASM started with the extension of MoWebA's Entity PIM to take advantage of conceptual elements of the structural type. Then, the meta-model and mobile UML profile were defined from the extensions made. Finally, as we will see in Section 4.2.2 and 4.2.3, the M2M and M2T transformation rules were determined to obtain the elements described in the model automatically.

4.2.1. Mobile meta-model and profile

MoWebA presents its Entity Diagram to define the structure and static relationships between the classes identified in the problem domain at the PIM level (González et al., 2016b). We extended this diagram to cover more conceptual elements of the structural type, which will later facilitate the modeling of mobile applications. We summarize two types of modifications: an extension of the entity properties (*EntityProperty*), and an addition of specific data types these properties can have (*dataType*). The aggregated properties allow: Specify a data type (*DataType*), whose values can be *bool, date, time, datetime, int, real, string* and *text*; select a maximum data size (*size*); restrict the field to being null (*notNull*); indicate a default value (*defaultValue*); and, establish if the property is constituted as the identifier of the entity (*id*).

According to the extensions made, we present the specific mobile architecture model or mobile ASM. From this model, we can define mobile applications with the established functionalities: the local data persistence and the design of data provider components. The details of the meta-model and UML profile of the proposal are shown next. Starting with Fig. 5, we observe the meta-model of the proposal. From the figure, we can highlight the following: Two sections are distinguished in particular, the persistence of data (within the box with dashed lines: *Persistence Data*) and the data providers (within the box with dotted lines: *Data Provider*). In data providers, we find three distinguishable subsections: (i) Other applications as data providers (within the box labeled as *MobileAppData*); (ii) external data providers (inside the box labeled as *External Data*); and, (iii) internal data providers (within the box labeled as *Internal Data*). Concerning the UML profile, the mobile profile model (not shown in this document) presents the same functionalities definition as the meta-model, only with the definition of the elements (stereotypes, tag values, and enumerations), which enable the modeling. The profile was defined as an extension of the UML Class Diagram (the UML profiles are available on the MDD+ project website (Nuñez, 2017)).

Regarding data persistence, we introduce elements to identify what data will be stored on the device (*persistentEntity* tag value), and what type of persistence will be used (*persistentType* tag value). The first *persistentType* (*Database* type) allows generating a database and uses the name of the persistent entity as a table. *File*, another *persistentType*, enable us to save the persistent entity in files through functions that facilitate to handle files (e.g., reading, writing, and others). The last *persistentType* (*KeyValue* type), allows storing the persistent entity in key-value pairs, through functions that we provide for the management of this data. In turn, each

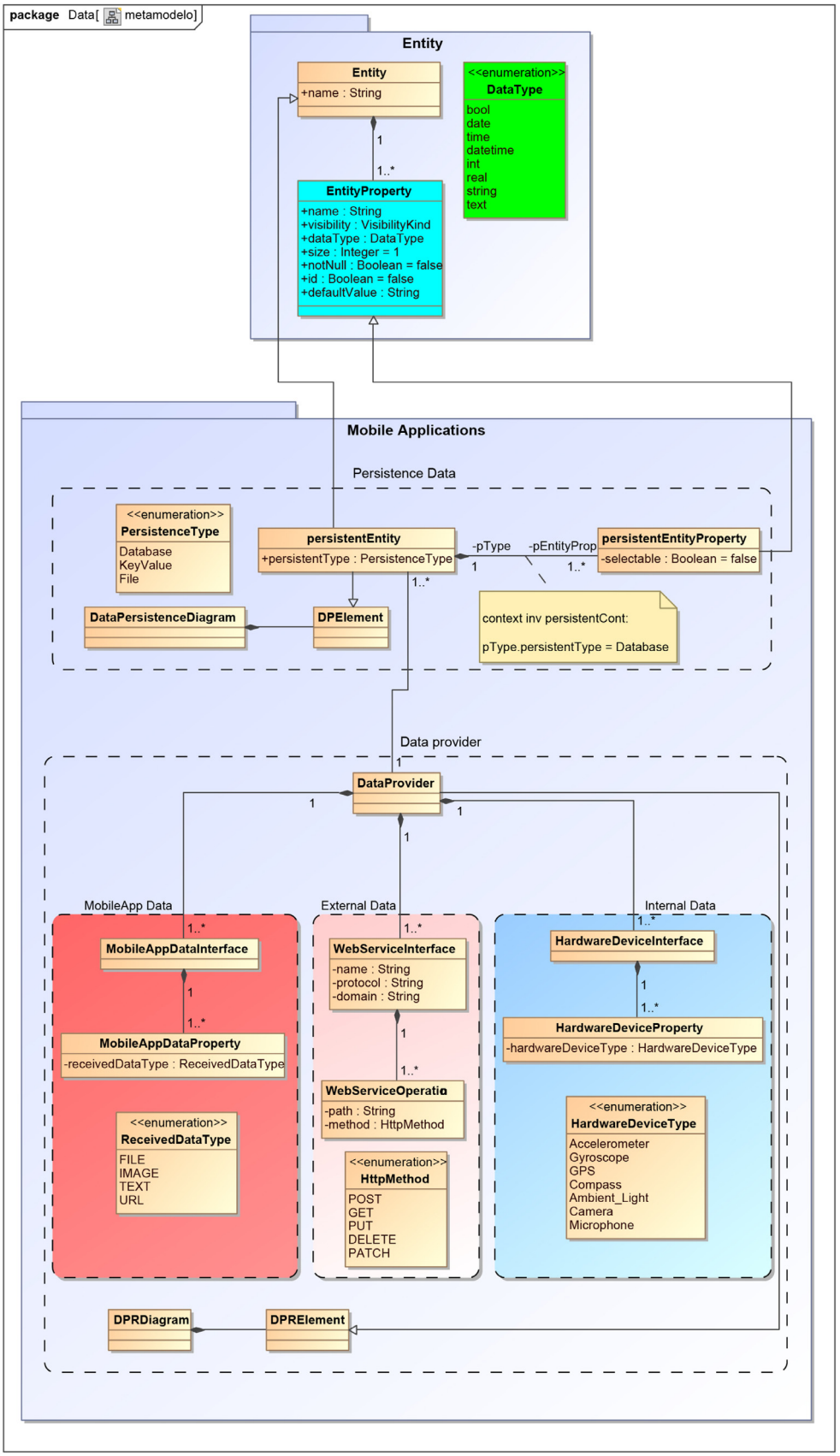


Fig. 5. Meta-model used for the definition of an ASM for mobile applications.

attribute of a persistent entity has properties with a stereotype ≪*PersistentEntityProperty*≫. At the same time, each property can be enriched with sub-properties, defined as tagged values. Apart from those mentioned, we define the property *selectable*, used for the persistence with databases (*persistentType* = *Database*). This property allows indicating if an attribute of the persistent entity will be used as a key for data selection, to execute *delete* and *update* operations in the database. Finally, we include the stereotype ≪*DataPersistence*≫ to identify the package where the persistent data model will be included. According to the transformation rules, this property allows identifying we are working with the persistent data model.

Concerning data providers, we introduce the following interfaces: *WebServiceInterface, HardwareDeviceInterface*, and *MobileAppDataInterface*, for the representation of external providers (via web-services), internal providers (through commonly supported sensors among platforms) and interoperability with other applications (data types defined for communication between applications), respectively.

External providers provide data by communicating the mobile application with servers or remote databases. This communication is made via web-services following the *REST* architecture. The class *WebServiceInterface*, through tagged values, allows defining the URL base connection: [*protocol*]: [*domain*]. With *WebServiceOperation*, we define the functions or services in the class *WebServiceInterface*. The name of the function corresponds to the name of the service, to which, through tagged values, we add the *HTTP* method (*method*) and the service access path (*path*). The *HTTP* method possible selected values are *POST, GET, PUT, DELETE*, and *PATCH*.

Smartphones have sensors and specific hardware resources (e.g., camera, microphone, among others) that provide data flow; we consider these as internal data providers. The *HardwareDeviceInterface* class allows defining which sensors and hardware resources to use through *HardwareDeviceProperty* and its tagged value *hardwareDeviceType*. As possible options, we have an *Accelerometer, Gyroscope, GPS, Compass, AmbientLight, Camera*, and *Microphone*.

The mobile application can interoperate with other applications installed on the same mobile device, sending or receiving data. Through the class *MobileAppDataInterface*, we can represent data that a mobile application could receive from other applications. With the tag value *ReceivedDataType* of the property (*MobileAppDataProperty*), we select what data type the application may receive and handle. *File* (for file exchange), *Image* (to receive images), *Text* (to receive texts), and *Url* (to receive links) are available as options. Finally, we include the stereotype ≪*DataProvider*≫ to identify the package where the data provider model will be included. According to transformation rules, this property allows identifying we are working with the model of data providers.

4.2.2. PIM to ASM transformation process

We chose the ATL language over QVT for the definition of the transformation rules. This choice is based, above all, on the fact that ATL is considered one of the most widely used transformation languages, both in academia and industry. Besides, a mature tool supporting is also available (Brambilla et al., 2012).

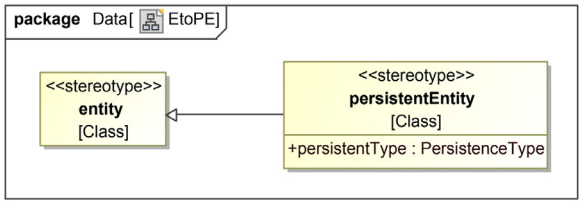


Fig. 6. A class Entity in the PIM model, becomes a *PersistentEntity* class in the ASM model.

A critical choice was selecting the engine execution mode of the ATL transformation, which has two modes of execution: the default, and the refining execution modes. In our case, we opted for the refining execution mode, since both, the meta-models of the source and target models are equals because MoWebA's implementation is based on profiles (González et al., 2016b). This choice has dramatically reduced the number of defining rules, although it has also led to some complications. The application of profiles in ATL is performed in the imperative block and turns out this option when using refining mode.

For the reason just mentioned above, we were forced to change the default compiler. The default compiler is EMF-specific Virtual Machine, but ATL provides other compilers. We finally opted for the ATL EMFTVM. Although the main reason for this choice was that it allows the use of the imperative block in the refining mode, necessary to work with profiles, there are some other advantages. For example, its performance is roughly 80% better than the default ATL EMF-specific VM (Jouault and Wagelaar, 2017b) and allows us to invoke native Java methods (Jouault and Wagelaar, 2017a).

Before the definition of the transformation rules, a mapping among PIM and ASM elements has been made. Firstly, seeking to identify which elements of the PIM model must be transformed and, above all, to which elements of the ASM model.

This mapping was performed through a visual analysis of the defined profile for each architecture (i.e., the profile for mobile data persistence (Nuñez, 2017)).

During the mapping phase, we noticed that, in general, when the relation between classes from source and target models is an inheritance, the transformation that allows obtaining the corresponding target element is straightforward and consists of the application of the correct stereotype. For example, we could detect that a class Entity in the PIM model, becomes a *PersistentEntity* class in the ASM model (see Fig. 6) and that property *entityProperty* in the PIM model is transformed into a *persistentEntityProperty* in the ASM model (see Fig. 7).

When the relation between two classes in one profile is not an inheritance, the mapping becomes a bit more difficult.

Next, we show some representative examples of M2M transformation rules defined for this project.

Headers The header section is an essential section of transformation rules as it details the compiler, the meta-models, the mode of execution, and the profiles used (see Listing 1).

```
1 -- @atlcompiler emftvm
2 -- @nsURI UML2=http://www.eclipse.org/uml2/2.0.0/UML
3
4 module ReglasM2M;
5
6 create OUT : UML2 refining IN : UML2,
7                            MOBILE_PROFILE : UML2,
8                            CONTENT_PROFILE : UML2;
```

Listing 1. Header section of the M2M transformation rules.

```
rule applyMobileStereoTypes(required : Boolean, s : UML2!Element, t : UML2!Element)
{
    using {
        new_stereotype : UML2!Stereotype = '';
        change_stereotype : Boolean = false;
        container : String = s.namespace.name.toString();
        element : String = s.name.toString();
        package : String = s.getNearestPackage().name;
    }
    do
    {
        for (stereotype in s.getAppliedStereotypes())
        {
            if (stereotype.getName() = 'entity')
            {
                new_stereotype <- thisModule.getStereotype('persistentEntity');
            }
            else if (stereotype.getName() = 'entityProperty')
            {
                new_stereotype <- thisModule.getStereotype('persistentEntityProperty');
            }
            --- It is decided whether or not to change the stereotype
            if (container = package) {
                --- if container is equal to package, then the element is a class
                change_stereotype <- let object : "#native"!"atl::conf::ConfM2M" =
                    "#native"!"atl::conf::ConfM2M".newInstance()
                        in object.aplicarEstereotipo(container,element);
            } else {
                --- if container is NOT equal to package, then the element is a property
                change_stereotype <- let object : "#native"!"atl::conf::ConfM2M" =
                    "#native"!"atl::conf::ConfM2M".newInstance()
                        in object.aplicarEstereotipo(package,container,element);
            }

            if ( new_stereotype <> '' and change_stereotype)
            {
                --if the UML!Element does not get its stereotype automically applied (i.e., UML not
                    required)
                --then we must apply it manually
                --note:surprisingly the "required" property doesn't seem to be exposed by UML2.
                --maybe look into this more
                if (not required)
                {
                    t.unapplyStereotype(stereotype);
                    t.applyStereotype(new_stereotype);
                }
                for (property in stereotype.getAllAttributes())
                {
                    --apply the value if there is one.
                    --don't apply the base type property as ATL cannot handle this.
                    --also don't touch read-only properties.
                    if(s.hasValue(stereotype,property.name)
                        and not property.name.startsWith('base_')
                        and not property.isReadOnly())
                    {
                        t.setValue(stereotype,property.name,s.getValue(stereotype,property.name));
                    }
                }
```

Listing 2. *applyMobileStereoTypes* called rule that change the stereotypes in the target model.

Called Rules The *called rules* is another vital section of transformation rules. The called rule named *applyMobileStereoTypes* allow changing the stereotypes in the target model (see Listing 2 below).

Configuration Files The configuration files allow the designer to control some transformation process aspects, allowing the achievement of two important capabilities. On the one hand, the possibility of capturing specific design decisions of the system under design that would otherwise not be automated, and on the other hand, the possibility that the designer has some level of influence over the transformation rules.

```
58              --marks the 'selectable' property as 'true' for the first property of the class
59              if (new_stereotype.getName() = 'persistentEntityProperty')
60              {
61                  if (not thisModule.Selectable_Control.includes(t.class.name)) {
62                      t.setValue(new_stereotype,'selectable',true);
63                      thisModule.Selectable_Control <-
64                          thisModule.Selectable_Control.including(t.class.name);
65                  }
66              }
67              for (property in new_stereotype.getAllAttributes())
68              {
69                  if (property.name.toString() = 'dataType')
70                  {
71                      if (s.refGetValue('type').name = 'Integer') {
72                          t.setValue(new_stereotype,'dataType', thisModule.getDataType('int'));
73                      }
74                      else if (s.refGetValue('type').name = 'Real') {
75                          t.setValue(new_stereotype,'dataType', thisModule.getDataType('float'));
76                      }
77                      else {
78                          t.setValue(new_stereotype,'dataType', thisModule.getDataType('text'));
79                      }
80                      t.refSetValue('type', OclUndefined);
81                  }
82              }
83          }
84      }
85  }
86 }
```

Listing 2. Continued

```
1 mod: 1
2 classes:
3     - package : package_containing_the_class
4       class : class_name
5 properties:
6     - package : package_containing_the_property
7       class : class_containing_the_property
8       property : property_name
```

Listing 3. Configuration files that allow the designer to control some transformation process aspects.

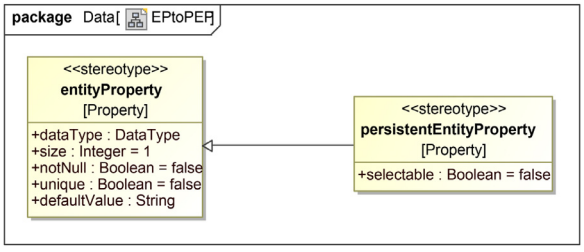


Fig. 7. A property *entityProperty* in the PIM model is transformed into a *persistentEntityProperty* in the ASM model.

The "ArchConfTransformacion.yaml" configuration file has two modes of operation (see Listing 3). Mode 1 specifies which elements of the PIM must be transformed, and mode 2 specifies which elements must not. If the file does not exist, all the PIM's elements referenced by the M2M transformation rules are transformed. Its structure is straightforward. It has three sections, and the first indicates the operation mode, the second, the affected classes, and the third, the affected properties. A class is referenced indicating the package containing it and its name, while a property, indicating the package and class containing it and the property name.

4.2.3. ASM to Code transformation process

From the modeling with the proposed mobile ASM, we automatically generate a mobile application to demonstrate the generated functionalities from each of the mentioned aspects. For this purpose, we have defined the M2T transformation rules. These rules perform a mapping between the model elements defined and the target code to be generated.

To build transformation rules, we followed an approach based on templates, and we used the Model Transformation Language (MTL) of Acceleo for defining the templates. To extend the functions of the MTL, we built a service in Java. In the same way, we used OCL (Abo) for doing queries to the model. Furthermore, we built these transformations rules based on the classes, properties, and operations with ASMâs profile defined stereotypes, tag values, and enumerations.

Listing 4 presents the main template rule of the Acceleo's transformations rules for the application generation. The generated target codes for Android and Windows Phone are Java and C#, respectively. Additionally, GUI code is also generated for Android

```
[comment encoding = UTF-8 /]
[module generate('http://www.eclipse.org/uml2/5.0.0/UML')]
[comment]Imports ...[/comment]

[template public generateElement(model: Model)]

[comment @main/]
[let aPackages: Sequence(Package) = model.eAllContents(Package) ]

[generateGeneralAndroidClasses(model)/]
[generateGeneralWindowsClasses(model)/]

[for (aPackage : Package | aPackages)]
[let aClasses: Set(Class) = aPackage.ownedElement->filter(Class) ]
[let p : Package = aPackage.ancestors(Package)->first()]
[comment]We go through the existing packages in the model, specifically: DataPersistence and
     DataProvider[/comment]

    [comment]If the DataPersistence package exists[/comment]
    [if (aPackage.hasStereotype('DataPersistence'))]

        [comment]Generate the beans or models of the application[/comment]
        [beansGenAndroid(aPackage, p.name.toLower())/]
        [beansGenWindows(aPackage)/]

        [comment]We only create this file if there is at least one Database type entity[/comment]
        [if (aPackage.isPackageHasThisPropertyStereotype('persistentEntity', 'persistentType',
             'Database'))]
            [generateDBForAndroid(aPackage, p.name.toLower())/]
            [generateDBForWindows(aPackage, p.name.toUpperFirst())/]
        [/if]

        [comment]We only create this file if there is at least one File entity type[/comment]
        [if (aPackage.isPackageHasThisPropertyStereotype('persistentEntity', 'persistentType', 'File'))]
            [generateFilesForAndroid(aPackage, p.name.toLower())/]
            [generateFilesForWindows(aPackage, p.name.toUpperFirst())/]
        [/if]

        [comment]We only create this file if there is at least one KeyValue entity type[/comment]
        [if (aPackage.isPackageHasThisPropertyStereotype('persistentEntity', 'persistentType',
             'KeyValue'))]
            [generateKVForAndroid(aPackage, p.name.toLower())/]
            [generateKVForWindows(aPackage, p.name.toUpperFirst())/]
        [/if]
    [/if]

    [comment]If the DataProvider package exists[/comment]
    [if (aPackage.hasStereotype('DataProvider'))]
        [for (aClass : Class | aClasses)]
            [comment]If the class has the WebServiceInterface package[/comment]
            [if (aClass.hasStereotype('WebServiceInterface'))]
                [generateRestAndroid(aClass, p.name.toLower())/]
                [generateRestWindows(aClass, p.name.toUpperFirst())/]
            [/if]

            [comment]If the class has the HardwareDeviceInterface package[/comment]
            [if (aClass.hasStereotype('HardwareDeviceInterface'))]
                [generateSensorsAndroid(aClass, p.name.toLower())/]
                [generateSensorsWindows(aClass, p.name.toUpperFirst())/]
            [/if]
        [/for]
    [/if]
[/let]
[/let]
[/for]
[/let]
[/template]
```

Listing 4. Main template of the transformation rules in Acceleo.

(XML (Ext)) and Windows Phone (XAML (XAM)). Finally, this generated code is ready to be executed for both mobile platforms, Android and Windows Phone, previous compilation in their respective IDEs.

As a summary of the main aspects of the generated code we have:

- Respect to data persistence, and based on Table 1, we used specific data persistence mechanisms for these platforms. Firstly, the generated database is SQLite for both platforms. Secondly, files are saved in the device, and specific functions are generated for each platform that allows executing *read* and *write* operations. Finally, key-value pairs enable handling *SharedPreferences* and *LocalStorage*, for Android and Windows Phone, respectively.
- Considering the data providers, we generate code from each of these identified providers. Firstly, the communication with external providers (e.g., servers or remote databases) is done via web-services following *REST* architecture. Secondly, we identified a series of common options for sensors and resources of hardware in smartphones: *Accelerometer, Gyroscope, GPS, Compass, AmbientLight, Camera*, and *Microphone*. Finally, we identify different data types commonly used in the exchange of data among applications (*ReceivedDataType*): *File*, for file exchange in general; *Image*, to receive images; *Text*, to receive text; and *URL*, to receive links (*links*).
- To get the first sight and test the generated functionalities, we generate a set of basic and general screens. These screens are forms defined from each persistent entity to load data and perform CRUD operations on them. Moreover, we generated a screen to verify the values thrown by each sensor and test the device hardware resources (e.g., GPS and camera).
- In addition to this application, we provide auxiliary classes (or *helpers*), equipping the user with functionalities to handle the different aspects mentioned.

4.2.4. Modeling example and code generation

To improve the understanding of the design process, we present an example of modeling an application. We use MoWebA and the extensions and aggregate definitions to design the functionalities of the data layer.

Following the development process described in Fig. 4, we use MagicDraw to perform the modeling example and code generation of a mobile application called *E-market*, starting from PIM, passing through the mobile ASM model, and generating the final code, focused on the aspects mentioned above.

E-market consists of a virtual store for product delivery. A catalog of products and the associated information of each product is available. A product can be selected and added to the shopping cart. Once the selection of the catalog products is made, the purchase is finalized, indicating if home delivery is desired. The user needs to be registered to access the application. The application must be available even without a permanent network connection.

Fig. 8 presents the PIM model for the *E-market* example. For this, we have applied to the PIM the M2M transformation rules to obtain the ASM automatically. The result obtained after executing the applied M2M transformation rules[8] can be seen in Fig. 9.

By carefully analyzing Fig. 9, several statements may arise: 1) All the classes of the persistence model were generated (the only model in the PIM can be seen in Fig. 8), that is to say, 100% of the classes present in the PIM were automatically modified. However, three classes corresponding to the model of data providers were left for manual generation, which is reasonable, since the data provider model is particular to the mobile world; 2) the modeling is simplified since the PIM, having a level of abstraction superior to the ASM, contemplates broader concepts, and therefore, it is easier to understand and model them; 3) the portability of the generated model is considerably improved since the modeled PIM could be used to develop a mobile application or even a web page. While starting the modeling in the ASM, we do not stick to mobile development only, although there are cases, like this particular study, where this rigidity contributes more than the flexibility of the PIM; 4) as we mentioned in Section 2.2, some manual adjustments are required.

The few manual adjustments can be concluded by comparing automatically generated ASM (see Fig. 9) and manually generated ASM (see Fig. 10). Basically, there are two types of manual adjustments that were required to obtain the final ASM, starting from a PIM, through the application of rules of transformation M2M: 1) those derived of transformations that cannot be automated by the absence, in the PIM, of elements that allow deducting them (e.g., the *persistentType* tag value); and, 2) those arising from subjective design decisions, that do not require technical criteria to be taken and are more related to the final appearance of the user interface (e.g., the *selectable* tag value). However, as it was shown in Nuñez et al., 2018, it is possible, even under these circumstances, to reduce the number of manual adjustments to zero through the use of external files that allow simply capturing these design decisions in a YAML (YAM) configuration file.

In Fig. 10, we observe the application's data persistence design. One of the requirements was that the application is available even without a network connection, so we use the persistent entities (*persistentEntity*) to indicate where the data will be stored. There are five persistent entities, each with stereotyped properties with ≪*persistentEntityProperty*≫. Each persistent entity has a specific type of persistence (*persistentType*): products, shopping cart, and providers will be stored in a local database (*persistentType* = *Database*), so they have an attribute with the tag value *selectable*. The images of each product will be stored in files in a local folder of the device (*persistentType* = *File*). The user's session and their data will be stored in key-value pairs (*persistentType* = *KeyValue*).

As shown in Fig. 11, the DataProvider model represents the application data providers by defining the next three classes. First, the *RestInterface* class with the stereotype ≪*WebServiceInterface*≫ to define the interfaces of the services. Second, we create the class *HardwareDevice* with the stereotype ≪*HardwareDeviceInterface*≫, where we define the attribute *MyLocation* with a stereotype ≪*HardwareDeviceProperty*≫, which indicates we could receive location data via GPS (*hardwareDeviceType* = *GPS*) in case of home delivery is required. Third, the class *MobileAppData* with the stereotype ≪*MobileDataInterface*≫ allows defining the application that may receive a shared URL as data (*receivedType* = *URL*). Through this, any other application can interact with this application and execute some actions in response.

From the models defined above, we obtain the generated code for both Android and Windows Phone. Taking as an example part of the application, in Appendix A, we present the data edition screens for the persistent entity *"ShoppingCart"* (with *Database* as a defined storage mechanism). For this article, we present only the respective generated code for Android[9]. First, a model of the entity is generated (see Listing 5). Then, the necessary database table configuration (see Listing 6), with the specific CRUD database functionalities for this model are generated (see Listing 7). Furthermore, as a means to ease the development task, we present some of the helper classes (see Listing 8), as well as part the

[8] For more details on how the M2M transformation rules were defined, see Nuñez et al., 2018.

[9] The full project for Android and Windows Phone application can be found in the *MDD+ project* repository: http://bit.ly/2QfHQu3.

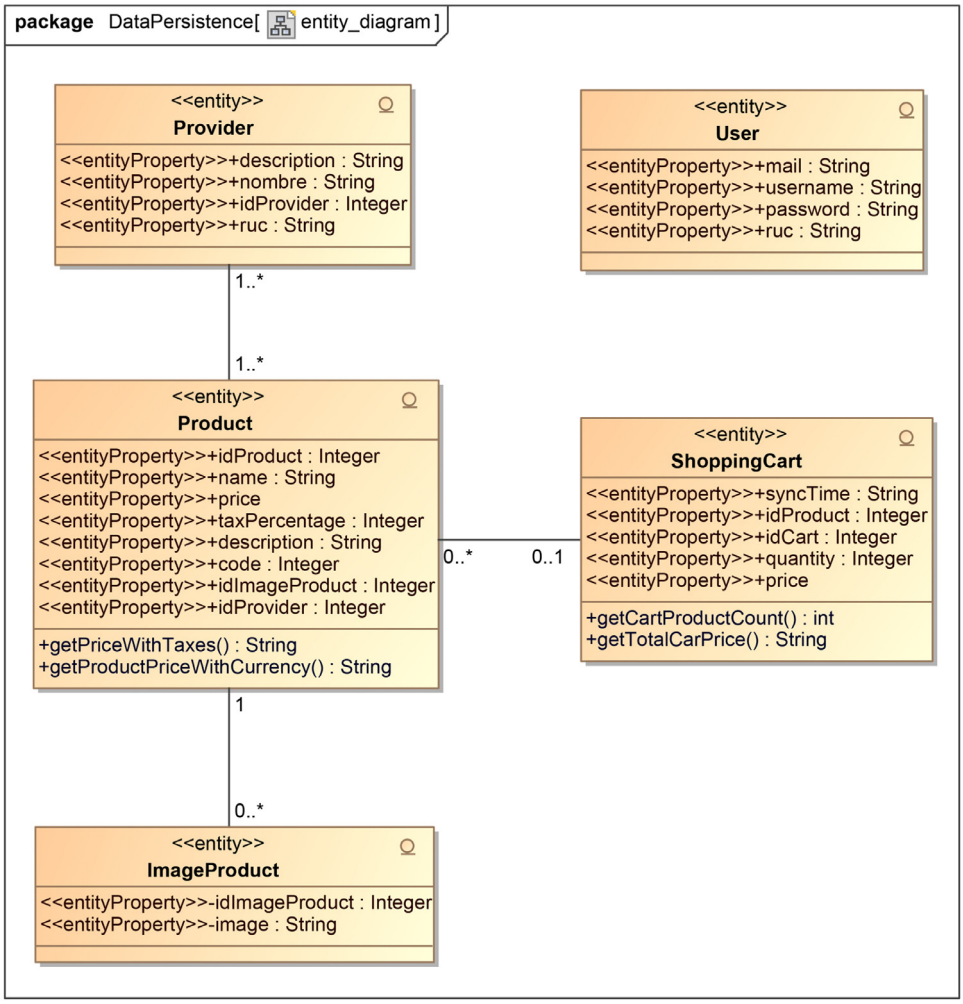


Fig. 8. PIM model for the *E-market* application example.

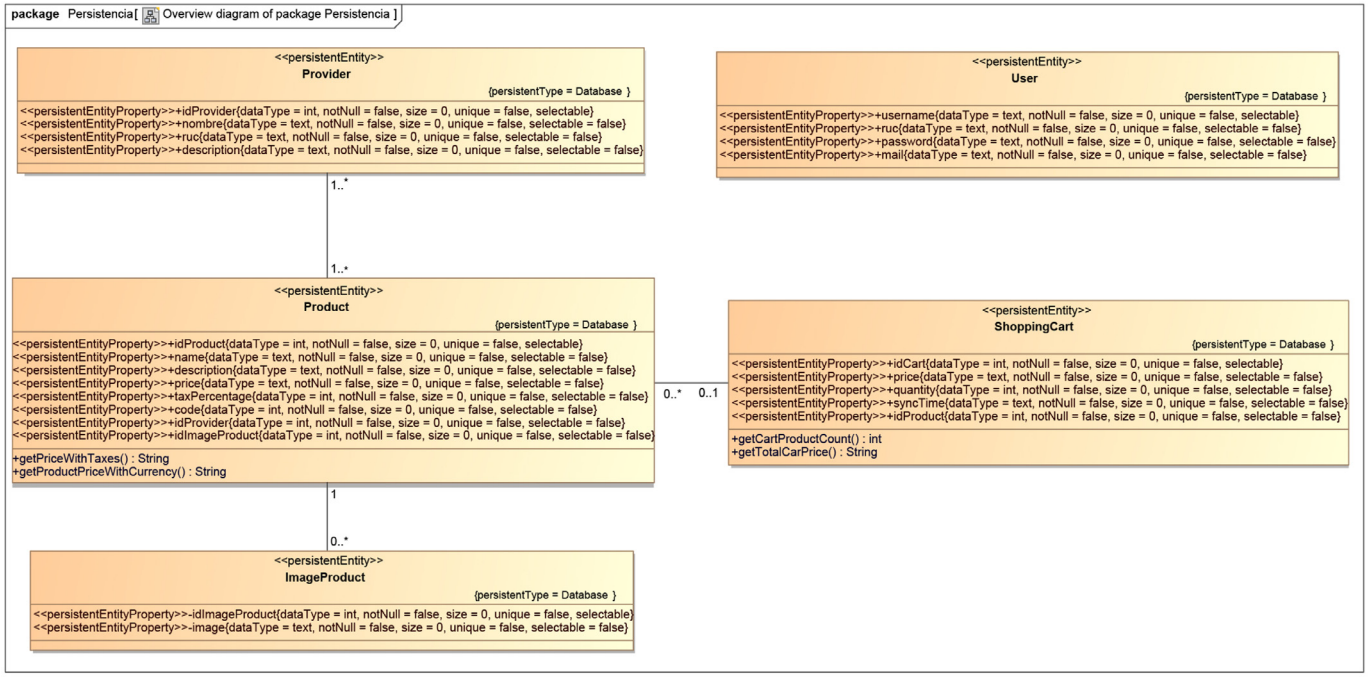


Fig. 9. ASM generated for *E-market* with the M2M transformation rules.

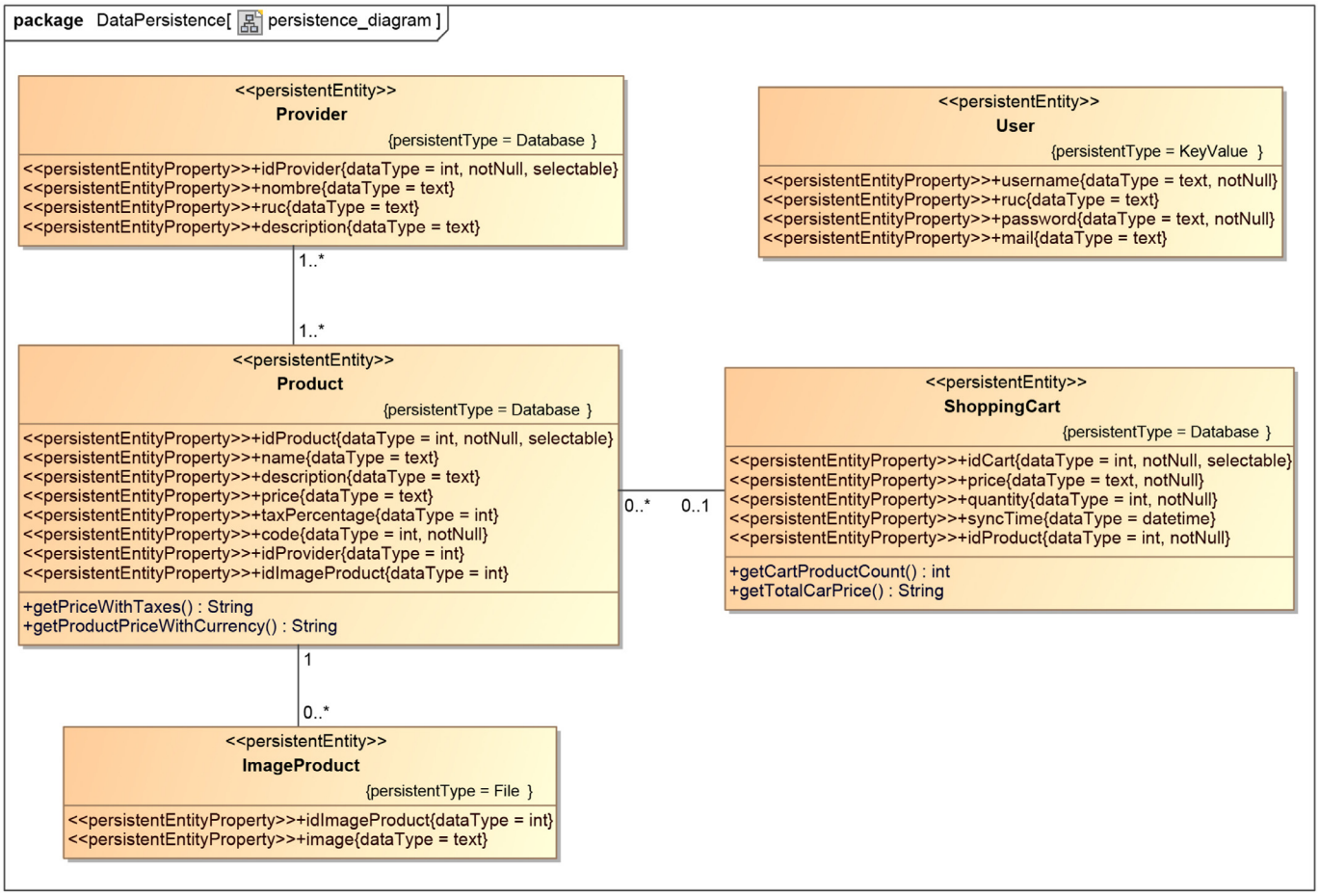


Fig. 10. Data Persistence model designed for *E-market*.

network communication created (see Listing 9), and utility classes (see Listing 10). Finally, as part of the user interface of the application generated (see Listing 11), we observe the Java code for this screen, which presents some data from the database as a form, and the option to update this data and make an update in the database.

This generated code should be compiled in the respective mobile platform IDEs. As a result, we obtain an application that allows testing all the defined functionalities. In Fig. 12, we observe the main screen of the application and the data edition screens (with CRUD functionalities) for both platforms.

5. Evaluating the usability and portability of moweba mobile

In this section, we present a first evaluation of the MoWebA Mobile proposal for native mobile applications, considering the usability and portability aspects.

For this evaluation, we were guided by the activities suggested by Wohlin et al., 2012 for case studies. Even though we follow the guidelines of such a research method, our experience of validation does not correspond to a case study because it was applied in an academic context instead of a real one (Wohlin et al., 2012). The following sections will describe each of these activities.

5.1. Evaluation setup

As mentioned in Section 4.2, it is a possibility to start the development process with MoWebA Mobile from the PIM modeling to obtain the code for different platforms, including those not mobile (e.g., RIAs). The validation of the M2M transformation rules, which allow obtaining the ASM automatically from the PIM, was already presented in Nuñez et al. (2018). Thus, interested in mobile platforms, the actual evaluation focuses on validating the M2T transformation rules that allow obtaining the final code from the ASM.

First of all, we focus on evaluating the usability of our approach to get the first evaluations of the end user's experience. The International Organization for Standardization (ISO 9241-11) (Iso, 1998) defines usability as *"the degree to which a product can be used by specific users to achieve specific objectives with effectiveness, efficiency, and satisfaction in a specific context of use"*. In this context, *effectiveness* is defined as the accuracy and completeness with which users can achieve specific objectives; *efficiency*, dedicated resources concerning effectiveness; and *satisfaction*, freedom of disagreement, and positive attitudes towards the use of the product. This evaluation could help us to make improvements and corrections to achieve an increasingly stable, consistent, reliable, and easy to use approach.

Secondly, we are interested in evaluating the portability of our approach. The fragmentation problem increases the effort of developing mobile applications as each mobile platform handle data persistence mechanisms differently. Systems and Software Quality Requirements and Evaluation or SQuaRE (ISO/IEC 25010:2011) (Iso, 2011) defines portability as the degree of effectiveness and efficiency with which a system, product, or component can be transferred from one hardware, software or another operating environment to another. Muñoz Riesle et al., 2015 emphasize that the metrics defined in ISO 9126 does not contribute to the evaluation of portability in the MDD approach, because this approach focuses on

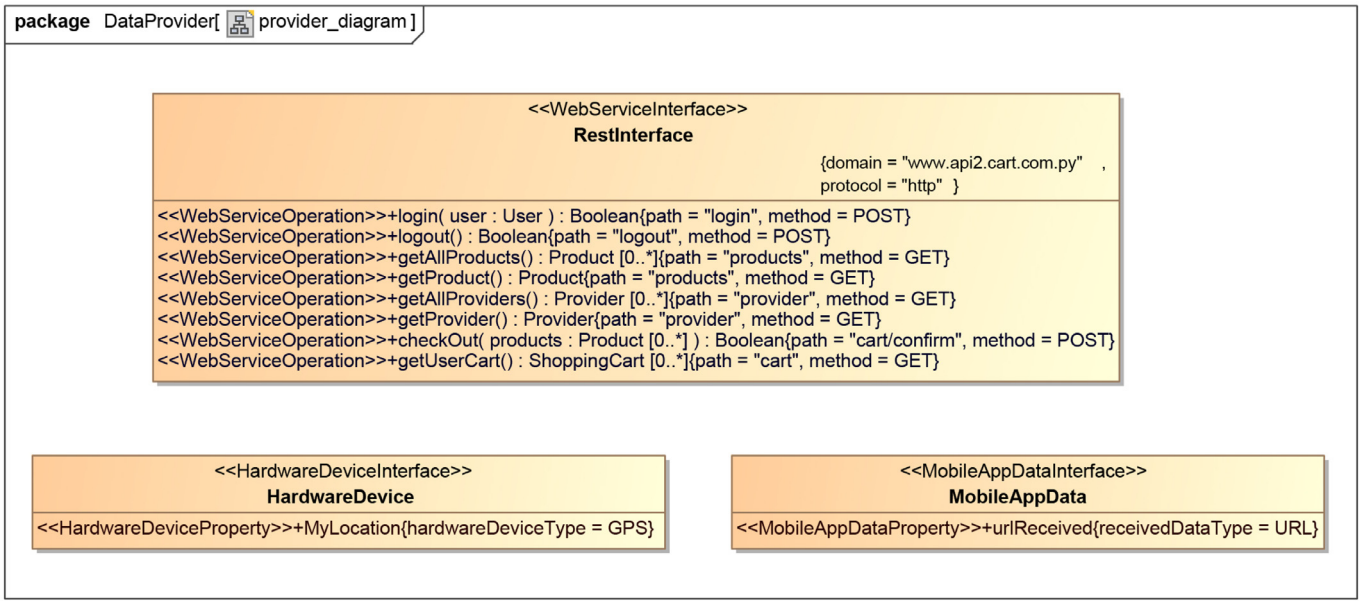


Fig. 11. Data Providers model designed for *E-market*.

re-using the developed software. We mentioned that some aspects of MoWebA Mobile could have a positive impact on the portability problem. Taking this into account, we find limitations searching for alternatives and methods for evaluating the portability of our approach. We chose to obtain some first approximations of this quality of the software, making tests with the generated application and collecting the perception of the participants in this respect. This analysis would help us to obtain first approximations and then to further validate portability in future work.

5.1.1. Design: Goal and research questions

To define the goal of the evaluation, we used the GQM paradigm (*Goal Question Metric*) (Basili and Rombach, 1988). We established the modeler and developer profiles. The first is a person with enough knowledge in modeling with MoWebA. The second is a person with experience in the development of mobile applications, able to understand and make changes to the generated code. The defined **goals**(G) are:

G1. *Analyze* the MDD approach for the development of native mobile applications focused on the data layer: MoWebA Mobile, *for the purpose of* a better understanding *in regard to* the usability *from the viewpoint* of the modeler *in the context of* mobile application development.
G2. *Analyze* the MDD approach for the development of native mobile applications focused on the data layer: MoWebA Mobile, *for the purpose of* a better understanding *in regard to* the usability and portability *from the viewpoint* of the developer *in the context of* mobile application development.

The **case analysis** consisted of the development of the *E-market* application for online purchases (see Section 4.2.4 for more details).

The complete development of the mobile application can be divided into the following **development stages**: (i) modeling with MoWebA Mobile; (ii) code generation from the model; (iii) generation of the application from the generated code; and, (iv) modifications to the generated application.

From the aspects mentioned above, and to achieve the stated goals, we established **research questions** (RQ).

Considering the first goal (G1) we have:

RQ1) What effectiveness, efficiency, and satisfaction does the modeling process of the proposed approach present?
RQ2) What effectiveness, efficiency, and satisfaction does the code generation process of the proposed approach present?
RQ3) What perception of satisfaction does the proposed MDD approach present?

From the second goal (G2) we have:

1. What effectiveness, efficiency, and satisfaction does the process of generating the application of the proposed approach present?
2. What effectiveness, efficiency, and satisfaction does the process of modifying the generated application present?
3. What perception of satisfaction does the proposed MDD approach present?
4. What perception of portability does the proposed MDD approach present?

From the established goals, we classify the **participants of the experience** with two profiles:

- *Modelers*: six students of the 8th semester of the Informatics Engineering career at the Catholic University "Nuestra Señora de la Asunción". At the time of the experience, they were finishing the Software Engineering I subject, with enough knowledge in modeling with MoWebA after they had been evaluated for a semester developing a complete application with MoWebA.
- *Developers*: five mobile developers, four of them with experience working with Android applications, and one developer with Windows Phone applications development experience. Using networking, we contacted a group of mobile developers active in the industry at that time. We started with our laboratory colleagues, and they, in turn, invited their acquaintances. The average development experience was 1.5 years in the industry.

We believe both profiles are significant enough for the type of experience conducted to achieve our first guesses of the usability and portability of the proposed approach.

According to the stages of this evaluation experience, the modeler was in charge of the modeling process and code generation. On the other hand, the developer was in charge of generating the

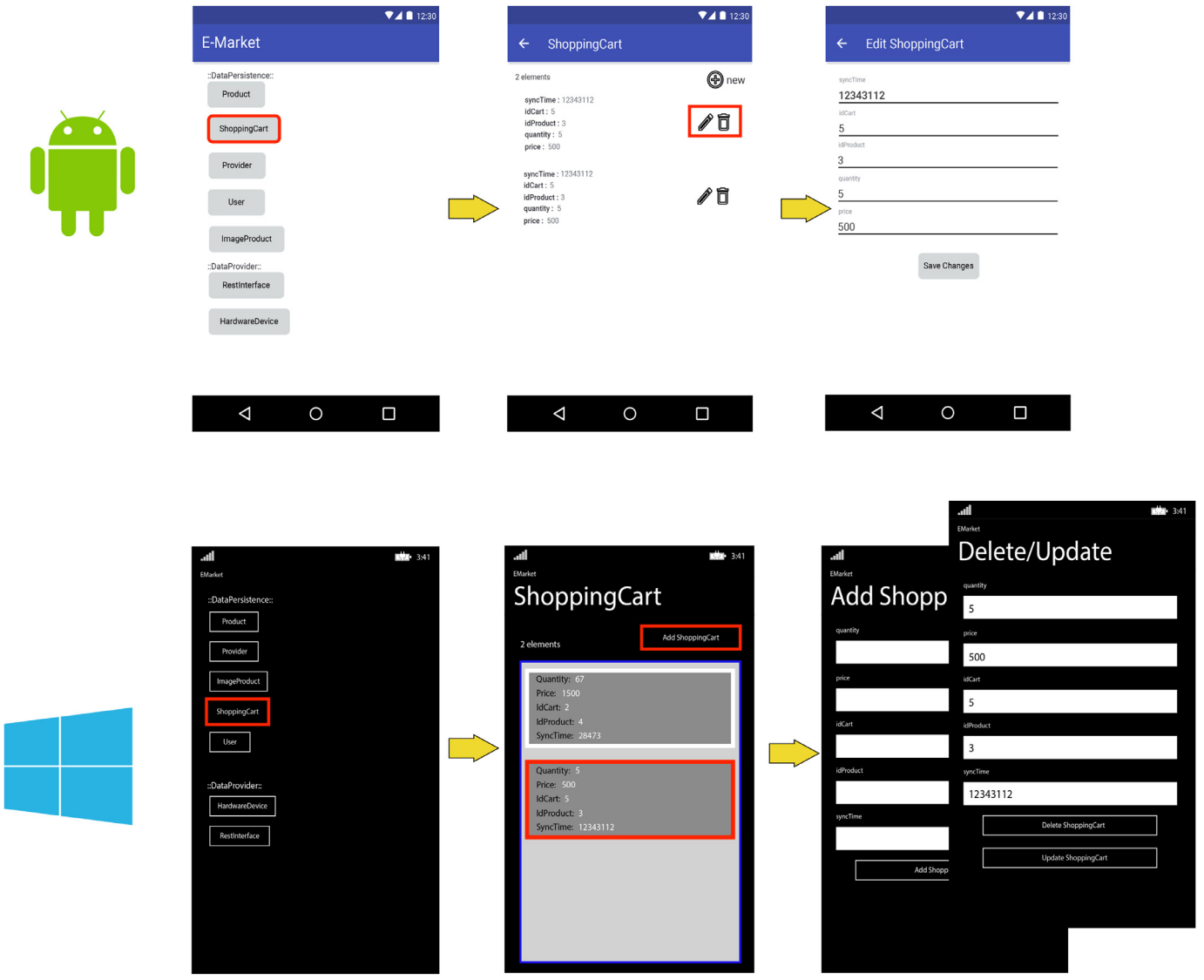


Fig. 12. Edit a *"ShoppingCart"* item in Android and Windows Phone. Starting with the main screen, the *"ShoppingCart"* option is selected. Next, a screen to list data of type *"ShoppingCart"* is presented. Here we can select the options to Edit and Delete one of these entries from the database. Finally, an item is selected for edition and update.

application from the generated code and making modifications to the application.

5.2. Planning

The experience was carried out in three work sessions. In the first and second sessions, we worked with modelers, and in the third session, with the developers. A researcher was in charge of conducting and supporting the experience.

In the first session (150 min long), the researcher presented MoWebA Mobile to the modelers, and together they worked in a mobile application case example using MoWebA Mobile, showing modeling and code generation processes.

In the second session (140 min long), the researcher presented the application *E-market* to the modelers. The modelers received a document with the system requirements. Modelers were asked to obtain the data persistence and data provider models. Then, they started the modeling process. This activity was divided into two stages: first, modeling without the support of the researcher, and then modeling with the support of the researcher. This whole process was timed to obtain data related to effectiveness. At the end of modeling, the modelers answered an *After Scenario Questionnaire* (ASQ). This questionnaire focuses on the measurement of the satisfaction of a person concerning the accomplishment of a task (Bangor et al., 2009), with a score ranging from 1 to 7, where values closer to 1 indicate a higher value of satisfaction (Tullis and Albert, 2013). Afterward, the modelers continued with the code generation process from the models they have made. Again, this process was timed. At the end of the code generation, the modelers had another ASQ. Concluding this session, the modelers answered a *System Usability Scale* (SUS) questionnaire about the entire session, which focuses mainly on the measurement of user satisfaction, covering a variety of the system usability aspects (Bangor et al., 2009).

Finally, in the third session (203 min long), the researcher presented MoWebA Mobile and the application *E-market* to the developers. The process to import the generated code to the IDEs was explained. The developers received a document with the requirements of the system. They were asked to import the generated code to the respective IDEs and generate the application. This process was timed to obtain data related to effectiveness. At

the end of this process, the developers answered an ASQ. In this case, and a difference with the previous session, we complement this ASQ with additional questions in order to collect more information, specifically asking about the comments and opinions of the participants. Subsequently, as a new task, the developers tested the generated application, made verification of the generated functionalities in the different mobile phones, and experienced the generated application in Android and Windows Phone platforms. This process focused on to get the first approximations about the portability of the proposed approach, for which we employed a *questionnaire with open questions* and asked the developers to answer it. With this questionnaire, we collected opinions from the developers regarding the possible differences in the functionalities and mechanisms of persistence, in the different platforms, and the usefulness of the approach for the automatic generation of mobile applications for different platforms. Afterward, the developers made modifications to the generated application, which consisted of changing the type of persistence mechanism of an entity. The developers were divided into two groups. The first group made modifications directly to the generated application (manual modifications). The second group made changes to the models (automatic modifications) and followed all the development steps with MoWebA Mobile to generate the application with the changes required. This modification process was timed, and the modifications made were saved separately. At the end of this process, the developers answered an ASQ. Concluding this session, the developers responded to a SUS questionnaire about the entire session.

5.3. Data collection

From the experience with modelers and developers, quantitative and qualitative data were obtained. These data allowed answering the research questions and improving the understanding of the perception collected.

We considered as **information sources** for data collection: (i) the *project documentation*, which includes an ASM model without corrections, an ASM model with corrections, the generated code, the generated application code imported into the IDE with manual adjustments, and the modifications made to the generated application; (ii) the work sessions' *timesheets*; and, (iii) *questionnaires*. The **quantitative data** were collected from these three information sources. On the other hand, the **qualitative data** were obtained from the comments and opinions of the participants (added to the used questionnaires). It is worth to mention we gathered comments and opinions only from the developers, as to complement questions of the ASQ and SUS questionnaires they used, and later, from the questionnaire with open questions about portability.

In the first place, the **project documentation** allowed us to determine the success rate of both modelers and developers. From modelers, we collected the correct ASM elements modeled without the researcherâs intervention, the number of corrections made to the models, and the generated code. These data helped us to determine success rates for ASM modeling and code generation. Furthermore, from developers, we collected the generated application code imported into the IDE with manual adjustments, and the modifications made to the same application. With this data, we find out the success rates for the application generation and the modifications made to the generated application.

Therefore, the **timesheets** permitted us to determine the completion time of each process. From modelers it was the average completion time of the ASM modeling and code generation. As from developers, it was the average completion time of the application generation and subsequent modifications to it.

Moreover, we used three types of **questionnaires**: (i) ASQ, to calculate the average satisfaction for each process; (ii) SUS, to determine the measurement of user satisfaction for each session; and, (iii) a questionnaire with open questions, to get first approximations about the perception of portability of the approach. The quantitative analysis of the questionnaires was carried out as follows: for each modeler and developer, the ASQ questionnaires were analyzed, obtaining the corresponding score for the activity. Subsequently, the average score per activity was obtained. Similarly, for each modeler and developer, the SUS questionnaire scores were obtained. From this, it was possible to obtain the corresponding average score per profile, and the general average score of the experience. On the other hand, the qualitative analysis allowed us to compile the difficulties and problems that arose in each activity carried out by the developers (as a complement of the ASQ). In the same way, we collect their general opinions on the proposed approach (as a complement of the SUS): (i) correctness of the generated application considering the initial requirements; and, (ii) suggestions or comments about their development experience with MoWebA Mobile. Likewise, we obtained comments about the portability of the approach, analyzing the participants' perception of: (i) the existence of differences between the data persistence mechanisms generated for the Android and Windows Phone platforms; and, (ii) their opinions on the usefulness of the approach for the automatic generation of mobile applications for different platforms.

As a result, the **usability** of modelers and developers for each process was measured by: (i) Effectiveness, considering the project documentation, the average success rate; (ii) Efficiency, considering the timesheets, the average completion time; and, (iii) Satisfaction, considering the ASQ questionnaires, the average satisfaction. The general perception of satisfaction using MoWebA Mobile was calculated from the average result of the SUS questionnaires. On the other hand, the perception of the **portability** was considered by opinions and comments gathered from the developers.

5.4. Results

RQ1) **What effectiveness, efficiency, and satisfaction does the modeling process of the proposed approach present?**
As we mentioned, this process was divided into two stages: first, modeling without the support of the researcher, and then modeling with the support of the researcher.
Considering the tasks in this process, we remark that all modelers completed the data persistence model. Difficulties were found during the data provider modeling, especially with the services modeling. Common errors found were: some tag values misapplying and stereotypes names missing. In general, 33% of modelers were able to complete the entire process without asking for the support of the researcher. The remaining modelers completed the task but asked for some help at some point during this process.
Based on our perception, most of the modelers were able to complete this process satisfactorily. Furthermore, looking at the quantitative analysis presented in Table 3, the average success rate was 82%, which we consider as a notable success rate.

Table 3
Usability measurements of the modeling process.

Process	Average Success Rate	Average Completion Time (minutes)	Average Satisfaction (ASQ score)
Modeling	82%	64.17	2.89

Table 4
Usability measurements of the code generation process.

Process	Average Success Rate	Average Completion Time (minutes)	Average Satisfaction (ASQ score)
Code generation	100%	15.33	1.94

Table 5
Usability measurements of the application generation process from the generated code.

Process	Average Success Rate	Average Completion Time (minutes)	Average Satisfaction (ASQ score)
Application generation	98%	51.6	2.20

In regard to the completion time, we measured that more time was required during the first stage, where the modelers, with the help of the documentation provided, worked on the modeling process. After this, some questions arose in order to complete the task. As a result, the average completion time of the complete process was 64.17 min. We consider this number as reasonable, taking into account the time lost because of some errors during modeling. Nevertheless, as a first experience working with MoWebA Mobile, this modest performance draws attention to the fact they had informational materials, and the complexity of modeling was reduced. This performance gives us clue that more training could be necessary to reduce the learning curve.

From all we mentioned above, we concluded MoWebA Mobile was accepted among the modelers, thanks to their experience using MoWebA, but more training was necessary to achieve higher levels of success rate and efficiency. The score obtained from the ASQ questionnaire reflects an acceptable level of satisfaction of the modelers with the modeling process.

RQ2) **What effectiveness, efficiency, and satisfaction does the code generation process of the proposed approach present?**

In this process, all the modelers were able to complete the tasks, generating the code application satisfactorily in Acceleo. Nevertheless, errors were detected during the code generation. In total, 33% of modelers presented errors in this process due to imperfections found in their models from the modeling process. Time was lost detecting such errors, increasing the average completion time, causing a number significantly high for an automatic generation. From Table 4, the average completion time was 15.33 min. Despite this fact, we noticed good results considering the efficiency of the modelers.

From our perspective, this automatic code generation process obtained good results. We noticed a compensatory experience once the modelers generated code from their models, seeing the effort of the previous task materialized. The score obtained from the ASQ questionnaire for this code generation process, and comparing with the obtained in the modeling process, reflects a very good level of satisfaction for the modelers.

RQ3) **What perception of satisfaction does the proposed MDD approach present?**

The modelers' average score of perception of satisfaction of MoWebA Mobile was 50 in the SUS, with a standard deviation of 16.89. Converting this score to Sauro and Lewis percentile rank,[10] we obtain a value of 13%. This result is below average.

In general, we noticed some doubts in the use of the proposed approach from the modelers. With the introduction of new concepts of the approach, and considering this as the first working experience with the approach, we believe this complexity perception could improve with more training, which could minimize the learning curve.

RQ4) **What effectiveness, efficiency, and satisfaction does the process of generating the application of the proposed approach present?**

As we mentioned in *RQ1*, there were errors during the modeling process. In order to all the developers could reach a unified generated application and do not drag previous errors in this process, we decided to provide a 100% complete model to start the code generation process.

From the collected comments during this activity, the main problems mentioned by the developers were related to the IDE. First, problems with the configuration of the IDE, loosing time in the installation of updates and import of required libraries. This setback could have been avoided with necessary adjustments during the preparation of the experience. It is not considered as an error of the developers. Similarly, problems also occurred importing the generated code, a process that required creating a new project in the IDE, and then locating the generated code in folders.

Although, with an almost perfect success rate (see Table 5), the drawback with the use of the IDE got our attention because, despite the work experience the developers had, the tasks of creating the project and then adapting the generated code to it took longer than expected. From these results, we analyzed methods to improve the fulfillment of this process: a complete IDE project structure generation or even the generation of the final application.

We noticed that the developers were very engaged with the experience of validation, willing to finish the tasks, participatory, and striving to achieve good performance. Furthermore, in general, we perceived a good level of satisfaction during the application generation process (also reflected in the ASQ results).

Concluding, we present additional data collected to be used in later analysis. The average of generated lines of code (LoC) for Android and Windows Phone were 6411 and 3828, respectively. This number of lines corresponds to the total amount of lines of each developer's project, so there were certain variations in each case. The lines of code counted include blank lines, comments, and the *imports* in each class.

RQ5) **What effectiveness, efficiency, and satisfaction does the process of modifying the generated application present?**

The requested modification consisted of changing the type of persistence mechanism of an entity: *key-value* instead of *database* as a new persistence mechanism. We worked in two groups: the manual development involved three developers (they developed for Android platform), and the development with MoWebA Mobile approach involved two developers (one developer working with Android and another with Windows Phone platform).

Analyzing the results of the first group mentioned above, and regarding efficiency, we can see a lower performance in comparison with previous processes. Only 33% of developers could fully complete the requested modification.

[10] About the SUS questionnaire, Sauro and Lewis (2016) mention that the best way to interpret the results is normalizing and obtaining the percentile rank. Any result with a percentile rank less than 50% is, by definition, below the average, and anything above 50% is above average.

Table 6
Usability measurements on the process of modifying the generated application: code modifications.

Process	Average Success Rate	Average Completion Time (minutes)	Average Satisfaction (ASQ score)
Modifications to the code	73.3%	57	3

Table 7
Usability measurements on the process of modifying the generated application: model modifications.

Process	Average Success Rate	Average Completion Time (minutes)	Average Satisfaction (ASQ score)
Modifications to the model	72%	38.5	1.67

The other two developers reached only 50% of the development. These developers made the requested modification but could not show the data from the database on the screen. For this group, the main problem commented was that the designated time to finish all the requested changes for this task was insufficient, but they encouraged saying that with a little more practice, they could do it without inconvenience. The previous discomfort is reflected in the level of satisfaction the ASQ questionnaire returned (see Table 6).

Concerning the results of the second group, firstly, only one of the developers, who worked with the Windows Phone application, could fully complete the requested modification and test successfully the changes made. The other developer, who worked with the Android application, was able to generate the application code successfully with the requested changes, but commenting some problems creating the project and importing folder to the IDE, so he could not test the application. In this case, during the evaluation, we could test this application, obtaining excellent results. The developers who made the requested modifications to the code highlighted the readability and proper structure of the generated code. Secondly, the average activity time of this second group was 38.5 min. This average time reflects the drawbacks mentioned above with one of the developers, but it is worth noting that this developer was able to complete the activity successfully in 20 min (see Table 7).

Finally, we observed the developers using MoWebA Mobile, more confident and satisfied with the experience. Unlike the score obtained in the manual development, the ASQ score for this group reflects a very good level of satisfaction.

To summarize, all developers were able to accomplish the requested task in this stage, changing the type of persistence of the entity, but only 40% of them could complete and test the changes made.

Next, we highlight some interesting points found during the analysis of both development groups:

- *Time*: we notice a considerable difference comparing the completion time of those who could fully finish this task. It can be said that the development with MoWebA Mobile allowed making the requested modification 2.85X faster than developing it manually.
- *LoC of the requested modifications*: comparing the development for the Android platform, we found a nontrivial difference: on average, 309 lines were added developing manually, and 226 lines were added using MoWebA Mobile. We highlight the conciseness of the transformation rules, in the context of the proposed exercise.
- *ASQ*: both groups experienced different types of errors during the task, but the difference in both ASQ scores reflects a better level of satisfaction using MoWebA Mobile.
- The same modification was made manually and by the proposed approach. We emphasize that the changes to the model did not require any extra manual modification, highlighting the expressiveness of MoWebA Mobile, in the context of the proposed exercise.

RQ6) **What perception of satisfaction does the proposed MDD approach present?**

We obtained the developers' opinion regarding the general development with our approach and the coverage of the initial requirements by the generated application. Many agreed to say that since the modification process could not be completed, it was difficult for them to give an opinion on the validation of the initial requirements. Regarding the suggestions and opinions about MoWebA Mobile, they stressed that it is an easy to use approach, and interesting things could be achieved with some adjustments. The suggested adjustments include the automation of this process, that means improving the import of the generated code to the IDE.

The mentioned above is reflected in the developers' average score of perception of satisfaction of MoWebA Mobile, which was 70 with a standard deviation of 15.2. In a percentile rank, the result is above average with a value of 56%. Furthermore, compared to the perception of satisfaction obtained by the modelers (50 points in the SUS), the score of the developers is high and may have been even higher if not for the errors mentioned in *RQ4* and *RQ5*. However, in general, analyzing the SUS questionnaires of the developers, and unlike the modelers, the proposed approach was well accepted, considering it consistent and not complicated.

RQ7) **What perception of portability does the proposed MDD approach present?**

We emphasize the activities about portability were carried out successfully by all the developers. By loading data, the developers were able to test the application and the data persistence mechanisms available. Then, they carried out the same tests in different mobile OSs.

Through the questionnaire, we were able to obtain the perception of the developers about the portability of MoWebA Mobile. Here is a summary of the answers gathered:

- *Did you notice any difference concerning the functionalities?* In general, all agreed there were no problems with data persistence, and they were able to test the different data persistence mechanisms successfully in the different platforms. However, they highlighted that there were problems with some data providers.
- *Do you find MoWebA Mobile useful?* Everyone found the proposal useful, noting the savings in effort and time in code generation is remarkable in this type of development.
- *The problems in this scenario* were with the data providers. These drawbacks were presented using some sensors like GPS and gyroscope, and some specific hardware such as the camera.

The experience towards the first approach of evaluation of portability was positive. The comments from developers allowed us to verify the data persistence mechanisms worked correctly in different versions of the mobile OS. We highlight that there were problems with some data providers and some interface details, which challenges us to continue improving to achieve a more robust approach.

5.5. Threats to validity

Considering Wohlin et al. (2012) threats to validity definitions, some aspects that may have attempted against the validity of this first evaluation and how they were mitigated, are discussed below:

- **Internal validity**: Two well-differentiated groups were formed, balancing the level and area of experience. First, the student/modelers (with sufficient knowledge in modeling with MoWebA). Second, the mobile developers (with an average experience of 1.5 years in the industry). The experience was carried out during Software Engineering I class time to avoid the absenteeism of the students/modelers during the sessions. The performed tasks were considered as possible topics for the Software Engineering I exam. Regarding the developers, we were in constant communication with them, organizing a schedule in which everyone could attend. Finally, to avoid plagiarism in the experience, each participant worked individually in different machines and was supervised to avoid communicating with each other.
- **External validity**: This aspect could be affected by the number of participants involved (six students/modelers and five mobile developers). Although this number is not statistically relevant, it is appropriate to issue an initial evaluation and initial judgments. More formal validations should be carried out later on (such as experiments and case studies) with a more significant number of participants, to obtain meaningful and precise conclusions. Besides, the development case was simple, but not far from the requirements that an industry case could imply. This case contemplated the development of a mobile application taking into account all aspects covered by MoWebA Mobile.
- **Construct validity**: The selected measures are typically used to measure the aspects of software quality. Specifically, standard questionnaires considered reliable and valid were used to measure usability: SUS (Bangor et al., 2009) and ASQ (Lewis, 1991). However, the measurement of portability could have been affected by not having standard questionnaires or accurate metrics, although we consider it appropriate to issue the first approximations to perform more formal validations in future works.
- **Reliability**: To avoid introducing biases through interpretation, the chain of evidence of all information sources involved in the experience was carried out respecting the literalness of the obtained data.

6. Discussion of results and future research directions

The usability evaluation results in MoWebA Mobile, in general, were good. All stages of development with MoWebA Mobile thrown good results in usability satisfaction. The generation of code and the process of modifying the application with MoWebA Mobile returned very good results. It was not so with the process of manual modification of the generated application code, which returned the lowest satisfaction score, followed by the modeling process.

To talk about the satisfaction of MoWebA Mobile, we have to consider modelers and developers' points of view. On the one hand, the modelers perceived the approach to be somewhat complex, feeling somewhat insecure using the approach and with the need to learn many things to handle the approach. This result gave us clues about more training could be necessary to reduce the learning curve. On the other hand, the proposed approach was well accepted by the developers, considered consistent and not complicated. Although this experience was interesting and useful to analyze the perception of both profiles, we believe it is necessary to carry out more experiences contemplating the entire development process, and thus obtain a unified perception of the general usability of the approach.

Comparing the manual development of mobile applications against MoWebA Mobile, we state that in the context of the experience carried out the development with MoWebA Mobile involves fewer steps. Besides, working MoWebA Mobile involves a learning curve regarding modeling using the mobile profile, but without needing prior knowledge of each mobile platform. Although the application generated focuses only on the data layer, the generated code for both platforms helps initiate mobile development projects focused on this type of application. In the same way, making modifications both manually and with MoWebA Mobile, we highlight MoWebA Mobile received a better perception of the usability from developers. MoWebA Mobile allowed the changes to be made much faster than manually, in addition to adding fewer lines. We emphasize the expressiveness of the models, considering the modifications to the model did not require any extra manual modification.

Finally, MoWebA Mobile is still recent, and not yet a mature proposal, with some limitations.

However, from our perspective, most can be considered as limits of work-in-progress that can be exceeded in the future.

- In a multi-layered architecture context, we have contemplated the modeling and generation of the data layer. A real-world mobile application development considered other aspects such as GUI and business logic, communication between layers, and cross-cutting aspects to the layers (i.e., security, configuration, and communication).
- In the mobile business application domain context, in our knowledge, MoWebA Mobile is suitable for some scenarios already. Nevertheless, considering that the business logic resides, almost exclusively on the server-side (Majchrzak et al., 2015), the lack of presentation layer design (i.e., layouts and UI components design) is one of the main limitations not contemplated yet. Summoning other aspects also not contemplated yet, we have data synchronization, data binding, and input validations.
- Our modeling scope needs to be extended to a more detailed data structure design having in mind the relationships between entities, as well as improving our REST network communication design and code generation (at this moment, only the REST API skeleton is generated).
- A relatively high learning curve, and lengthy modeling and generation process, associated with the use of different tools. First, the process of modeling using MagicDraw, and later, export this model to Acceleo to generate the code. Finally, the generated code has to be imported to an IDE to get the final application. Besides, it is still necessary a minimum intervention of the developer to obtain the final application from the generated code.
- The experience of validation we have done is the first evaluation. In order to have a more formal validation and conclusions, it is necessary to conduct experiments and further studies.
- As output platforms, we consider only Android and Windows Phone. It would be interesting to take advantage of our ASM to contemplate other architectures and platforms.
- We did not contemplate a reference architecture in the development of our approach.

These limitations, rather than affecting the value of the proposal, should be taken into account for improving and expanding the approach. Therefore, as future research directions, we propose:

- Add other layers to MoWebA Mobile to contemplate other mobile development aspects like GUI and business logic. Fur-

thermore, consider communication between layers, and cross-cutting aspects to the layers (i.e., security, configuration, and communication).
- To improve our business application development capabilities, we need to think about layer design (i.e., considering layouts and UI components) and other related aspects like data synchronization, data binding, and input validations.
- Extend our meta-model, profile, and transformation rules contemplating the relationship between entities in our modeling. Furthermore, integrating our work with the proposal of Sanchíz et al. (2018). The authors present an extension of MoWebA focused on network communication between mobile applications and their functions in the cloud. In this sense, this is an aspect of mobile network communication we partially cover. This integration would add a more precise REST network definition, as well, extending our data persistence analysis to the cloud.
- Build a specific tool for making the modeling process simpler and faster. For instance, the creation of classes, properties, relations, and tag values is tedious in a general-purpose tool, as is MagicDraw. The specific tool should also integrate the process of generation of code. At this moment, we have to export our models from MagicDraw to Acceleo to run the transformation rules and, consequently, to obtain the code. Furthermore, based on MD2, an automatic application generation without the need to compile it first using an IDE would improve the development process.
- Design and evaluation of industry applications using MoWebA Mobile, considering the evaluation of more software qualities (e.g., maintainability and efficiency), and extending our first usability and portability evaluations to more formal experiments.
- Extension of transformation rules to contemplate other platforms (e.g., iOS) and architectures (e.g., smartwatches).
- Adopt new and promising trends in regards to MDD, such as the reference architecture proposed by Evers et al. (2016) and refined by Ernsting et al. (2016a). This proposal aims to facilitate the understanding of the archetype's structure (i.e., the meta-model and its instances) of the approaches that adopt it. Accordingly, making the process of approaching new platforms easier, such as mobile (e.g., iOS), and in our case, websites (e.g., new architectures, such as SOA or REST, or new programming languages, such as Python or Java), or even Internet of Things (IoT) platforms.

7. Conclusion

The design of the data layer for mobile applications is a critical issue to assure the constant access to remote data, making them available offline in case of network connectivity problems. Besides, another issue is to facilitate the portability of the implemented solutions due to the variety of mobile OSs and platforms (fragmentation phenomenon) that handles data storage differently. As an attempt to deal with these issues, we presented MoWebA Mobile, an MDD approach, through the extension of MoWebA, for the development of native mobile applications focused on the data layer. MoWebA Mobile defines meta-models and ASM profiles, and transformation rules for the generation of Android and Windows Phone applications, which allows testing of the applications' functionalities.

This proposal, apart from considering different mechanisms to save data to local storage, considers different data providers for mobile applications. Also, considering the multi-layer architecture, conceptually, it is framed within a data layer, allowing the expansion of the approach towards other layers. The MoWebA Mobile process was illustrated by means of modeling, design, and development of a typical business mobile application. It includes an ASM layer (to ensure the portability of the architecture) as well as the transformation rules for the generation of functional applications, with the possibility to work offline, for Android and Windows Phone platforms. The generated application also contemplates a graphical interface and provides useful functionalities that facilitate the task of the mobile developer, such as REST interfaces skeleton, database CRUD operations, and helper classes with functions to handle different data persistence mechanisms. Sanchíz et al. (2018), in a complementary study to ours, present a complete MoWebA extension for REST API modeling as part of its specialization on network communication between mobile applications and their functions in the cloud. Summarizing, the main contributions of MoWebA Mobile and this study are:

- A Meta-Model for the mobile ASM;
- The transformation rules from PIM to ASM;
- The rules for automatic code generation for two different platforms (i.e., Android and Windows Phone);
- A detailed study about mobile applications, focusing on the domain of business applications, data persistence on mobile phones, MDD, and the MoWebA approach;
- A first evaluation of the MoWebA Mobile proposal through a mobile application development experience.

The first evaluation results with MoWebA Mobile, in general, were good in terms of usability satisfaction, the generation of code and the process of modifying the application, and the portability (i.e., data persistence mechanisms worked correctly in different versions of the mobile OS). Despite the positive results, the experience helped us to identify some specific points to be improved for obtaining an increasingly consistent version of the approach. Specifically, to deal with the perception of the complexity of the approach and its learning curve for the modeling process, further validation of the proposal (e.g., formal experiments or case studies) in industrial or commercial contexts, is still pending. Finally, this experience opens the expectation about the potential of MoWebA Mobile to expand and generate applications dealing with other layers' problems and become a more comprehensive approach for developing mobile applications.

Declaration of competing interest

The authors report no conflicts of interest. The authors alone are responsible for the content and writing of this article.

CRediT authorship contribution statement

Manuel Núñez: Conceptualization, Software, Validation, Formal analysis, Investigation, Data curation, Writing - original draft, Writing - review & editing, Visualization. **Daniel Bonhaure:** Software, Writing - review & editing, Validation. **Magalí González:** Methodology, Writing - review & editing, Resources, Funding acquisition. **Luca Cernuzzi:** Supervision, Writing - review & editing, Resources, Validation.

Acknowledgment

We thank Linda Riquelme, Emanuel Sanchiz, Guido Nuñez for useful discussions. We thank Nathalie Aquino for guidance and planning, and Ruben Jimenez for assistance during the experience and validation. We also thank Aleksandra Budzyńska for her support with the article. CONACYT had co-funded this study through the PROCIENCIA program under the project "Improving the software development process: a proposal based on MDD" (14-INV-056).

Appendix A. Automatic code-generation samples for the Android *E-market* application using MoWebA Mobile

```
public class ShoppingCart implements Serializable{
    private BigDecimal syncTime; //the syncTime attribute.
    private Integer idProduct; //the idProduct attribute.
    private Integer quantity; //the quantity attribute.
    private String price; //the price attribute.
    private Integer idCart;//the idCart attribute.
    // Empty Constructor.
    public ShoppingCart (){
    }

    //Constructor.
    public ShoppingCart (BigDecimal syncTime, Integer idProduct,
    Integer quantity, String price, Integer idCart){...}

    public BigDecimal getSyncTime() {...}

    public Integer getIdProduct() {...}

    public Integer getQuantity() {...}

    public String getPrice() {...}

    public Integer getIdCart() {...}

    public void setSyncTime(BigDecimal syncTime) {...}

    public void setIdProduct(Integer idProduct) {...}

    public void setQuantity(Integer quantity) {...}

    public void setPrice(String price) {...}

    public void setIdCart(Integer idCart) {...}

    public String getTotalCarPrice() {...}

    public int getCartProductCount() {...}

}
```

Listing 5. ShoppingCart bean model.

```
public class ShoppingCartTable {
    private ContentResolver contentResolver;

    // DATABASE TABLE
    public static final String TABLE_NAME = "shoppingCart";

    public static final String COLUMN_ID = "_id";
        public static final String COLUMN_SYNCTIME = "syncTime";
        public static final String COLUMN_IDPRODUCT = "idProduct";
        public static final String COLUMN_QUANTITY = "quantity";
        public static final String COLUMN_PRICE = "price";
        public static final String COLUMN_IDCART = "idCart";

    // DATABASE CREATION SQL STATEMENT
    private static final String CREATE_TABLE = "create table "
            + TABLE_NAME
            + "("
            + COLUMN_ID + " integer primary key autoincrement,"
            + COLUMN_SYNCTIME + " numeric,"
            + COLUMN_IDPRODUCT + " integer NOT NULL ,"
            + COLUMN_QUANTITY + " integer NOT NULL ,"
            + COLUMN_PRICE + " text NOT NULL ,"
            + COLUMN_IDCART + " integer NOT NULL );";

    public ShoppingCartTable(ContentResolver contentResolver) {...}

    public static void onCreate(SQLiteDatabase database) {...}

    public static void onUpgrade(SQLiteDatabase database,
        int oldVersion, int newVersion) {...}

}
```

Listing 6. ShoppingCart database table definition.

```
public class ShoppingCartDAO {
    private SQLiteHelper mySQLiteHelper;
    //Constructor
    public ShoppingCartDAO(SQLiteHelper mySQLiteHelper) {...}
    // Adding new ShoppingCart
    public void addShoppingCart(ShoppingCart shoppingCart) {...}

    // Getting single ShoppingCart
    public ShoppingCart getShoppingCart(int id) {...}

    // Getting All ShoppingCart
    public List<ShoppingCart> getAllShoppingCart() {...}

    // Getting shoppingCart Count
    public int getShoppingCartCount() {...}

    // Updating single shoppingCart
    public int updateShoppingCart(ShoppingCart shoppingCart) {...}

    // Deleting single shoppingCart
    public void deleteShoppingCart(ShoppingCart shoppingCart) {...}

    // Deleting all shoppingCart
    public void deleteAllShoppingCart() {...}
}
```

Listing 7. ShoppingCartDAO - CRUD functionalities definition.

```
public class SQLiteHelper extends SQLiteOpenHelper {
    public static final String TAG = "Tag";
    public static final String DATABASE_NAME = "dataPersistence.db";
    public static final int DATABASE_VERSION = 1;

    public SQLiteHelper(Context context) {
        super(context, DATABASE_NAME, null, DATABASE_VERSION);
    }

    @Override
    public void onCreate(SQLiteDatabase sqLiteDatabase) {
        Log.d(TAG,"Creating database " + DATABASE_NAME + "with version " + DATABASE_VERSION);
        ProductTable.onCreate(sqLiteDatabase);
        ProviderTable.onCreate(sqLiteDatabase);
        ShoppingCartTable.onCreate(sqLiteDatabase);
    }

    @Override
    public void onUpgrade(SQLiteDatabase sqLiteDatabase, int oldVersion, int newVersion) {
        Log.d(TAG,"Upgrading database " + DATABASE_NAME + "from version " + oldVersion + "to version" +
             newVersion);
        ProductTable.onUpgrade(sqLiteDatabase, oldVersion, newVersion);
        ProviderTable.onUpgrade(sqLiteDatabase, oldVersion, newVersion);
        ShoppingCartTable.onUpgrade(sqLiteDatabase, oldVersion, newVersion);
    }
}

public class StorageManager {
    /*
    * Saving data in sharedPreferences
    */
    public static void save(Context context, String key, String value) {...}
    /*
    * Getting value from sharedPreferences
    */
    public static String getValue(Context context, String key) {...}
    /*
    * Clear the sharedPreferences. Delete all the saved data in sharedPreferences
    */
    public static void clearSharedPreference(Context context) {...}
    /*
    * Remove or delete an specific value from the sharedPreferences
    */
    public static void removeValue(Context context, String key) {...}
}
```

Listing 8. Helpers - SQLite Helper for database table management and StorageManager for key-value management.

```
1
2 public class ApiClient {
3     public static final String BASE_URL = "http://www.api2.cart.com.py";
4     private static Retrofit retrofit = null;
5
6     public static Retrofit getClient() {...}
7 }
8 public interface ApiInterface {
9     @POST("login")
10    Call<Boolean> login( User user );
11
12    @POST("logout")
13    Call<Boolean> logout();
14
15    @GET("products")
16    Call<List<Product>> getAllProducts();
17
18    @GET("products")
19    Call<Product> getProduct();
20
21    @GET("provider")
22    Call<List<Provider>> getAllProviders();
23
24    @GET("provider")
25    Call<Provider> getProvider();
26
27    @POST("cart/confirm")
28    Call<Boolean> checkOut( Product products );
29
30    @GET("cart")
31    Call<List<ShoppingCart>> getUserCart();
32 }
```

Listing 9. Network communication - Android Retrofit library skeleton for web-services implementation.

```
1
2  public class GPSTracker extends Service implements LocationListener {
3      // The minimum distance to change Updates in meters
4      private static final long MIN_DISTANCE_CHANGE_FOR_UPDATES = 10; // 10 meters
5
6      // The minimum time between updates in milliseconds
7      private static final long MIN_TIME_BW_UPDATES = 1000 * 10 * 1; // 1 minute
8
9      private final Context mContext;
10
11     // Declaring a Location Manager
12     protected LocationManager locationManager;
13
14     boolean isGPSEnabled = false; // flag for GPS status
15     boolean isNetworkEnabled = false; // flag for network status
16     boolean canGetLocation = false; // flag for GPS status
17
18     double latitude; // latitude
19     double longitude; // longitude
20
21     Location location; // location
22
23     public GPSTracker(Context context) {...}
24
25     public Location getLocation() {...}
26
27     /**
28      * Stop using GPS listener
29      * Calling this function will stop using GPS in your app
30      * */
31     public void stopUsingGPS() {...}
32
33     /**
34      * Function to get latitude
35      * */
36     public double getLatitude(){...}
37
38     /**
39      * Function to get longitude
40      * */
41     public double getLongitude(){...}
42
43     /**
44      * Function to check GPS/wifi enabled
45      * @return boolean
46      * */
47     public boolean canGetLocation() {...}
48
49     @Override
50     public void onLocationChanged(Location location) {...}
51
52     @Override
53     public void onProviderDisabled(String provider) {...}
54
55     @Override
56     public void onProviderEnabled(String provider) {...}
57
58     @Override
59     public void onStatusChanged(String provider, int status, Bundle extras) {...}
60
61     @Override
62     public IBinder onBind(Intent arg0) { return null; }
63 }
```

Listing 10. Utilities - GPSTracker implementation.

```
public class ShoppingCartFormActivity extends AppCompatActivity {
    private Boolean booleanEditMode = false; //Determine the type of operation to be performed
    private TextInputEditText tieSyncTime, tieIdProduct, tieQuantity, tiePrice, tieIdCart;
    private Intent intent;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_form_shoppingcart);

       // {...} Variables initialization

        if (booleanEditMode) { //Edit mode
            btnSave.setText("Save Changes");
            loadShoppingCartData(); //Populating the form with the provided data
        }

        SQLiteHelper db = new SQLiteHelper(this); //Instantiating the helper
        final ShoppingCartDAO shoppingCartDAO = new ShoppingCartDAO(db); //Instantiating the
                                                                         // DAO functionalities
        btnSave.setOnClickListener(new View.OnClickListener() {...});
    }
    private void loadShoppingCartData(){...}//Retrieving all the data to show in form

    //Getting data from each field of the form, and initializing the model with these data
    private ShoppingCart getShoppingCartFromEditext() {...}

    @Override
    public boolean onOptionsItemSelected(MenuItem item) {...}
}
```

Listing 11. ShoppingCartFormActivity - Edit UI screen definition.

References

About the Object Constraint Language Specification (OCL) Version 2.4. https://www.omg.org/spec/OCL/About-OCL/ (Accessed on 08/16/2019).

About the XML Metadata Interchange (XMI) Specification Version 2.1. https://www.omg.org/spec/XMI/2.1/About-XMI/. (Accessed on 08/16/2019).

Acerbis, R., Bongio, A., Brambilla, M., Butti, S., 2015. Model-Driven Development Based on OMG's IFML with WebRatio Web and Mobile Platform. In: Cimiano, P., Frasincar, F., Houben, G.-J., Schwabe, D. (Eds.), Engineering the Web in the Big Data Era - 15th International Conference, ICWE 2015, Rotterdam, The Netherlands, June 23–26, 2015, Proceedings. Springer, pp. 605–608. doi:10.1007/978-3-319-19890-3_39.

Acerbis, R., Bongio, A., Brambilla, M., Butti, S., 2015. Model-Driven Development of Cross-Platform Mobile Applications with Web Ratio and IFML. In: 2015 2nd ACM International Conference on Mobile Software Engineering and Systems, pp. 170–171. doi:10.1109/MobileSoft.2015.49.

Acceleo Official Site. https://www.eclipse.org/acceleo/. (Accessed on 08/16/2019).

ACM Digital Library. https://dl.acm.org/dl.cfm. (Accessed on 01/28/2019).

Android Developers - Official Site. https://developer.android.com/. (Accessed on 08/03/2019).

Apple Developer - Official Site. https://developer.apple.com/. (Accessed on 08/03/2019).

Balagtas-Fernandez, F.T., Hussmann, H., 2008. Model-Driven Development of Mobile Applications. In: 23rd IEEE/ACM International Conference on Automated Software Engineering (ASE 2008), 15–19 September 2008, L'Aquila, Italy. IEEE Computer Society, pp. 509–512. doi:10.1109/ASE.2008.94.

Bangor, A., Kortum, P., Miller, J., 2009. Determining what individual SUS scores mean: adding an adjective rating scale. J. Usability Stud. 4 (3), 114–123.

Barnett, S., Avazpour, I., Vasa, R., Grundy, J.C., 2015. A multi-view framework for generating mobile apps. In: Li, Z., Ermel, C., Fleming, S.D. (Eds.), 2015 IEEE Symposium on Visual Languages and Human-Centric Computing, VL/HCC 2015, Atlanta, GA, USA, October 18–22, 2015. IEEE Computer Society, pp. 305–306. doi:10.1109/VLHCC.2015.7357239.

Barnett, S., Vasa, R., Grundy, J., 2015. Bootstrapping Mobile App Development. In: Bertolino, A., Canfora, G., Elbaum, S.G. (Eds.), 37th IEEE/ACM International Conference on Software Engineering, ICSE 2015, Florence, Italy, May 16–24, 2015, Volume 2. IEEE Computer Society, pp. 657–660. doi:10.1109/ICSE.2015.216.

Barnett, S., Vasa, R., Tang, A., 2015. A Conceptual Model for Architecting Mobile Applications. In: Bass, L., Lago, P., Kruchten, P. (Eds.), 12th Working IEEE/IFIP Conference on Software Architecture, WICSA 2015, Montreal, QC, Canada, May 4–8, 2015. IEEE Computer Society, pp. 105–114. doi:10.1109/WICSA.2015.28.

Basili, V.R., Rombach, H.D., 1988. The TAME project: towards improvement-Oriented software environments. IEEE Trans. Software Eng. 14 (6), 758–773. doi:10.1109/32.6156.

Benouda, H., Azizi, M., Moussaoui, M., Esbai, R., 2017. Automatic code generation within MDA approach for cross-platform mobiles apps. In: 2017 First International Conference on Embedded Distributed Systems (EDiS), pp. 1–5. doi:10.1109/EDIS.2017.8284045.

Bernaschina, C., Comai, S., Fraternali, P., 2018. Formal semantics of OMG's interaction flow modeling language (IFML) for mobile and rich-client application model driven development. J. Syst. Softw. 137, 239–260. doi:10.1016/j.jss.2017.11.067.

Biørn-Hansen, A., Grønli, T.-M., Ghinea, G., 2019. A survey and taxonomy of core concepts and research challenges in cross-Platform mobile development. ACM Comput. Surv. 51 (5), 108:1–108:34. doi:10.1145/3241739.

Biørn-Hansen, A., Grønli, T.-M., Ghinea, G., Alouneh, S., 2019. An empirical study of cross-Platform mobile development in industry. Wirel. Commun. Mobile Comput. 2019, 5743892:1–5743892:12. doi:10.1155/2019/5743892.

Biørn-Hansen, A., Majchrzak, T.A., Grønli, T.-M., 2018. Progressive web apps for the unified development of mobile applications. In: Majchrzak, T.A., Traverso, P., Krempels, K.-H., Monfort, V. (Eds.), Web Information Systems and Technologies. Springer International Publishing, Cham, pp. 64–86. doi:10.1007/978-3-319-93527-0_4.

Botella, F., Escribano, P., Benavent, A.P., 2016. Selecting the best mobile framework for developing web and hybrid mobile apps. In: Moreno, L., de la Rubia Cuestas, E.J., Penichet, V.M.R., García-Peñalvo, F.J. (Eds.), Proceedings of the XVII International Conference on Human Computer Interaction, Interacción 2016, Salamanca, Spain, September 13, - 16, 2016. ACM, pp. 40:1–40:4. doi:10.1145/2998626.2998648.

Botturi, G., Ebeid, E.S.M., Fummi, F., Quaglia, D., 2013. Model-driven design for the development of multi-platform smartphone applications. In: Proceedings of the 2013 Forum on specification and Design Languages, FDL 2013, Paris, France, September 24-26, 2013. IEEE, pp. 1–8. http://ieeexplore.ieee.org/document/6646646/

Brambilla, M., Cabot, J., Wimmer, M., 2012. Model-Driven Software Engineering in Practice. Morgan & Claypool Publishers doi:10.2200/S00441ED1V01Y201208SWE001.

Brambilla, M., Mauri, A., Umuhoza, E., 2014. Extending the Interaction Flow Modeling Language (IFML) for Model Driven Development of Mobile Applications Front End. In: Awan, I., Younas, M., Franch, X., Quer, C. (Eds.), Mobile Web Information Systems - 11th International Conference, MobiWIS 2014, Barcelona, Spain, August 27–29, 2014. Proceedings. Springer, pp. 176–191. doi:10.1007/978-3-319-10359-4_15.

Channonthawat, T., Limpiyakorn, Y., 2016. Model Driven Development of Android Application Prototypes from Windows Navigation Diagrams. In: 2016 International Conference on Software Networking (ICSN), pp. 1–4. doi:10.1109/ICSN.2016.7501929.

Android Studio | Android Developers. https://developer.android.com/studio. (Accessed on 08/16/2019).

Ernsting, J., Rieger, C., Wrede, F., Majchrzak, T.A., 2016. Refining a Reference Architecture for Model-Driven Business Apps. In: Proceedings of the 12th International Conference on Web Information Systems and Technologies - Volume 2: WEBIST,. INSTICC. SciTePress, pp. 307–316. doi:10.5220/0005862103070316.

Ernsting, J., Rieger, C., Wrede, F., Majchrzak, T.A., 2016. Refining a reference architecture for model-driven business apps. In: Majchrzak, T.A., Traverso, P., Monfort, V., Krempels, K.-H. (Eds.), Proceedings of the 12th International Conference on Web Information Systems and Technologies, WEBIST 2016, Volume 2, Rome, Italy, April 23-25, 2016, pp. 307–316. doi:10.5220/0005862103070316. SciTePress

Evers, S., Ernsting, J., Majchrzak, T.A., 2016. Towards a Reference Architecture for Model-Driven Business Apps. In: Bui, T.X., Sprague Jr., R.H. (Eds.), 49th Hawaii International Conference on System Sciences, HICSS 2016, Koloa, HI, USA, January 5–8, 2016. IEEE Computer Society, pp. 5731–5740. doi:10.1109/HICSS.2016.708.

Extensible Markup Language (XML). https://www.w3.org/XML/. (Accessed on 08/16/2019).

Francese, R., Risi, M., Scanniello, G., Tortora, G., 2015. Model-driven development for multi-platform mobile applications. In: Abrahamsson, P., Corral, L., Oivo, M., Russo, B. (Eds.), Product-Focused Software Process Improvement - 16th International Conference, PROFES 2015, Bolzano, Italy, December 2–4, 2015, Proceedings. Springer, pp. 61–67. doi:10.1007/978-3-319-26844-6_5.

Freitas, F., Maia, P.H.M., 2016. JustModeling: An MDE Approach to Develop Android Business Applications. In: VI Brazilian Symposium on Computing Systems Engineering, SBESC 2026, João Pessoa, Paraíba, Brazil, November 1–4, 2016. IEEE Computer Society, pp. 48–55. doi:10.1109/SBESC.2016.016.

Geiger-Prat, S., Marín, B., España, S., Giachetti, G., 2015. A GUI modeling language for mobile applications. In: 9th IEEE International Conference on Research Challenges in Information Science, RCIS 2015, Athens, Greece, May 13–15, 2015. IEEE, pp. 76–87. doi:10.1109/RCIS.2015.7128866.

Goaer, O.L., Waltham, S., 2013. Yet another DSL for cross-platforms mobile development. In: Combemale, B., Cazzola, W., France, R.B. (Eds.), Proceedings of the First Workshop on the Globalization of Domain Specific Languages, GlobalDSL@ECOOP 2013, Montpellier, France, July 1, 2013. ACM, pp. 28–33. doi:10.1145/2489812.2489819.

González, M., Cernuzzi, L., Aquino, N., Pastor, O., 2016. Developing web applications for different architectures: The MoWebA approach. In: Tenth IEEE International Conference on Research Challenges in Information Science, RCIS 2016, Grenoble, France, June 1–3, 2016. IEEE, pp. 1–11. doi:10.1109/RCIS.2016.7549344.

González, M., Cernuzzi, L., Pastor, O., 2016. A navigational role-centric model oriented web approach - Moweba. Int. J. Web Eng. Technol. 11 (1), 29–67. doi:10.1504/IJWET.2016.075963.

Google Scholar - Official Site. https://scholar.google.com.py/. (Accessed on 01/28/2019).

Grønli, T.-M., Hansen, J., Ghinea, G., Younas, M., 2014. Mobile application platform heterogeneity: Android vs windows phone vs ios vs firefox OS. In: Barolli, L., Li, K.F., Enokido, T., Xhafa, F., Takizawa, M. (Eds.), 28th IEEE International Conference on Advanced Information Networking and Applications, AINA 2014, Victoria, BC, Canada, May 13–16, 2014. IEEE Computer Society, pp. 635–641. doi:10.1109/AINA.2014.78.

Heitkötter, H., Kuchen, H., Majchrzak, T.A., 2015. Extending a model-driven cross-platform development approach for business apps. Sci. Comput. Program. 97, 31–36. doi:10.1016/j.scico.2013.11.013.

Heitkötter, H., Majchrzak, T.A., Kuchen, H., 2013. Cross-platform model-driven development of mobile applications with md^2. In: Shin, S.Y., Maldonado, J.C. (Eds.), Proceedings of the 28th Annual ACM Symposium on Applied Computing, SAC '13, Coimbra, Portugal, March 18–22, 2013. ACM, pp. 526–533. doi:10.1145/2480362.2480464.

ResearchGate - Home. https://www.researchgate.net/. (Accessed on 01/28/2019).

ScienceDirect - Home. https://www.sciencedirect.com/. (Accessed on 01/28/2019).

Springer - Home. https://link.springer.com/. (Accessed on 01/28/2019).

IDC - Smartphone Market Share - OS. https://www.idc.com/promo/smartphone-market-share/os. (Accessed on 02/02/2019).

IEEE Xplore Digital Library. https://ieeexplore.ieee.org/Xplore/home.jsp. (Accessed on 01/28/2019).

Iso, I., 2011. ISO/IEC25010:2011 Systems and software engineering – Systems and software quality requirements and evaluation (SQuare) – system and software quality models. Int. Org. Standard. 34, 2910.

Iso, W., 1998. 9241-11. Ergonomic requirements for office work with visual display terminals (VDTs). Int. Org. Standard. 45.

Jia, X., Ebone, A., Tan, Y., 2018. A performance evaluation of cross-platform mobile application development approaches. In: Julien, C., Lewis, G.A., Segall, I. (Eds.), Proceedings of the 5th International Conference on Mobile Software Engineering and Systems, MOBILESoft@ICSE 2018, Gothenburg, Sweden, May 27, - 28, 2018. ACM, pp. 92–93. doi:10.1145/3197231.3197252.

Johnson, J., 2014. Designing with the Mind in Mind, Second Edition: Simple Guide to Understanding User Interface Design Guidelines, 2nd Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

Jones, C., Jia, X., 2014. The AXIOM Model Framework - Transforming Requirements to Native Code for Cross-platform Mobile Applications. In: Filipe, J., Maciaszek, L.A. (Eds.), ENASE 2014 - Proceedings of the 9th International Conference on Evaluation of Novel Approaches to Software Engineering, Lisbon, Portugal, 28–30 April, 2014. SciTePress, pp. 26–37. doi:10.5220/0004882100260037.

Jones, C., Jia, X., 2016. An empirical evaluation of AXIOM as an approach to cross-platform mobile application development. In: Maciaszek, L.A., Cardoso, J.S., Ludwig, A., van Sinderen, M., Cabello, E. (Eds.), Proceedings of the 11th International Joint Conference on Software Technologies (ICSOFT 2016) - Volume 1: ICSOFT-EA, Lisbon, Portugal, July 24, - 26, 2016.. SciTePress, pp. 264–271. doi:10.5220/0005995902640271.

Jouault, F., Wagelaar, D., 2017a. ATL EMF Transformation Virtual Machine (research VM) - Invoking native Java methods. https://wiki.eclipse.org/ATL/EMFTVM#Invoking_native_Java_methods. Accessed at 19-02-2017.

Jouault, F., Wagelaar, D., 2017b. ATL EMF Transformation Virtual Machine (research VM) - Performance. https://wiki.eclipse.org/ATL/EMFTVM#Performance. Accessed at 19-02-2017.

Keele, S., 2007. Guidelines for performing Systematic Literature Reviews in Software Engineering. Technical report, Ver. 2.3 EBSE Technical Report. EBSE.

Ko, M., Seo, Y., Min, B.-K., Kuk, S.H., Kim, H.S., 2012. Extending UML Meta-model for Android Application. In: Miao, H., Lee, R.Y., Zeng, H., Baik, J. (Eds.), 2012 IEEE/ACIS 11th International Conference on Computer and Information Science, Shanghai, China, May 30, - June 1, 2012. IEEE Computer Society, pp. 669–674. doi:10.1109/ICIS.2012.48.

Kramer, D., Clark, T., Oussena, S., 2010. MobDSL: A Domain Specific Language for multiple mobile platform deployment. In: Proceedings of the 1st IEEE International Conference on Networked Embedded Systems for Enterprise Applications, NESEA 2010, November 25–26, 2010, Suzhou, China. IEEE Computer Society, pp. 1–7. doi:10.1109/NESEA.2010.5678062.

Lachgar, M., Abdali, A., 2017. Modeling and generating native code for cross-platform mobile applications using DSL. Intell. Autom. Soft Comput. 23 (3), 445–458. doi:10.1080/10798587.2016.1239392.

Lewis, J.R., 1991. Psychometric evaluation of an after-scenario questionnaire for computer usability studies: the ASQ. ACM SIGCHI Bulletin 23 (1), 78–81.

MagicDraw. https://www.nomagic.com/products/magicdraw. (Accessed on 02/02/2019).

Mahmoud, Q.H., Zanin, S., Ngo, T., 2012. Integrating mobile storage into database systems courses. In: Connolly, R.W., Armitage, W.D. (Eds.), ACM Special Interest Group for Information Technology Education Conference, SIGITE' 12, Calgary, AB, Canada, October 11, - 13, 2012. ACM, pp. 165–170. doi:10.1145/2380552.2380602.

Majchrzak, T.A., Ernsting, J., Kuchen, H., 2015. Achieving Business Practicability of Model-Driven Cross-Platform Apps. Open Journal of Information Systems(OJIS) 2 (2), 4–15. http://nbn-resolving.de/urn:nbn:de:101:1-201705194768

Marinho, E.H., Resende, R.F., 2015. Native and Multiple Targeted Mobile Applications. In: Gervasi, O., Murgante, B., Misra, S., Gavrilova, M.L., Rocha, A.M.A.C., Torre, C.M., Taniar, D., Apduhan, B.O. (Eds.), Computational Science and Its Applications - ICCSA 2015 - 15th International Conference, Banff, AB, Canada, June 22–25, 2015, Proceedings, Part IV. Springer, pp. 544–558. doi:10.1007/978-3-319-21410-8_42.

MetaObject Facility | Object Management Group. https://www.omg.org/mof/. (Accessed on 08/16/2019).

Min, B.-K., Ko, M., Seo, Y., Kuk, S., Kim, H.S., 2011. A UML metamodel for smart device application modeling based on Windows Phone 7 platform. In: TENCON 2011 - 2011 IEEE Region 10 Conference, pp. 201–205. doi:10.1109/TENCON.2011.6129092.

Nuñez, G., Bonhaure, D., González, M., Aquino, N., Cernuzzi, L., 2018. A model-Driven approach to develop rich web applications. CLEI Electron. J. 21 (2). doi:10.19153/cleiej.21.2.4.

Nuñez, M., 2017. Proyecto MDD+ - MowebA para Mobile - Perfiles UML. http://www.dei.uc.edu.py/proyectos/mddplus/herramientas/mowebamobile/. Accessed at 01-03-2017.

OMG | Object Management Group. https://www.omg.org/. (Accessed on 08/16/2019).

Dropbox - Official Site. https://www.dropbox.com/. (Accessed on 08/03/2019).

Patterns, M., 2009. Microsoft Application Architecture Guide, 2nd Microsoft Press.

Ribeiro, A., Silva, A.R., 2012. Survey on cross-platforms and languages for mobile apps. In: Faria, J.P., Silva, A.R., Machado, R.J. (Eds.), 8th International Conference on the Quality of Information and Communications Technology, QUATIC 2012, Lisbon, Portugal, 2–6 September 2012, Proceedings. IEEE Computer Society, pp. 255–260. doi:10.1109/QUATIC.2012.56.

Ribeiro, A., Silva, A.R., 2014. XIS-mobile: a DSL for mobile applications. In: Cho, Y., Shin, S.Y., Kim, S.-W., Hung, C.-C., Hong, J. (Eds.), Symposium on Applied Computing, SAC 2014, Gyeongju, Republic of Korea - March 24, - 28, 2014. ACM, pp. 1316–1323. doi:10.1145/2554850.2554926.

Rieger, C., Kuchen, H., 2018. A process-oriented modeling approach for graphical development of mobile business apps. Comput. Lang. Syst. Struct. 53, 43–58. doi:10.1016/j.cl.2018.01.001.

Rieger, C., Kuchen, H., 2019. A Model-Driven Cross-Platform App Development Process for Heterogeneous Device Classes. In: Bui, T. (Ed.), 52nd Hawaii International Conference on System Sciences, HICSS 2019, Grand Wailea, Maui, Hawaii, USA, January 8–11, 2019. ScholarSpace / AIS Electronic Library (AISeL), pp. 1–10. doi:10.24251/HICSS.2019.894.

Rieger, C., Majchrzak, T.A., 2019. Towards the definitive evaluation framework for cross-platform app development approaches. J. Syst. Softw. 153, 175–199. doi:10.1016/j.jss.2019.04.001.

Most Widely Deployed SQL Database Engine. https://www.sqlite.org/mostdeployed.html. (Accessed on 08/03/2019).

Muñoz Riesle, R., Marín, B., López Cuesta, L., 2015. Applying ISO 9126 metrics to

MDD projects. In: ICSEA 2015: the Tenth International Conference on Software Engineering Advances, November 15–20, 2015, Barcelona, Spain, pp. 326–332.
Google Drive - Official Site. https://drive.google.com/. (Accessed on 08/03/2019).
Sabraoui, A., El Koutbi, M., Khriss, I., 2013. A MDA-Based model-Driven approach to generate GUI for mobile applications. Int. Rev. Comput. Softw.(IRECOS) 8 (3), 845–852.
Sanchez, D., Florez, H., 2018. Model Driven Engineering Approach to Manage Peripherals in Mobile Devices. In: Gervasi, O., Murgante, B., Misra, S., Stankova, E.N., Torre, C.M., Rocha, A.M.A.C., Taniar, D., Apduhan, B.O., Tarantino, E., Ryu, Y. (Eds.), Computational Science and Its Applications - ICCSA 2018 - 18th International Conference, Melbourne, VIC, Australia, July 2–5, 2018, Proceedings, Part IV. Springer, pp. 353–364. doi:10.1007/978-3-319-95171-3_28.
Sanchiz, E., González, M., Aquino, N., Cernuzzi, L., 2017. Development of mobile applications with functions in the cloud through the model driven approach: a systematic mapping study. CLEI Electron. J. 20 (3). doi:10.19153/cleiej.20.3.6.
Sanchíz, E., González, M., Aquino, N., Cernuzzi, L., 2018. Extending MoWebA for MobileApps with functions in the Cloud. In: Proceedings of XXI Ibero-American Conference on Software Engineering, CIbSE 2018, Bogot, Colombia, April.
Sauro, J., Lewis, J.R., 2016. Quantifying the User Experience: Practical Statistics for User Research. Morgan Kaufmann.
Sommer, A., Krusche, S., 2013. Evaluation of Cross-Platform Frameworks for Mobile Applications. In: Wagner, S., Lichter, H. (Eds.), Software Engineering 2013 - Workshopband (inkl. Doktorandensymposium), Fachtagung des GI-Fachbereichs Softwaretechnik, 26. Februar - 1. März 2013 in Aachen. GI, pp. 363–376. https://dl.gi.de/20.500.12116/17386
SQLite Home Page. https://www.sqlite.org/index.html. (Accessed on 02/02/2019).
Visual Studio IDE, code editor, Azure DevOps and App Center - Visual Studio. https://visualstudio.microsoft.com. (Accessed on 08/16/2019).
Tullis, T., Albert, B., 2013. Chapter 6 - Self-Reported Metrics. In: Tullis, T., Albert, B. (Eds.), Measuring the User Experience (Second Edition). Morgan Kaufmann, Boston, pp. 121–161. doi:10.1016/B978-0-12-415781-1.00006-6.
Usman, M., Iqbal, M.Z.Z., Khan, M.U., 2014. A Model-Driven Approach to Generate Mobile Applications for Multiple Platforms. In: Cha, S.S., Guéhéneuc, Y.-G., Kwon, G. (Eds.), 21st Asia-Pacific Software Engineering Conference, APSEC 2014, Jeju, South Korea, December 1–4, 2014. Volume 1: Research Papers. IEEE Computer Society, pp. 111–118. doi:10.1109/APSEC.2014.26.
UWP Documentation - UWP app developer - Windows UWP applications | Microsoft Docs. https://docs.microsoft.com/en-us/windows/uwp/. (Accessed on 08/03/2019).
Vaupel, S., Taentzer, G., Gerlach, R., Guckert, M., 2018. Model-driven development of mobile applications for android and iOS supporting role-based app variability. Softw. Syst. Model. 17 (1), 35–63. doi:10.1007/s10270-016-0559-4.
Veisi, P., Stroulia, E., 2017. AHL: Model-Driven Engineering of Android Applications with BLE Peripherals. In: Aïmeur, E., Ruhi, U., Weiss, M. (Eds.), E-Technologies: Embracing the Internet of Things - 7th International Conference, MCETECH 2017, Ottawa, ON, Canada, May 17–19, 2017, Proceedings. In: Lecture Notes in Business Information Processing, Vol. 289, pp. 56–74. doi:10.1007/978-3-319-59041-7_4.
UML Official Site. https://www.uml.org/. (Accessed on 08/16/2019).
Wohlin, C., Runeson, P., Höst, M., Ohlsson, M.C., Regnell, B., 2012. Experimentation in Software Engineering. Springer doi:10.1007/978-3-642-29044-2.
W3C HTML - Official Site. https://www.w3.org/html/. (Accessed on 08/03/2019).
XAML overview (WPF) | Microsoft Docs. https://docs.microsoft.com/en-us/dotnet/framework/wpf/advanced/xaml-overview-wpf. (Accessed on 08/16/2019).
YAML Ain't Markup Language. https://yaml.org/start.html. (Accessed on 02/02/2019).

Manuel Núñez obtained an Engineering degree in Computer Science from the Catholic University "Nuestra Señora de la Asunción" (UC), in 2017. Since 2017 he collaborates, as an early-stage researcher, in the Dream Project at the Polish Japanese Academy of Information Technology (PJATK), Warsaw, Poland. His collaboration included the development of a crowdsourcing tool for the social inclusion of older adults. He is also working as a software developer, leading many projects of mobile application development as a freelancer. His current research interests include software engineering, data science, social informatics, big data, mobile applications development, among others.

Daniel Bonhaure obtained the Engineering degree in Computer Science from the Catholic University "Nuestra Señora de la Asunción" (UC), in 2017. In the same year, he obtained a Master's degree in Software Engineering ("Facultad de Informática within Universidad Nacional de La Plata" - Argentine). His research interests include software engineering, model-driven engineering, web engineering, among others.

Magalí González obtained an Engineering degree in Computer Science from the Catholic University "Nuestra Señora de la Asunción" (UC), Paraguay, in 2000. Since 2002 she is an Associate Professor in the Department of Electronics and Computer Science Engineering (DEI) at the UC, Asunción, Paraguay. She is teaching model-driven engineering and introduction to Computer Science. Currently, she is the Director of the DEI. She is also doing Ph.D. studies at the "Universitat Politécnica de Valencia", Spain. Her research interests include software engineering, model-driven engineering, e-governance, web engineering, web semantics, adaptive systems, among others. In these areas, she has published about 35 papers in electronic journals, book chapters, international conferences and workshops.

Luca Cernuzzi is a Full Professor in Software Engineering at the DEI - Catholic University "Nuestra Señora de la Asunción" (UC), Paraguay. He obtained a Ph.D. in Engineering (University of Modena & Reggio Emilia). Since 2009, he is the Dean of the School of Sciences and Technology at UC. His current research interests include software engineering, data science and social informatics. In these areas, he has published over 130 papers in journals, book chapters, and international conferences and workshops.

Update

The Journal of Systems & Software

Volume 169, Issue , November 2020, Page

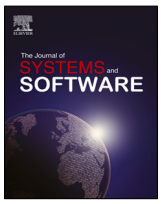
DOI: https://doi.org/10.1016/j.jss.2020.110779

The Journal of Systems & Software 169 (2020) 110779

Contents lists available at ScienceDirect

The Journal of Systems & Software

journal homepage: www.elsevier.com/locate/jss



Corrigendum

Corrigendum to "A model-driven approach for the development of native mobile applications focusing on the data layer" [Journal of Systems and Software volume 161 (March 2020)]

Manuel Núñez *, Daniel Bonhaure, Magalí González, Luca Cernuzzi

Department of Electronics and Computer Science Engineering (DEI), Catholic University "Nuestra Señora de la Asunción", Tte. Cantaluppi y G. Molinas, 1366, Asunción, Paraguay

The authors regret that one of the affiliations of Magalí González has been omitted due to an involuntary mistake.

For this reason, we would hereby like to request the inclusion of the missing affiliation. The correct details are given below:

Magalí González
Department of Electronics and Computer Science Engineering (DEI), Catholic University "Nuestra Señora de la Asunción", Tte. Cantaluppi y G. Molinas, 1366 Asunción, Paraguay

Centro de Investigación en Métodos de Producción de Software-PROS, Universitat Politècnica de València, Camino de Vera s/n, 46022 Valencia, Spain

The authors would like to apologise for any inconvenience caused.

DOI of original article: https://doi.org/10.1016/j.jss.2019.110489.
* Corresponding author.
E-mail address: manuel.nunez@uc.edu.py (M. Núñez).

https://doi.org/10.1016/j.jss.2020.110779