Universidad Católica
"Nuestra Señora de la Asunción"

Facultad de Ciencias y Tecnología
Departamento de Electrónica e Informática

MoWebA Mobile: Modeling and Generation of the Communication of Mobile Apps with their Functions in the Cloud

---

Final Project submitted for the degree of Informatics Engineering

by

Emanuel Antonio Sanchiz Flores

With the participation as project advisors

of

Ing. Magalí González and
Ph.D. Luca Cernuzzi

Asunción, Paraguay

October 2017

Acknowledgement

I would like to express my personal gratitude to:

God Almighty, for guiding me all these years and for giving me the strength to persevere and complete the degree of engineer satisfactorily.

My parents, Margarita and Francisco, for their love, advices and prayers of every day. Without them I would not have got where I am today.

My sister Lourdes, my brothers Anthony and Rodrigo, and my godparents, Santiago and Teresa, who have helped and supported me in many moments.

Advisors Magalí, Luca and Nathalie, for their patience, predisposition and advices along all this time.

Friends of all these years, who helped me and from whom I have learned so much.

Pablo Santa Cruz and Mauricio Merín, for their predisposition and advices.

This work has been funded by CONACYT through the PROCIENCIA program with resources from "Fondo para la Excelencia de la Educación e Investigación - FEEI" from FONACIDE. This work has been developed under the project "Mejorando el proceso de desarrollo de software: propuesta basada en MDD" (14-INV-056)

Contents

Chapter 1
Introduction

Currently, a growing interest is being caused by mobile applications and the cloud. In this work, we have focused on mobile applications which have functions implemented in the cloud (MobileApps-FC). Improvements related to the portability of these applications among different platforms and different service providers are a critical need. Model Driven Development (MDD) constitutes one of the alternatives to address portability issues. From a general perspective, in this work, we propose a MDD approach, called MoWebA, for the design and generation of the MobileApps-FC. Furthermore, MoWebA have a set of properties which could have a positive impact on the portability design. Consequently, it could help to alleviate the effects of the difficult of portability at the platform level. We called to our proposal, MoWebA Mobile. About the implementation scope of the modeling and generation, we have focused on the network communication between the mobile applications and their functions in the cloud. In this introduction, we present the motivation, the goals and the preliminary contribution of our proposal.

1.1 Motivation

In recent years, the number of users of smart-phones and tablets has been increasing quickly, overcoming the number of users of computers and notebooks[1]. Consequently, the interest for the development of mobile applications has also increased[2], particularly for native applications. Moreover, it is important to consider the current growing market of the cloud, specially the public cloud[3], and its role as enhancer of the restricted resources of mobile devices [1, 2]. Taking into account both perspectives, mobile devices and the cloud, it is worth noting the growing convergence between them [3]. Therefore, the interest of this study is focused on mobile applications which have at least one module or implementation (e.g., procedure, service, database) running in the cloud. We refer to these applications as mobile applications with functions in the cloud (MobileApps-FC).

MobileApps-FC are composed of a mobile environment and a cloud environment. Both of them consider different operating systems, programming languages, libraries, services, frameworks, etc. Therefore, they constitute heterogeneous environments where working can be difficult, since usually different languages and tools must be employed.

On one side, the mobile environment can be based on several different platforms, being iOS and Android the most popular ones. These platforms, again, consider different operating systems, programming languages, tools, etc. In most cases, mobile applications must be developed for more than one mobile platform. And due to differences among these platforms, specific application versions must be built for each of them. Therefore, more effort and time must be employed, resulting in higher costs in the development process. Since a same implementation can hardly be used for different mobile platforms, there is a portability problem in the mobile environment [3, 4, 5].

On the other side, the cloud environment is constituted by several different service providers. In a similar way than previously exposed, each of these providers uses its own specific platform (operating systems, programming languages, libraries, etc.). This causes a portability problem, which in the cloud context is known as vendor lock-in. Therefore, tasks such as building an application that is able to run in clouds from different service providers or migrating an application

[1] Morgan Stanley, link: https://goo.gl/Ifznvz

[2] Mobile is all about application, link: https://goo.gl/tCnZfN

[3] Growing of public clouds, link: https://goo.gl/WOVYJW

from the cloud of one service provider to another one constitute interesting challenges, since a same implementation can hardly be used in different cloud service providers [6, 7].

Therefore, when considering applications that include, at the same time, the mobile and the cloud environments, the portability problem is highlighted.

Several possible solutions have been proposed to address the portability problem in the mobile context, from mobile web applications to cross-platform compilers [8].

It is also worth noting that most of the improvement efforts related to the portability problem in the mobile context have focused on the implementation stage of the development process (for instance, using cross-platform compilers). Nevertheless, it would be beneficial to address the portability problem earlier, at the design stage. These possible improvements at the design stage could contribute to the portability of the resulting implementation.

In this sense, Model Driven Development (MDD) emerges as a possible solution. In fact, one of the main motivations of MDD is related to the improvement of portability. MDD focuses efforts on designing domain models that capture the knowledge of the data and functionality that an application must provide independently of the platform in which it will be run. In this way, one same domain model is valid for any implementation platform. Model Driven Architecture (MDA) goes one step further and defines different abstraction layers for the design and development of applications: i) Computer Independent Model (CIM), which represents an application independently of computer-based concepts; ii) Platform Independent Model (PIM), which represents an application independently of an implementation platform; iii) Platform Specific Model (PSM), which represents an application for a specific platform; and iv) code, which corresponds to the source code of the application. Furthermore, following MDA the development process can start at higher abstraction layers (CIM or PIM) and then subsequent (semi-)automatic transformations can be performed until the source code is reached. In this way, PIM and transformation rules allow the solution logic of an application to be reused for different target platforms [9, 10]. This reuse helps to achieve savings of cost, time and effort in the design and implementation of applications that have to run in different platforms.

1.2 Goals

The interest of this work is on the development of the MobileApps-FC through the model driven approach. Furthermore, for the implementation we focus on one aspect of such applications, which is the network communication. Therefore, our goal is to propose a model driven approach for the modeling and generation of the network communication of the MobileApps-FC as an alternative for addressing the extra effort caused by the difficulty of platform portability. From such goal, following we present our objectives:

- Analyze the existent MDD schemes of development of the MobileApps-FC.
- Propose a new approach for the design and generation of the network communication of the MobileApps-FC.
- Make a preliminary validation of the results of our proposal.

1.3 Preliminary Contributions of our Proposal

Our work is part of a bigger project of our department called “Mejorando el proceso de desarrollo de software: propuesta basada en MDD” (14-INV-056)[4]. Inside the mentioned project, our work contributes to the mobile applications modeling and generation area. We have already published two articles of this work. The first one, for the Latin American informatics and computer science conference (CLEI 2016 - Valparaíso, Chile) [11]. The second one, for the CLEI Electronic Journal (version 2017) [12].

[4] Project link: http://www.dei.uc.edu.py/proyectos/mddplus/

1.4 Following Chapters

Following, we describe the remaining content of this book. In Chapter 2, we introduce the concepts related to the area of our work. Some of the concepts are mobile applications with functions in the cloud, portability, REST architecture, model driven approach, MoWebA. In Chapter 3, we present our SMS of the state of the art, where we describe the planning, the execution, the results and the analysis of the SMS. In Chapter 4, we describe our proposed solution, MoWebA Mobile. Moreover, we explain the scope of the implementation, the process of modeling and generation, and an example for the modeling and generation of the network communication. In Chapter 5, we present two comparative studies with our proposal. The first study against the traditional (manual) development and the second one, against a tool used in the industry. In Chapter 6, we finalize with our contributions, limitations and future works.

Chapter 2
Research Context

This chapter presents several concepts and considerations in order to understand the context of our research. This work has been developed under the project "Mejorando el proceso de desarrollo de software: propuesta basada en MDD" (14-INV-056), funded by the CONACYT. We started our research with the study of the Rich Internet Applications (RIA) [13]. It is an architecture for web applications. RIA was one of the main subject of interest inside the mentioned project. Nevertheless, considering the relevance of the mobile applications we redirected our study towards them. In other words, we have looked for a RIA for mobile applications. In this way, we have found the Rich Mobile Applications (RMAs) [3]. Therefore, we start this chapter talking about the RMAs. Then, we explain one of its main properties, the rich functions, which in turn, suggest the functions in the cloud. Subsequently, we describe briefly some issues related to mobile and cloud portability. Such issues constitute the problem faced by our work. Following, we detail the aspects about REST for building the communication between the mobile and cloud environments. Finally, we explain the paradigm and the approach of software development we adopted in this work. Furthermore, we present aspects of such approach on which we focused our proposal.

2.2 Rich Mobile Applications, Rich Functions and Functions in the Cloud

As we already have mentioned, we started our research from RIA. We looked for a RIA for the mobile applications since the relevance of them. In this way, we have found the RMAs, which is defined as[3]: *"Energy efficient, multi-tier, online mobile applications originated from the convergence of mobile cloud computing, future web, and imminent communication technologies envisioning to deliver rich user experience via high functionality, immersive interaction, and crisp response in a trustworthy wireless environment while enabling context-awareness, offline usability, portability, and data ubiquity".*

The area of the RMAs is subject of several researches. These studies are looking for solutions which offer friendly and compelling user interfaces, wide range of functions, high capability of interaction, portability between different platforms and so on. The objective is to offer such properties besides the constraints related to a mobile device like its reduced screen size, the differences among platforms, the dependency on a wireless network (which is unreliable), the limited computational resources, the reduced capability of storage and energy. All of these constraints, plus the inherent difficulties of the mobility like the decreased user ability to concentrate, the change of connection point, the high exposition regard security.

Among the main properties of the RMAs, the rich functions are the most visible. From the perspective of the user, a rich function is the ability to conveniently perform any computation regardless of the application resource requirements and mobile device's constraints. In that sense, as resource enhancer, the cloud emerges as an option that cooperate with the provision of the rich functions [3].

Therefore, a key aspect to face various of the RMAs' challenges is the cloud [1, 2]. About it, there are research and projects proposing an adaptation of the traditional cloud for the mobile environment. Such adaptation is called *Mobile Cloud Computing (MCC)*, where MCC consider aspects of mobility, the low quality of connection and the resource constraints [14] [15] [16] [17] [18].

Summarizing so far, the cloud is an emerging option to provide an extension of the mobile capabilities. Hence, in the interest of this work is on the functions in the cloud. For example, among such functions we could cite the next: i) great capability of storage, which additionally provides remote-only storage enabling a security factor for sensitive information; ii) great capability of computation, which at the same time could lead to a significance energy saving in case of complex processes.

2.3 The Cloud

Since, the interest of this work is on the functions in the cloud, it is necessary to do a summary of some concepts related to the cloud. In this section, we refer to concepts about cloud models, cloud computing and level of services provided. The two main types of cloud deployment models are [6]:

- Public cloud: The cloud infrastructure is made available to the general public or a large industry group and is owned by an organization selling cloud services (provider).
- Private cloud: The cloud infrastructure is operated solely for an organization. Such infrastructure can belong to same organization or by a third party (provider).

Since the considerable growth of the public cloud, we have focused on it instead of the private one. Closely related to the cloud there is a popular term, cloud computing, which is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction [19]. Examples of providers: Amazon Web Services, Openshift, Microsoft Azure Platform, Google App Engine, Salesforce [20]. Cloud computing is provided by third parties and includes the provision of private and public cloud models. In our case, we consider the provision of public cloud models. We have worked with Openshift and Amazon Web Services. Moreover, there are three levels of services provided by cloud computing [6]:

- *Infrastructure as a Service* o *IaaS*: The capability provided to the consumer is to provision processing, storage, networks, and other fundamental computing resources where the consumer is able to deploy and run arbitrary software, which can include operating systems and applications. The consumer does not manage or control the underlying cloud infrastructure but has control over operating systems, storage and deployed applications.
- *Platform as a Service* o *PaaS*: The capability provided to the consumer is to deploy onto the cloud infrastructure consumer created or acquired applications created using programming languages and tools supported by the provider. The consumer does not manage or control the underlying cloud infrastructure including network, servers, operating systems, or storage, but has control over the deployed applications and possibly application hosting environment configurations.
- *Software as a Service* o *SaaS*: The capability provided to the consumer is to use the provider's applications running on a cloud infrastructure. The applications are accessible from various client devices through a thin client interface such as a web browser (e.g., web-based email). The consumer does not manage or control the underlying cloud infrastructure with the possible exception of limited user-specific application configuration settings.

Among those levels we focus on the platform as a service or PaaS one, provided as a public cloud from the cloud computing service providers. In this way, we consider Openshift and Amazon Web Services as the service providers

2.4 Portability Challenge

In this work, the challenge we address is the effort introduced by the difficulty of portability. We refer to the effort in the development of applications.

The portability is defined in terms of how easy it is to migrate software from one environment to another [7]. Following, we specify issues related to the mobile environment, the cloud and the communication interface between them.

Regarding the mobile side, the popular platforms differ in architectural aspects of hardware and software. Therefore, the portability becomes hard [1] [21]. There are differences at operating systems

levels, programming languages, libraries, methods or functions. Consequently, the developers must write different versions of code for just one application. Hence, the development cost and the time are increased [3] [4] [15].

At platform level in the cloud, each provider has its own programming style, its own libraries, technologies, services and execution environments. The developer must know the details of each platform. Migration between two provider platforms is also hard. Therefore, the portability becomes expensive and complex [6] [7] [15]. This portability challenge is called *vendor lock-in* or *proprietary lock-in* problem. Similarly, at the moment of considering both sides, mobile and cloud, it is necessary to pay attention in the communication interface between them. This interface is made through APIs, where those could change according the platforms[1].

2.5 REST, the Architecture Style

Considering the cloud in the scheme of the MobileApps-FC, we present the communication architecture to be used between these two environments (mobile and cloud) and the application design in the cloud side. For the mentioned issues, REST seems to be the most convenient architecture style.

REST (REpresentational State Tranfer) is a coordinated set of architectural constraints that attempts to minimize latency and network communication, while at the same time maximizing the independence and scalability of component implementations. REST is considered as an abstract model of the Web architecture and used to guide the redesign and definition of the Hypertext Transfer Protocol and Uniform Resource Identifiers (URIs) [22].

For a better understanding of REST, we describe some fundamental concepts according Richardson et al. [23]:

- Resource: A resource is anything that is important enough to be referenced as a thing in itself. Usually, a resource is something that can be stored on a computer and represented as a stream of bits: a document, a row in a database, or the result of running an algorithm. In order to be referenced, a resource must have an URL associated.
- URL: The Uniform Resource Location (URL) is a global address of a resource. If an URL is assigned to a thing, then, that thing becomes a resource. The URL makes it possible the communication between the client and the server.
- Representation: Representation is just some data about the current state of a resource or changes to be applied on a resource. The representation is a useful description, which can have different formats like JSON[2], *XML*[3] and others. The transfer of representations is done in both directions, from the server to the client and from client to the server.
- HTTP methods: They come from the HTTP Protocol[4]. In REST, the interaction is made through the HTTP methods. The most used are GET, PUT, POST and DELETE. These methods operate on the URL.

About the architectural constraints, REST considers the next [22]:

- Client-server architectural style: Separation of concerns is the principle behind the client-server constraints. By separating the user interface concerns from the data storage concerns, we improve the portability of the user interface across multiple platforms and improve scalability by simplifying the server components. Such separation allows the components to evolve independently.
- Stateless communication: Where each request from client to server must contain all of the information necessary to understand the request, and cannot take advantage of any stored context on the server. Session state is therefore kept entirely on the client. This constraint induces the properties:
  - Visibility: In case of monitor systems, it does not have to look beyond a single request datum in order to determine the full nature of the request.
  - Reliability: It eases the task of recovering from partial failures.

[1] Backendless, enlace: `https://goo.gl/FHjIxX`

[2] JavaScript Object Notation, enlace: `http://goo.gl/ErtWK1`

[3] eXtensible Markup Language, enlace: `http://goo.gl/QwUt1r`

[4] https://tools.ietf.org/html/rfc2616

  - Scalability: Not having to store state between requests allows the server component to quickly free resources, and further simplifies implementation because the server doesn't have to manage resource usage across requests.

- Cache: Cache constraints require that the data within a response to a request be implicitly or explicitly labeled as cacheable or noncacheable. If a response is cacheable, then a client cache is given the right to reuse that response data for later, equivalent, requests. The advantage of adding cache constraints is that they have the potential to partially or completely eliminate some interactions, improving efficiency, scalability, and user-perceived performance by reducing the average latency of a series of interactions. The tradeoff, however, is that a cache can decrease reliability if stale data within the cache differs significantly from the data that would have been obtained directly from the server.
- Uniform interface: The central feature that distinguishes the REST architectural style from other network-based styles is its emphasis on a uniform interface between components. By applying the software engineering principle of generality to the component interface, the overall system architecture is simplified and the visibility of interactions is improved. Implementations are decoupled from the services they provide, which encourages independent evolvability. The tradeoff, though, is that a uniform interface degrades efficiency, since information is transferred in a standardized form rather than one which is specific to an application's needs. In order to obtain a uniform interface, multiple architectural constraints are needed to guide the behavior of components. REST is defined by four interface constraints: identification of resources; manipulation of resources through representations; self-descriptive messages; and, hypermedia as the engine of application state.

Other two constraints are the layered system style and the code on demand, though, this last is optional.

REST vs SOAP

In addition, it is important to highlight that REST and SOAP[5] are the most predominant paradigms of web communication [24]. SOAP is a standard protocol which define how two objects in different process can communicate through an interchange of XML data. However, REST take advantage of SOAP. Some of the reason are the next [25]: i) complexity: The serialization or deserialization of the native languages to SOAP messages take a considerable time; ii) redundancy: There is a lot of information in the SOAP messages and its description could be redundant. Such redundant information increases the network communication volume.

Furthermore, a simple SOAP message contains a great portion of XML data. This consumes considerable bandwidth and it can produce an incrementation of the network latency time [26]. In [27], a performance comparison concludes that an implementation based on REST take advantage regard to one based on SOAP.

2.6 Model Driven Approach

The software development approach to follow in this work is the model driven one. The models can be used for different purposes. One of them is the modeling for the development of software artifacts. This approach is known as the Model Driven Software Engineering (MDSE) or directly Model Driven Engineering (MDE).

MDE can be defined as a methodology for applying the advantages of modeling to software engineering activities. In such methodology, the models become in fundamental elements of the software development.

MDE is specified as follows [10]:

- Concepts: The components that build up the methodology, spanning from language artifacts to actors, and so on.
- Notations: The way in which concepts are represented, i.e., the languages used in the methodology.

[5] Simple Object Access Protocol. SOAP specifications: `http://goo.gl/P2JxZu`

- Process and rules:The activities that lead to the production of the final product, the rules for their coordination and control, and the assertions on desired properties (correctness, consistency,etc.) of the products or of the process.
- Tools: Applications that ease the execution of activities or their coordination by covering the production process and supporting the developer in using the notations.

Next, we detail MDE specific paradigms, which are Model Driven Development and Model Driven Architecture.

Model Driven Development (MDD)

MDD is a development paradigm that uses models as the primary artifact of the development process. Usually, in MDD the implementation is (semi)automatically generated from the models. Some of the benefits of the MDD adoption are the next [9]: i) increase of productivity; ii) adaptation to the technological changes; iii) adaptation to the requirements changes; iv) consistency; v) reuse; vi) improvements about the communication with the user; vii) improvements about the communication with the developers; viii) capture of professional experience; ix) the models are long-lasting products; x) possibility to delay technological decisions.

Model Driven Architecture (MDA)

MDA is the particular vision of MDD proposed by the Object Management Group (OMG) and thus relies on the use of OMG standards. Therefore, MDA can be regarded as a subset of MDD, where the modeling and transformation languages are standardized by OMG [10]. As we already mentioned in the introduction, MDA defines different abstraction layers for the design and development of applications: i) Computer Independent Model (CIM), which represents an application independently of computer-based concepts; ii) Platform Independent Model (PIM), which represents an application independently of an implementation platform; iii) Platform Specific Model (PSM), which represents an application for a specific platform; and iv) code, which corresponds to the source code of the application.

Moreover, MDA is a family of standards, from which the Meta Object Facility (MOF) and the Unified Modeling Language (UML) are universally accepted [6]. Following, we present a brief description of some standards:

- Standards related to the transformation and the foundational model management: i) MOF: It provides the fundamental base for MDA. MOF uses metamodels specified in UML for describing modeling languages as interrelated objects. MOF is a subset of UML for building metamodels; ii) XML Metadata Interchange (XMI): It specifies how to interchange models using a specific XML structure; iii) Query View Transform (QVT): Standard based on patterns for the transformation between models.
- UML and Profiles: UML is a family of modeling notations. It is unified by a common metamodel and covers multiple aspects of system and business modeling. A key property of UML is the extension through profiles. These profiles can be adapted to specific requirements, extending and adapting the capabilities of the UML core.

2.6.1 Model Oriented Web Approach (MoWebA)

The model driven approach considered for this work is MoWebA [28]. MoWebA is a navigational role-centric MDD proposal for Web applications development. Furthermore, the approach adopts the MDA standard. In this sense, MoWebA defines methodological aspects (processes, stages, work products, dimensions) and complements these aspects with an entire environment, including modeling and transformation tools, automatic code generation and a layered architecture.

Most approaches, which provide a modeling based on a data-oriented navigation, starts the modeling process from a conceptual model (e.g., data model). Instead, MoWebA considers a function-oriented navigation. Such navigation is directly related to the functions defined through the use cases.

[6] OMG-MOF-UML, link: `http://goo.gl/utxGiQ`

MoWebA is heavily based on the separation of concerns. Consequently, it is proposed a navigation-oriented modeling, an automation of tasks, a standard adoption and the handling of the web environment evolution considering the new technological trends. In this sense, the aim is to achieve a greater interoperability and extensibility of the web systems.

MoWebA defines two types of complementary process: the modeling process and the transformation process, which we will explain in the next subsections. Afterwards, we will explain some aspects of MoWebA which could have a positive impact on the design portability. Finally, we describe the metamodel and the profile of MoWebA to be extended in this work.

Modeling Process of MoWebA

The modeling process includes the CIM, PIM, PSM defined by MDA. In addition, MoWebA includes an extra layer called the Architectural Specific Modeling (ASM), which is between the PIM and the PSM. Such modeling layers are systematized in seven stages (see 2.1). Stages 1 through 6 are oriented to CIM and PIM definitions, based on the dependency relationships between the different models, the level of granularity of the modeling task, and the type of modeling to be done; these stages are done manually. MoWebA adopts the Use Case model for CIM definition, focusing on modeling the functional requirements of the system-to-be. For PIM definition, MoWebA proposes the following models: i) Entity Model; ii) Navigational Model; iii) Behavioral Model; iv) Presentation Model; and v) User Model. Each model is composed of one or more diagrams. Figure 2.2 presents the dependency relationships between the different models.

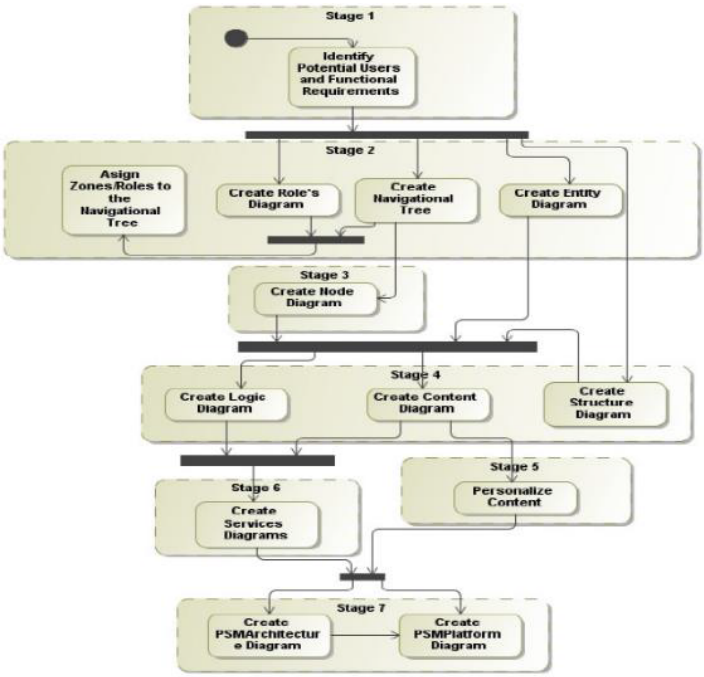


Fig. 2.1: Modeling process of MoWebA

Each stage includes several diagrams defined according dependencies among the models, the granularity level of the task to be done and the model type. Also, the Figure 2.2 presents the different diagrams included in each model.

Stage 7 contemplates the architectural and platform aspects. This stage is done in a semi-automatic way. It proposes an enrichment of existing models in order to consider aspects related to the final architecture of the system (e.g. RIAs, SOAs, REST), specifying the ASM diagram. The next step proposes to add platform specific information (e.g. Ruby on Rails, Python, PHP, Java), specifying the PSM diagrams.

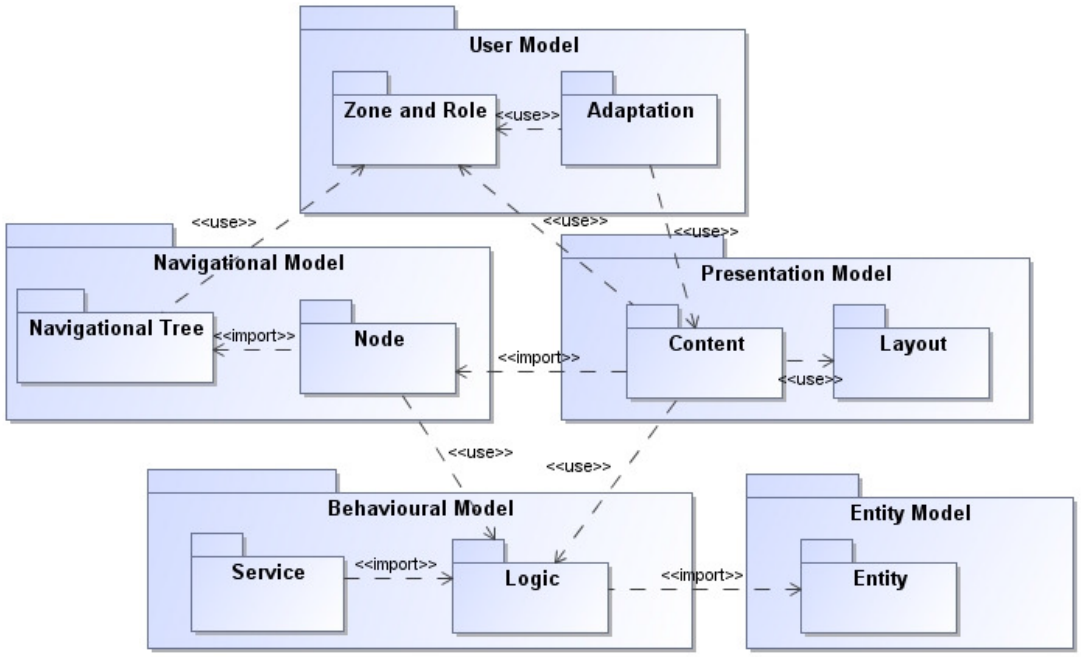


Fig. 2.2: Diagrams in MoWebA

Transformation Process of MoWebA

The transformation process implies steps and activities for transformation specification in order to go through each MoWebA phase (CIM/PIM− >ASM/PSM, ASM/PSM− > Implementation Specific Model (ISM)/ Manual). This process aims to define intermediate specific models before the final implementation (see Figure 2.3).

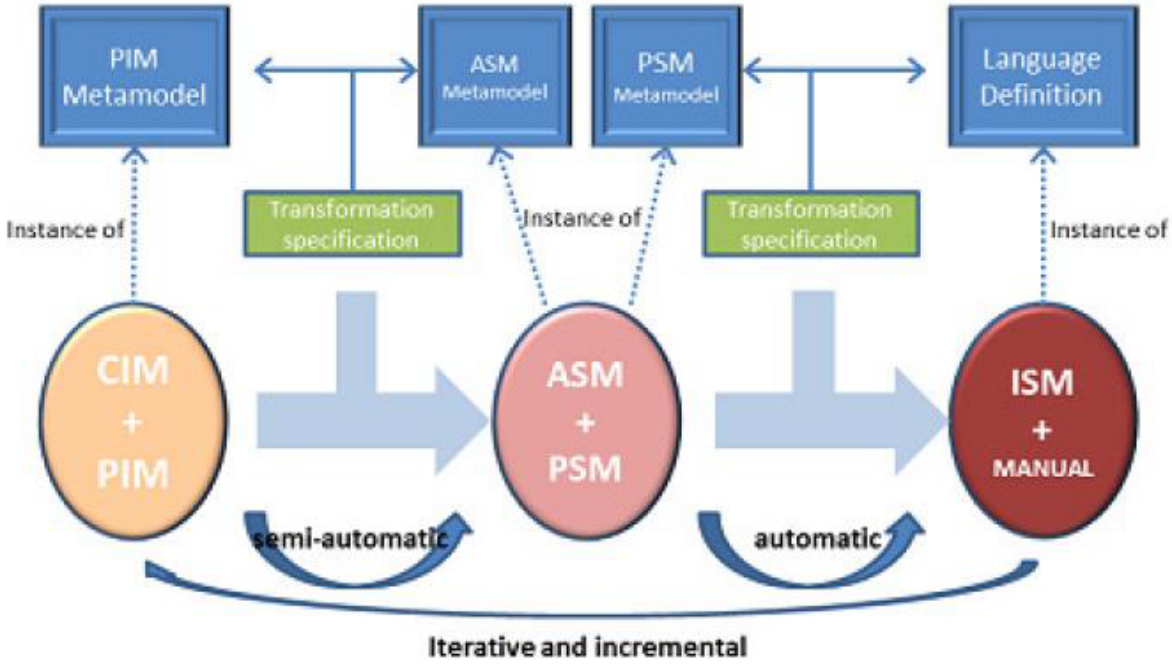


Fig. 2.3: Iterative and incremental transformation process of MoWebA

The transformation process is based on metamodels (PIM− >ASM− >PSM transformation). The PIM − > ASM/PSM phase should be done in a semi-automatic way; since sometimes the information to be added requires human intervention (e.g. in RIAs, the modeller needs to specify

where services will be executed, on the client or on the server). The ASM/PSM− >ISM phase should be done automatically by using open source tools (e.g. Acceleo, AndroMDA). The input models of this phase are the ASM or the PSM the obtained at the previous phase, and the output will be source code.

2.6.2 Positive Impacts of MoWebA for the Design Portability

In MoWebA [28] we have identified certain aspects that could have a positive impact in the portability problem that is found in the MDD/MDA development of MobileApps-FC. Following, we present these aspects arguing about their possible contributions on portability.

- Incorporation of an Architecture Specific Model (ASM) as a new modeling layer. While the portability to different platforms is well resolved by the MDA approach, the constant technological evolution brings also new architectures (e.g., Rich Internet Applications (RIA), mobile architectures). Usually, when MDA development methods are extended to cope with these new architectures, it happens that extensions are included at the PIM level, resulting in an enriched PIM that includes characteristics and constraints that are related to a certain specific architecture [29]. The enriched PIM loses its independence from the architecture and becomes more complex to understand and manage. The consequence is a loss of portability and reusability of models. To resolve this problem, some authors have proposed the introduction of the ASM, for instance, as an intermediate layer between PIM and PSM (e.g., [30, 28]). In this way, one PIM model can result in different ASM models, since one same application can be implemented in different architectures (e.g., a banking application with a web version using RIA and a mobile version based on the mobile architecture). In a similar way, one ASM model can result in different PSM models, since one same architecture can be implemented in different platforms (e.g., the RIA architecture can be implemented in different platforms such as Backbase, Dojo, GWT, jQuery, among others). Therefore, the ASM prevents adding architectural details at the PIM level, improving its portability.
- A clear separation of the presentation layer with regard to the navigation and behavior layers, based on the separation of concerns. The presentation layer is the one that presents more difficulties related to portability [4]. There are differences about screen size and resolution; text style (fonts, size, colors); user interface elements (e.g., buttons, search bars, menu); position and visibility; availability of physical buttons (e.g., a physical back button is available in Android and Windows phone, but not in iOS); types of control user interface elements; types of navigation representation; etc. For example, several of the mentioned differences can be seen in the Android and iOS versions of Twitter[7]. Hence, a clear separation of the presentation layer details regarding details of other layers would prevent additional complications to the presentation. This prevention could help the portability of the presentation to be improved, as well as for the other layers.
- Different MDD proposals deal with the modeling and construction of the navigational perspective according to a data-oriented approach (sometimes also called object-oriented) [31, 32, 33, 34]. This means that the navigation is based on data (e.g., classes) and its structure (e.g., relationships among classes). In practice, the content and structure of the menu of an application are the data and a derivation of its structure as it is stored, respectively [35, 36]. Alternatively, according to a function-oriented approach, navigation is based on functions. In this case, the content of the menu are functions (functional requirements) and the structure of the menu depends mainly on user requirements [37, 36, 38]. For instance, an application could not suffer modifications on its functional requirements but its data model could change (e.g., splitting or joining different classes because of a normalization or de-normalization). As a consequence, in the case of the data-oriented approach, such changes could affect the navigation. For example, a function requirement that *registers* users in an application can be implemented by means of a *user* class. This class has different attributes, including, for instance, home address and business address. If we normalize *user* regarding address, we obtain two classes, *user* and *address*. Considering a data-oriented navigation, before the normalization, the registration function involves a single screen. After the normalization, we have to navigate to a new screen to fulfill the addresses. Instead, considering a function oriented approach, independently of the data organization we still have just the same function *register*, and therefore navigation does not require to

[7] Twitter iOS vs Twitter Android, link: https://goo.gl/Ro3qbA

Table 2.1: Possible contributions of aspects with regard to portability

Aspect	Possible Contribution
ASM	Portability of the PIM with regard to different architectures
Separation of presentation from navigation and behavior	Portability of the presentation layer
Function oriented navigation	Portability of the modeling of navigation

be modified. Furthermore, regarding platforms in mobile environments, the navigation design is different for each of them[8]. Each platform considers particular navigation patterns and design restrictions [39]. Therefore, it is necessary to adapt the navigation according to the constraints of each platform. Building a navigation structure that makes sense to the user (i.e., it is fast and easy to get to the content) is a common suggestion coming from design guidelines of mobile platforms. They have their own standard navigation elements and patterns. For example, the *tab bar* for iOS, the *side drawer* for Android, the *swipe views* in Windows phone. These kind of standards create habits in users. Therefore, to get an intuitive navigation, those standards have to be considered in the design process of each platform. Considering the registration example, in the data-oriented approach changes in the data model have to be applied to navigation. Since navigation elements and patterns differ according to platforms, it would be necessary to review the design made in each platform taking into account the mentioned changes. Therefore, such analysis and reviews for each platform imply more effort and could make even worse the effects of the portability problem. Nevertheless, considering a function-oriented navigation, the mentioned changes do not affect the navigation, whence there is no additional work to be performed. In this sense, the function-oriented navigation would help to attenuate the effects of the difficulties of portability. Therefore, there is an indirect positive relationship between the navigational function-oriented approach and the portability of applications.

Table 2.1 summarizes the possible contributions of the previous aspects with regard to portability. Despite the intuition on the positive impacts that these aspects could have, more studies and research are necessary to get a clear assess of them. In that way, this work is intended to offer a step forward.

In summary, such aspects could help to improve the portability design. Such improvement means saving of effort in the design as we saw above. This saving, from an overall perspective, could relieve the effects (extra effort) introduced by the platform portability difficulties, which is the problem in the development of MobileApps-FC, we have focused on.

Following we describe the MoWebA metamodel and the profile extended in order to model the network communication aspect. We have focused on such aspect to do the implementation.

2.6.3 Metamodel and Profile of MoWebA to be Used

First of all we present what a metamodel and a profile are. A metamodel is a set of related concepts, which defines in this case a certain diagram. In order to be able to model, a profile is needed. The profile is a notation built from the metamodel. In other words, the profile elements represents concretely the concepts of the metamodel. Such elements can be stereotypes or tag values. In this sense, such notation enables the modeling.

MoWebA proposes metamodels and profiles to the definition of the PIM models' diagrams, previously mentioned. Such diagrams do not include architectural details, since, the PIM is independent of architecture. Consequently, the diagrams of the PIM models can be extended for different architectures (e.g., RIAs, REST, SOAs). In this sense, taking advantage of such property, we can extend the metamodels and profiles of the MoWebA's PIM models in order to get the ASM for the MobileApps-FC. Particularly, our efforts focus on the network communication aspect. Consequently, next, we describe the metamodel and the profile to be extended for the modeling of the network communication functions of the MobileApps-FC.

The network communication is a logic process. Therefore, from all the PIM models' diagrams of MoWebA, we selected the Logic one. The metamodel is shown in Figure 2.4.

[8] Android: https://goo.gl/kcCuxc; iOS: https://goo.gl/Jox14J; Windows phone: https://goo.gl/Ql3iOH

The logic diagram enables the definition of logic processes. Each process is called *TProcess* and it is executed inside an application. For instance, for an application with registered users, a *TProcess* can be the *user management*. Such logic processes, at the same time, are composed by *Services*, where each one of them is a procedure for doing a specific task. Taking the user management example, a service can be *updating user profile*. Furthermore, in the same diagram, it is possible to define *ValueObjects* for grouping attributes. These attributes could come from one or more entities of the data model. The *ValueObjects* build an abstraction layer on the access to the data model. In Figure 2.4 are shown the metamodel and the profile of the logic diagram.

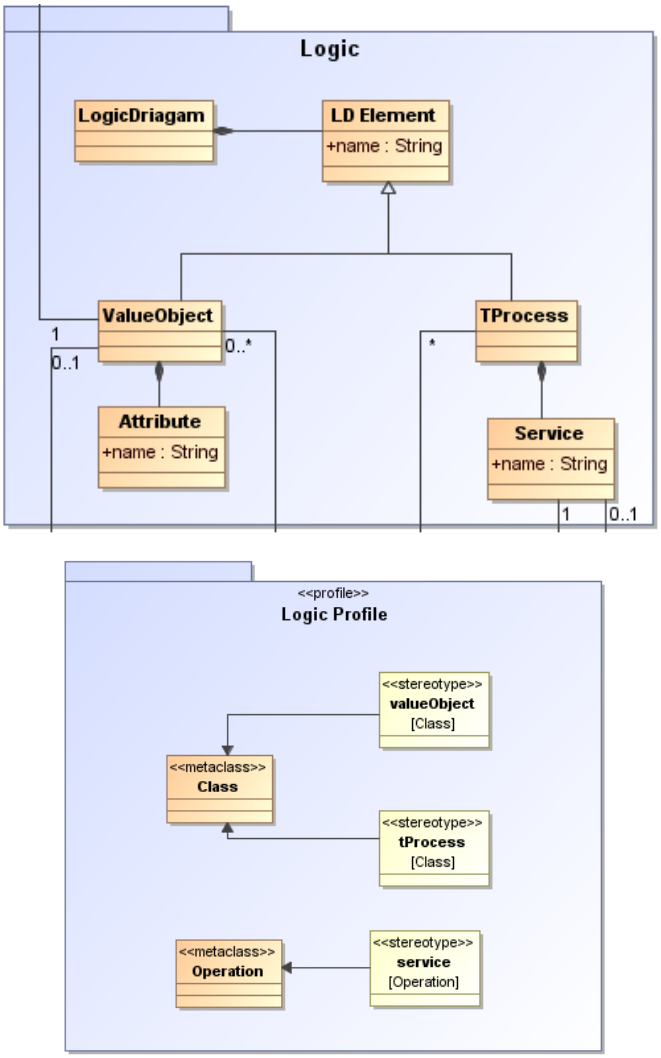


Fig. 2.4: Metamodel and profile of MoWebA's logic diagram

Thus, considering all the diagrams of the MoWebA's PIM, the logic one seems to be a suitable option for designing the network communication. Therefore, the extensions and additional definitions for obtaining the corresponding ASM are applied on such diagram.

2.7 Summary of the Chapter

In summary, this work is related to the development of the MobileApps-FC, where the functions in the cloud enables the enrichment of the mobile applications through the provision of additional resources. Regarding the cloud, we focus on functions running in the public cloud. This public cloud is available thanks to the cloud computing providers, providing PaaS. In this way, we consider to Openshift and Amazon Web Services as the service providers at the platform level. The problem, which we are focus on, is the extra effort introduced by the difficulty of portability be-

tween platforms. This problem occurs in the mobile side as well as in the cloud, and consequently, in the communication between them. Specifically, we focus on one aspect of the MobileApps-FC, which is the network communication. We based the design of such communication on the REST architecture, which have the next properties: i) Cloud design based on resources; ii) Uniform communication interface; iii) Simplicity and efficiency about the client-server interaction. About the approach to be followed, we proposed the adoption of MoWebA, a MDD approach. We highlight some of the MoWebA's aspects, which could have a positive impact on the portability design. Finally, we present the metamodel and the profile of the logic diagram, which is the MoWebA's diagram to be extended.

Chapter 3
SMS of the State of the Art

In this chapter, we present a Systematic Mapping Study (SMS) about the use of the Model Driven Development (MDD) for the development of Mobile Applications with Functions in the Cloud (MobileApps-FC). We performed a SMS since this study is characterized for being more general than a Systematic Literature Review (SLR). Giving a global vision about a subject of interest, with empirical evidences or not, is the main goal of a SMS [40]. Moreover, we focused our study to identify and analyze proposals that address portability challenges. Therefore, each selected proposal has been analyzed regarding aspects that could improve the portability in the context of the development of MobileApps-FC based on MDD. Based on MoWebA [28], we have identified three aspects that could have a positive impact in the portability problem that is found in the MDD development of MobileApps-FC. These aspects are: i) incorporation of an Architecture Specific Model (ASM) as a new modeling layer. The aim of the ASM is to keep the portability of the PIM regarding the different architectures (e.g., RIA, SOA, mobile architecture). In other words, the ASM enables the reuse of the Platform Independent Model (PIM) to the different architectures. Such reuse could mean a saving of effort in the design phase; ii) clear separation of the presentation layer with regard to the navigation and behavior layers. The presentation layer is the one that presents more difficulties related to portability. Hence, a clear separation of the presentation layer details regarding details of other layers would prevent additional complications to the presentation. This prevention could help the portability of the presentation to be improved, as well as for the other layers; and iii) definition of the navigational structure according to a function-oriented approach. The function-oriented approach prevents the modification of the navigation design caused by implementation changes (e.g., a data-oriented navigation has to be modified if there are changes in the data model structure). Such prevention means a saving of effort, where this effort would have required to modify the navigation design. For more details about the analysis of such three aspects, see Chapter 2, Section 2.6.2.

The sections of this chapter are organized as follows. Section 3.1 presents related works. Then, Section 3.2, Section 3.3, and Section 3.4 present the planning, execution and results of the SMS, respectively. In Section 3.5, we analyze the results. In Section 3.6, we specify the limitations of our study. Finally, Section 3.7 presents a conclusion.

3.1 Related Work

As a starting point of this work, we first conducted a search for SMS and SLR related to the development of MobileApps-FC under the MDD approach. Despite not having identified the existence of such work, we found studies that are partially related to our interest. Next, we briefly describe those studies.

Heitkötter et al. [8] present an analysis that focuses on the development of portable mobile applications. They compare cross-platform solutions (mobile web applications: PhoneGap[1] and Titanium Mobile[2]) against mobile native applications, according to two categories of criteria. On one side, criteria related to infrastructure: license and costs, supported platforms, access to platform-specific features, long-term feasibility, look and feel, application speed and distribution. On the other side, criteria related to development: development environment, GUI design, ease of development, maintainability, scalability, opportunities for further development, speed and cost

[1] Link: https://goo.gl/e2gG

[2] Link: https://goo.gl/794vzn

of development. From their perspective, they conclude that the cross-platform approaches present interesting properties like the use of standard and popular programming languages (development side) and the access to a considerable set of platform-specific features (specially from PhoneGap and Titanium Mobile in the infrastructure side). Therefore, they could be considered as a substitute of the native development approach.

The solutions analyzed in this work are not model-driven. Furthermore, they are focused on the implementation stage. These are differences with regard to our main interest, which relates to model-driven approaches that deal with portability at the design stage. Nevertheless, they propose alternative perspectives (as the web[3] and hybrid applications[4]) to reduce the impact of the portability problem in the mobile environment.

Ribeiro and da Silva [41] present a set of technologies that focuses on the development of cross-platform mobile applications. The technologies are Rhodes[5], PhoneGap, Dragon RAD, Titanium, mobl[6] and mdsl[7].

The study presents a comparative analysis of the technologies cited above, about the approaches, languages, tools, type of applications generated and device access. Among other conclusions, the study highlights the low consideration of the model driven approach for the development of mobile applications, just mobl and mdsl take into account MDD, and suggest a greater consideration of it, since MDD may represent a good solution not only to develop cross-platform apps, but also to ease the development and to captivate a larger number of developers for the specific domain (in this case, the mobile domain).

Di Martino et al. [42] describe a set of projects which consider MDD to give support to portability and interoperability in the cloud. The set of projects includes MODAClouds[8], ARTIST[9], REMICS[10] and PaaSage[11].

Furthermore, this work also presents an interesting summary of cloud patterns. Some of these patterns are independent of the provider and consequently, platform independent. The platform independent patterns were extracted from CloudPatterns.org[12], which is a community dedicated to the documentation of a catalog of patterns that includes a set of design solutions for the modularization of the relevant technological solutions for the configuration of clouds [42]. This information is relevant to our area of research since it considers platform independent design aspects that could be used to enrich and to improve the portability of cloud mobile applications developed under the MDD process. In contrast to our work, Di Martino et al. refer to a level of modeling that is not precisely related to the design of an application, but rather refers to configurations and functions of the cloud environment.

Umuhoza and Brambilla [43] performed a survey on MDD approaches for the development of mobile applications. The selected studies were classified according to: i) the covered phases of the development process (requirements analysis, design, implementation, testing); ii) the covered aspects of mobile applications, like application data, user interaction, user interface, and business logic; iii) applied model driven techniques, including type of modeling language, model transformations, code generation or model interpretation; iv) the type of mobile application developed, like native, hybrid, or mobile web; v) the supported mobile platforms, including Android, iOS, or Windows Phone. As results, the survey shows a preference of code generation over model interpretation and of native applications over cross-platform ones. Regarding the relationship with our work, this study was focused exclusively on the development of mobile applications (leaving out implementations in the cloud) and it does not consider the analysis of portability issues.

Despite the contributions of the works that have been presented in this section, it is worth to mention that they have not followed the guidelines of a SLR or SMS. Therefore, since we have not identified a SLR or SMS that focuses on the development of MobileApps-FC under the MDD approach, we were motivated to perform the SMS that is described in the next sections.

---

[3] Web mobile applications rely the portability in browsers like Safari or Chrome.

[4] Hybrid applications are web applications (or web pages) in native browsers, such as UIWebView in iOS and WebView in Android (not Safari or Chrome). Hybrid applications are developed using HTML, CSS and Javascript, and then wrapped in a native application using platforms like Cordova.

[5] Rhodes, link: https://goo.gl/Ivrq2f

[6] mobl, link: https://goo.gl/7xh46X

[7] Canappi mdsl, link: https://goo.gl/FWezFE

[8] MODAClouds project web-site, link: https://goo.gl/lLBnPF

[9] ARTIST project web-site, link: https://goo.gl/6ecBGO

[10] REMICS project web-site, link: https://goo.gl/U7fDxp

[11] PaaSage project web-site, link: https://goo.gl/PPMYgz

[12] Link to the site: https://goo.gl/OsXNFZ

3.2 Planning of the Systematic Mapping Study

We conducted this SMS following the guidelines proposed by Genero et al. [40] and Kitchenham and Charters [44].

The study consisted of the following steps: i) identification of the need for a systematic mapping; ii) formulation of research questions; iii) definition of the search protocol; iv) derivation of the search string and selection criteria; v) searching for primary studies; vi) identification of the data needed to answer the research questions; vii) summary and synthesis of study results; viii) report-writing.

In the Introduction and Section 3.1 we described the need for this SMS. In this section, we present information related to steps ii), iii) and iv). Steps v) and vi) are presented in Section 3.3 and Section 3.4, and the step vii) in Section 3.5.

3.2.1 Research Questions

As previously mentioned, this study's goal is to compile and analyze works that follow an MDD approach and that, at the same time, consider the improvement of the portability of MobileApps-FC *(see research question RQ1)*. The aim is to analyze if these works provide, or not, a layer related to the ASM, separation of presentation with regard to behavior and navigation, and function oriented navigation, since we consider these aspects could provide benefits related to the portability problem *(see RQ2, RQ3 and RQ4)*. Furthermore, we consider the identification of other contributions or limitations, specially with regard to portability, to learn about complementary approaches that could improve this aspect *(see RQ5)*. In addition, we analyze the evaluation presented by the selected articles, in order to have an assessment of the practical benefits of the MDD approach regarding the portability problem, and to get a perspective of its possible evolution *(see RQ6)*.

Given these goals and interests, we have defined the next research questions:

- **RQ1**: Which are the model driven proposals for the development of MobileApps-FC that address the portability problem?
  In the identified proposals:
- **RQ2**: Are there evidences of an independent layer for ASM?
- **RQ3**: Are there evidences of a clear separation of the presentation layer regarding the navigation and behavior layers?
- **RQ4**: Are there evidences about the adoption of the function oriented navigation?
- **RQ5**: Are there other contributions or limitations related to portability?
- **RQ6**: Are there validations? If so, are there positive results regarding the portability problem?

3.2.2 Search Protocol

The search protocol that we followed consists of the next steps:

1. Derive terms from research questions to build an initial search string.
   a. Use similar and related terms along with acronyms to obtain the final search string.
2. Query information sources with the defined search string. Focus the search on titles, abstracts and keywords. Adapt the search string according to formats an restrictions of searching tools. Information sources to be considered are digital libraries and indexing services, including journals and conferences.
3. Apply a selection procedure on the query results. This procedure consists of applying some selection criteria in three stages:
   a. First stage: Apply selection criteria taking into account the title of the works identified with the query of information sources (this is, works identified in step 2).
   b. Second stage: Apply selection criteria taking into account abstracts and keywords of the works selected in step 3(a).
   c. Third stage: Apply selection criteria taking into account the full text of the works selected in step 3(b). Discard duplicate articles and those which refer to the same work or proposal.

We have not considered any restriction about publication years at the moment of searching. The search was performed in October 2016. The selection procedure was carried out by one of the researchers. In cases of doubts, the other three researchers were consulted in order to achieve agreement.

3.2.3 Search String

The first research question (RQ1) has been used to derive terms. Then, we used these terms to build the initial search string. This was the case because from that question (RQ1) it is possible to identify the relevant proposals for this study. Then, the initial search string was enriched with acronyms, and similar and related terms. The resulting search string is:

((“model driven” OR “model-driven” OR MDD OR MDE OR MDA OR “model based” OR “model-based”) AND (mobile OR smartphone OR tablet OR Android OR “windows phone” OR iOS) AND (cloud OR “remote server” OR backend))

The search string was adapted according requirements and restrictions of each searching tool. These adaptations are shown in Table 3.1.

Table 3.1: Search string adapted to each considered information source

Information source	Search string
IEEE	((“model driven” OR “model-driven” OR MDD OR MDE OR MDA OR “model based” OR “model-based”) AND (mobile OR smartphone OR tablet OR Android OR “windows phone” OR iOS) AND (cloud OR “remote server” OR backend))
ScienceDirect	(((“model driven” OR “model-driven” OR MDD OR MDE OR MDA OR “model based” OR “model-based”) AND (mobile OR smartphone OR tablet OR Android OR “windows phone” OR iOS) AND (cloud OR “remote server” OR backend))) AND LIMIT-TO(topics, ”model,cloud”)
ACM	((“model driven” OR “model-driven” OR MDD OR MDE OR MDA OR “model based” OR “model-based”) AND (mobile OR smartphone OR tablet OR Android OR “windows phone” OR iOS) AND (cloud OR “remote server” OR backend))
Scopus	((“model driven” OR “model-driven” OR MDD OR MDE OR MDA OR “model based” OR “model-based”) AND (mobile OR smartphone OR tablet OR Android OR “windows phone” OR iOS) AND (cloud OR “remote server” OR backend))
Google Scholar	allintitle: mobile cloud ”model driven” OR MDD OR MDE OR MDA OR ”model based” OR smartphone OR tablet OR remote OR server OR backend OR Android OR ”windows phone” OR iOS
DBLP	(model.driven|MDD|MDE|MDA) (mobile|smartphone|ios|android)

3.2.4 Selection Criteria

The remaining research questions (RQ2, RQ3, RQ4, RQ5, RQ6) have been used to guide the definition of inclusion and exclusion criteria.

3.2.4.1 Inclusion Criteria

This SMS includes studies and proposals that:

1. Consider some of the following properties related to portability: inclusion of an additional abstraction level for modeling, like the ASM; clear separation of the presentation layer regarding the navigation and behavior layers; function oriented navigation; or other mechanisms related to manage or improve portability.
2. Present some of the following elements: meta-models, profiles, transformation rules, frameworks, tools.

3.2.4.2 Exclusion criteria

This SMS excludes studies and proposals that:

1. Are not related to software engineering.
2. Do not follow a model driven approach.
3. Are not related to the development of mobile applications.
4. Focus on mobile web applications instead of native or hybrid applications.
5. Do not consider the modeling and generation of functions in the cloud.
6. Do not consider the portability problem.

3.2.5 Information Sources

In principle, we have chosen information sources in accordance with a list proposed by Kitchenham and Charters [44], including digital libraries and indexing services. Since some libraries of the mentioned list, like Citeseer library and Inspec, did not give results that match our research area, we focused our search on the information sources listed below:

- Digital libraries: IEEE, ACM and Science Direct.
- Indexing services: Scopus, Google Scholar and DBLP.

Nevertheless, the indexing services have identified studies belonging to different digital libraries, even those that we did not consider directly. It is worth mentioning that this work has included Scopus as an additional source of information. Scopus was not used in the previous version of this SMS [11].

3.3 Execution of the Systematic Mapping Study

Table 3.2 presents the number of articles which have been obtained when the search protocol was executed. Based on the protocol previously presented, column *Search* of Table 3.2 presents the number of articles that have been found after applying the search string in each digital source (step 2 of the protocol). Similarly, columns *Criteria on title*, *Criteria on abstract and keywords*, and *Criteria on full text* show the number of articles that have been selected after the execution of the steps in which inclusion and exclusion criteria were applied (steps 3(a), 3(b) and 3(c) of the protocol, respectively). As we can see, finally, six articles were selected to be analyzed in this SMS. Table 3.3 shows the proposals presented in the six selected articles with their corresponding references.

It is worth mentioning that, at the stage of applying criteria on titles, we included all studies whose titles refers at least to the MDD approach and the mobile environment, since we obtained articles from other research areas, which did not refer to the development of mobile applications. At this stage, we considered all selection criteria, except for exclusion criteria number 5 and 6 since they relate to more particular issues (cloud and portability), which are not necessarily appearing in titles. At the stage of applying selection criteria on abstracts and keywords, we focused on selecting articles that include portability as their main problem, and excluding those which did not fill that criteria. At this stage, we considered all exclusion criteria except for number 5, which refers to cloud. Finally, at the stage of performing the analysis on the full text of the articles, all selection

criteria were considered. Some more details about the application of selection criteria are presented in Section 3.6.

Table 3.2: Number of articles found in different steps of the search protocol

Digital sources		Search	Criteria on title	Criteria on abstract and keywords	Criteria on full text
Digital libraries	IEEE	82	1	1	6
	Science Direct	180	2	2	
	ACM	20	2	2	
Indexing services	Scopus	291	6	4	
	Google Scholar	169	4	3	
	DBLP	58	39	19	
Total of results		800	54	31	6

Table 3.3: Summary of results by research questions

	Works					
	WebRatio	MD^2	SIMON	MobiCloud	Steiner et al.	Ruokonen et al.
Ref.	[4]	[45]	[46]	[47]	[48]	[49]
Desc.	Framework with graphical modelling language based on IFML-mobile, UML, and BPMN	Textual modeling language based on the MVC schema	Framework with XML as a textual modeling language and middlewares for the adaptation to different plarforms	Textual modeling language based on the MVC schema	Textual modeling language based on the MVC schema	Graphical modeling for UI generation based on web services
RQ1	Yes	Yes	Yes	Yes	Yes	Yes
RQ2	Conceptually no, in practice yes	No	Partially	No	No	No
RQ3	No	Yes	No	Regarding navigation no, regarding behavior yes	There are not enough details about modeling	Regarding, navigation no, regarding behavior yes
RQ4	It is possible to get	It is possible to get	No	No	It is possible to get	No
RQ5	Pros: Open source generation and REST. Cons: Different tools and languages for modeling and generation	Pros: Server portability based on the JAVA Server and REST. Cons: Basic approach for modeling and generation in the cloud	Pros: Reuse of common functions and details of clouds. REST. Cons: Focused in a specific type of application.	Pros: REST. Cons: Basic modeling and generation.	Cons: Basic modeling and generation.	Cons: It considered an old mobile environment.
RQ6	Yes	Yes	Yes	Yes	Yes, but there is a lack of details	No

3.4 Results

Table 3.3 presents a summary of the results of our SMS, considering each research question. Results associated to each research question are described next.

- **RQ1**: Which are the model driven proposals for the development of MobileApps-FC that address the portability problem?
  As previously mentioned, in this SMS six studies were identified that fulfill our selection criteria (see Tables 3.2 and 3.3). These studies constitute the answer to our first research question. In other words, the model driven proposals for the development of MobileApps-FC that address the portability problem are WebRatio [4], MD^2 [45], SIMON [46] ,MobiCloud [47], and the proposals of Steiner et al. [48], and Ruokonen et al. [49]. All these proposals were analyzed to answer the remaining research questions.
  Regarding WebRatio and MD^2, it is worth mentioning that their corresponding papers do not explicitly mention functions running in the cloud. Nevertheless, they are both able to generate applications with functionalities that can be run in the cloud. In the case of WebRatio, the documentation of their tools mention that it is possible to generate code that will be run in the cloud[13]. In the case of MD^2, it generates a Java EE 6 application as back-end and this kind of application can be easily deployed in the cloud[14]. Therefore, both approaches were considered as MobileApps-FC and were included in this work.
  Among the selected proposals, we could consider WebRatio as the most complete work in our study area. WebRatio includes modeling capabilities that allow the design of demanding applications for real use cases. Furthermore, it has earned a place in industry. MD^2, based on the Model-View-Controller (MVC) schema, could be considered as the second work regarding modeling and generation scope. MD^2 has been used in experiments in industry. Unlike the other selected works, SIMON, a framework focused on data oriented applications, does not generate code from the models. Instead, it includes runtime modules, which are middlewares that are adapted according to the specifications coming from models. MobiCloud and Steiner et al. are also based on the MVC schema, but they consider more basic use cases for modeling and generating applications. The work of Ruokonen et al. refers to the old Nokia platform. They consider business processes for modeling, and from these processes they generate different user interfaces according to the specific model of the target device. According the model, a device could have particular specifications of display size, positions of user interface elements, functions.
  Although we have identified more studies which refer to MDD applications for the mobile environment, we have only found these six studies that also include the cloud.
- **RQ2**: Are there evidences of an independent layer for ASM?
  Starting from WebRatio, it defines a way to extent a pre-existent PIM (the IFML[15] language) in order to obtain models oriented to the mobile environment. For the cloud side, other languages like UML and BPMN are used. In practice and from our perspective, WebRatio has an ASM, although they do not formally recognize this ASM as a new independent layer in their development process. Also in SIMON, it is possible to identify some aspects of architecture (specially details of the cloud, like operations on it), which are encapsulated in a runtime module. This module is separated from the PIM and also is still independent of a platform. It is considered as a middleware that abstracts details of the cloud architecture. In this sense, unlike WebRatio, SIMON distinguishes an additional level of abstraction besides the PIM and the PSM. Therefore, we could say that SIMON considers an ASM, although, in the case of MobileApps-FC, we should say that the considered ASM is partial, because it considers only the cloud side, and there is no middleware (independent from platform) for the mobile architecture.
  In Ruokonen et al., from our perspective, they have a PIM, composed by the business processes and the user interface model, and a PSM where they define the specific characteristics of the target devices. Therefore, they do not consider an additional level of modeling as the ASM.
  In the remaining works, they only talk about domain specific languages, specifically defined for modeling and generating MobileApps-FC. Hence, they consider a PIM, and from it, they generate platform specific implementations.
- **RQ3**: Are there evidences of a clear separation of the presentation layer regarding the navigation and behavior layers?

[13] WebRatio Mobile, https://goo.gl/rTDSrU

[14] Push Java EE 6 Application to the cloud, https://goo.gl/DpgNOK

[15] IFML, link: http://goo.gl/mWdPpo

We identified this separation in MD^2, where navigation and behavior are modeled in the controller module while presentation (e.g., graphic elements, styles) is isolated in the view module (schema MVC). In MobiCloud and in Ruokonen et al. there is no mention about navigation modeling, hence the separation of the details of these layers is not clear. Regarding the behavior layer we could assume, as a first judgment, that they consider a clear separation since MobiCloud models it inside the controller and Ruokonen et al. through business processes which are isolated from user interface modeling. Also, in Steiner et al. there is not a clear specification of the presentation layer design, neither of the behavior nor navigation ones, whereby the separation is not clear. Nevertheless, in WebRatio a mix of details of the presentation layer regarding the navigation and behavior modeling can be identified, since graphical elements, navigation, events, and service callings are specified in the same diagram. This mixture could be reducing the portability of the mentioned layers. In SIMON, there are three models: the user interface configuration model (where presentation is defined), the data model, and the security policy model. Nevertheless, there is no specification about the navigation modeling, and furthermore there are not enough details about the behavior modeling. According to the article, in principle, we could say that behavior is divided between the user interface configuration and security policy models. Hence, we cannot ensure that there is a clear separation between the layers.

- **RQ4**: Are there evidences about the adoption of the function oriented navigation?
  In WebRatio, MD^2 and Steiner et al. we identified particular models to design navigation. Whereby, it is possible to obtain a function oriented navigation, but it depends on the designers decision because the mentioned proposals do not impose a specific type of navigation. In WebRatio, transitions among different elements of the user interfaces can be defined by means of the IFML-mobile language, as well as transitions among different screens that conform the application structure. Similarly, in MD^2 navigation modeling is carried out in the controller module (schema MVC) and the workflow concept makes it possible to model different types of transitions. In Steiner et al. navigation can be modeled but it is limited to a subset of patterns that are available on both mobile platforms considered for generation, iOS and Android. In this case, we can get function oriented navigation but with limitations. In SIMON, MobiCloud and Ruokonen et al. there is no consideration of navigation modeling, whence there is no guarantee to get a function oriented design.
- **RQ5**: Are there other contributions or limitations related to portability?
  Since we have only considered proposals based on MDD for the development of MobileApps-FC, the six selected works aim to abstract the developer from specific implementation details of the mobile and cloud environments, through the PIM, allowing the generation of the final applications for different platforms from the same model. This implies possible savings in effort, time and cost, considering the portability problem.
  Besides other aspects, which we have verified through the previous three research questions, we did an analysis in order to identify other strategies which complement the use of MDD for the improvement of the portability of applications.
  WebRatio considers the generation of mobile code based on Apache Cordova[16], also known as an hybrid approach or native wrapper, and the generation of standard Java code for the cloud, relying its portability on the Java server. In general, we could say that WebRatio chooses for a complement between MDD and the generation of open source code, which is considered as an approach that gives more flexibility for addressing the vendor lock-in problem[17]. All of this can be considered as a suitable combination to face the difficulties originated by the portability problem [4, 45]. Furthermore, WebRatio provides the automatic generation of a smart method for synchronization, which is portable thanks to the PIM and transformation rules.
  At the same time, this property could be relevant for saving resources of mobile devices at the moment of network communication. Nevertheless, for the design of MobileApps-FC, WebRatio uses different modeling languages, with different modeling tools. This could mean, in some sense, a limitation regarding the abstraction about the differences between the mobile and cloud environments, and consequently, derives in the need of learning to work with different tools (we could have a harder learning curve).
  Regarding SIMON, the modularization of architectural aspects of the cloud and the reuse of them through different providers, can be mentioned as contributions related to portability. Another contribution is related to the level of abstraction, since there is a unique modeling language, XML. About limitations, SIMON focuses only on data oriented applications.

[16] Apache Cordova, link: https://goo.gl/QocB2x

[17] More information in: https://goo.gl/mOCr8P

MD^2, MobiCloud and Steiner et al. use textual domain specific languages, based on the MVC design pattern. Then, from the same language, they generate implementations for both, the mobile and cloud environments. They manage only one language and one tool for modeling and generation. This could be considered as an advantage regarding the learning curve, comparing with WebRatio. These three works consider the generation of native code for the mobile side. Generally, this kind of code is the most preferred because it presents better possibilities for a suitable use of resources of mobile devices. This means that applications can be obtained with better performance and presentation. As well as WebRatio in the cloud side, MD^2 considers the generation of Java applications, relying the portability of the implementation on the Java server.

MobiCloud generates specific implementations for two proprietary cloud providers, Amazon Elastic Compute Cloud (Amazon EC2) and Google App Engine (GAE). Regarding disadvantages, the modeling and generation approach in MobiCloud is restricted to basic CRUD operations and to the generation of communication between the mobile and cloud environments. Likewise, Steiner et al. consider the generation for GAE, but in this case it was the only one taken into account.

From all proposals, we identified REST as the most adopted communication architecture. This uniformity in communication interfaces becomes a positive property for the improvement of the portability of the communication between the mobile and the cloud, and at the same time for the different platforms belonging to these two environments respectively [22, 23].

About Ruokonen et al., they consider the user interface generation for devices of different physical characteristics (e.g., according to model, screen size). They have identified common and repetitive patterns in the context of business process models. About disadvantages, they generate non-native code for user interfaces. Besides, according to their work context (Nokia Research Center, 2008), we could say that they focused in a specific type of environment, so they did not take into account current mobile platforms.

In general, as a limitation we can mention that most of the studies evidence early research stages. In some cases, the modeling and generation in the cloud side is minimal.

- **RQ6**: Are there validations? If so, are there positive results regarding the portability problem?
  Five of the six selected works included validations. All of them presented positive results.
  The most applied evaluation method consisted in comparing the number of lines of textual models against the number of lines of code generated from those models. This kind of evaluation was performed by proposals that rely on textual modeling languages: MobiCloud, MD^2 and Steiner et al. These studies relate the number of lines of code with the effort, time and cost that would take the manual implementation, following the traditional development approach. In all cases, differences in at least one magnitude order were found regarding the number of lines. Specifically, the number of lines of textual models was smaller than the number of lines of code of final implementations. From that, it can be derived that the effort, time and cost to obtain a MobileApps-FC for more than one platform, using an MDD approach, are lower than in the case of the traditional development.
  Regarding SIMON, the considered validation method was the comparison of lines of code, again. Unlike the evaluations described above, this comparison was made between the traditional development and the use of SIMON. An application for gathering data was developed. The considered target platforms were Google App Engine (GAE) and Microsoft Azure, as cloud providers, and Android and Windows Phone, as mobile platforms. First, the application was developed for GAE and Android. In this case, SIMON already had the native mobile runtime engine and the cloud adapter (which are middlewares that encapsulate the specific details of a particular platform. For each mobile platform, it is necessary to build its corresponding mobile runtime engine. The same case for the cloud side, for each provider/platform, a cloud adapter must be developed). Nevertheless, when they developed the version for Azure and Windows Phone, they had to developed an specific native mobile runtime engine and cloud adapter. The results of the evaluation were the following:

  – GAE and Android: For the traditional development the lines of code were 6224 and for SIMON were 280. A 95% of lines of code decreasing.
  – Azure and Windows phone: For the traditional development the lines of code was 6324 and for SIMON were 4014. A 37% of lines of code decreasing.

  Regarding WebRatio, they compared their MDD proposal against the traditional development. Specially, they considered the mobile side. Three teams, with two members each, developed an application according to the following description:

- Team 1: They worked in the server side, implementing REST services.
- Team 2: They worked in the client side, adopting a manual development approach (traditional approach).
- Team 3: They also worked in the client side, but adopting the MDD approach proposed by WebRatio.

Teams 2 and 3 developed the front-end following the Apache Cordova cross-platform approach. All teams were tested during 2 weeks.
The following results were obtained:

- 9 person-days were employed for the server side development, using the MDD approach through the WebRatio Framework.
- 21 person-days were employed for the manual client side development: i) graphical style: 2 person-days; ii) structure of the application and user interaction: 12 person-days; iii) client-server interaction and push notifications: 7 person-days.
- 11 person-days were employed for the MDD development of the client side: i) graphical style: 1 person-day; ii) structure of the application and user interaction: 7 person-days; iii) client-server interaction and push notifications: 3 person-days.

From the data described, the following benefits were derived: 48% of effort saving, 21% of cost reduction, time reduction from 21 person-days to 11 person-days. In all the cases, the numbers favor the MDD approach proposed by Brambilla et al. [4] (WebRatio).
Ruokonen et al. do not present a validation experience regarding their implementation. Instead, they pose some issues related to design patterns, from the mobile environment, which can be automated through MDD.
Finally, from all these validations and considering the portability problem, we can say that their results are positives and consequently, encouraging.

3.5 Analysis of Results

Table 3.4 presents a summary of the results regarding the aspects that we considered for the improvement of portability: inclusion of an ASM, separation of presentation with regard to navigation and behavior, and function oriented navigation.

From the perspective of these aspects, it is possible to model a function oriented navigation in three proposals: WebRatio, MD^2 and Steiner et al. Hence, this aspect is the most feasible one to be adopted by most of the selected works. From the perspective of the works, WebRatio and MD^2 are closer to meet the studied aspects than the others. Both of them are able to adopt the function oriented navigation. MD^2 considers a clear isolation of the presentation layer and, in practice, WebRatio has an ASM. In general, the separation of details between the presentation and behavior layers, and the ASM are the least considered aspects.

From this situation, we can propose studies about either the enrichment of the selected proposals, specially with the two least considered aspects, according to their current status, or new proposals that include these aspects in a native way. First, we further analyze these two alternatives and suggest possibles ways to follow. Then, we propose other possible ways and complementary issues about future researches.

Table 3.4: Summary of Works vs Aspects

	Works					
Aspects	WebRatio	MD^2	SIMON	MobiCloud	Steiner et al.	Ruokonen et al.
ASM	Conceptually no, in practice yes	No	Partially	No	No	No
Clear separation of the presentation layer regarding navigation and behavior layers	No	Yes	No	No	No	No
Function Oriented Navigation	It is possible to get	It is possible to get	No	No	It is possible to get	No

3.5.1 Enrichment of the Selected Proposals

3.5.1.1 WebRatio

WebRatio presents a navigation model independent of the data modeling, so we deduce that it is possible to obtain a function oriented navigation. About the clear separation of the presentation layer regarding the navigation and behavior layers, we identified a mixture of details because in the same diagram they model user interface elements, transitions among these elements, events and calling of services. Therefore, it would be interesting an study that compare or analyze in a deeper way the actual modeling of WebRatio against a modeling where these details are specified in different diagrams, and see if this separation contributes to the portability of the design, or at least, present other benefits. About the ASM, WebRatio do not consider this aspect in a conceptual or formal way. Nevertheless, they did extensions from an existing PIM, so in practice and from our perspective, they have ASM. It would be interesting if some experiments are made to build model to model transformation rules for different architectures, for example for the RIA architecture and for the mobile architecture, in order to verify the portability of their PIM.

3.5.1.2 MD^2

MD^2 includes navigation modeling, so it is possible to obtain function oriented navigation. It also includes the separation of the details of the presentation layer regarding the navigation and behavior layers, through a clear encapsulation of these details in the view and controller modules respectively (considering the MVC schema). MD^2 does not consider an ASM, hence it would be interesting to perform experiments about the obtainment of the PIM, which could be reused to generate the MobileApps-FC architecture and other architectures (e.g., RIA), trough its textual model language.

3.5.1.3 SIMON

Compared with the other proposals, SIMON has a particular approach. It considers XML as a textual modeling language, to specify as a Blueprint Engine (BE) three design aspects: data model, security policy and user interface configuration. Besides the blueprint, SIMON includes two additional modules, the Data Runtime Engine (DRE) and the Security Runtime Engine (SRE), which according to the blueprint performs, respectively, data-oriented operations in specific clouds (e.g., interoperation data, inserts, updates, etc.) and security validations between the user and the application (e.g., authentication, validation of privileges, etc.). The BE, DRE, and SRE are the core components and they are independent of platform. Attached to DRE, there is an adapter for each cloud provider. The adapter is a middleware, between the DRE and the cloud provider, that encapsulates specific details of a cloud platform. On the mobile side, it is considered a Native Mobile Runtime Engine (NMRE). It is also a middleware, between the core components and the mobile platform, composed of prefabricated components that are responsible for presenting applications functions to the user by using necessary services provided by BE, DRE and SRE. As we saw, SIMON does not consider the three aspects studied here. SIMON does not present navigation modeling, so it would be interesting that it includes the mentioned modeling and suggests a function oriented approach. Also, some experiments could be perform to clearly separate the modeling of the presentation and the behavior layers, in order to see if this separation contributes to a greater reuse of the existing NMRE and cloud adapters, to build new implementations according to other specific platforms. Partially, SIMON considers an ASM that is encapsulated in the DRE and SRE modules. But, these modules are focused on the cloud side. In this sense, they could consider to build this kind of modules on the mobile side.

3.5.1.4 MobiCloud, Steiner et al. and Ruokonen et al.

From these studies, only Steiner et al. consider one of the aspects that we have studied. We have not identified that MobiCloud and Ruokonen et al. consider at least one of the mentioned aspects. Steiner et al. include a navigation modeling that gives the possibility to obtain a function oriented navigation. But, this modeling is limited to a subset of patterns (of navigation) that are present in

two specific mobile platforms that they consider for the generation of code. These studies evidence an early stage about MDD for the development of MobileApps-FC. The cases of study considered by these three last works implies basics CRUD operations on a simplified data model, or some application dimensions are not considered for the modeling (like the navigation modeling), and the user interface is restricted to basic elements (like labels, buttons and entry boxes) [47, 48, 49].

3.5.1.5 Other Considerations

It should be noted that despite the mentioned limitations, all these works present initial evidences that are positive for the adoption of MDD for the development of MobileApps-FC. Therefore, they should be considered as reference points for the continuation of the research. More complex cases of study could be considered, in order to identify other patterns, repetitive processes, and functions which can be incorporated in the MDD approach. Even MD^2, would need to consider more complex cases of study to verify the applicability and feasibility of their proposal in industry cases.

Further and besides the three aspects studied here, these studies could be enriched by a unification of their research or at least they could benefit from learning from their respective research. Specially, those which consider a textual modeling language: MD^2, MobiCloud and Steiner et al. Also these works are based on the MVC schema. At first glance, they consider a quite similar process and approach.

Once there are more evidences, it would be possible to get more information about the effects of the three studied aspects regarding the portability and, also, about the applicability of MDD for the development of MobileApps-FC.

3.5.2 New Proposal Including the Desirable Aspects

A new proposal that includes in a native way the three main aspects considered here could be proposed. This proposal could be a methodology considering a complete modeling and generation process, complemented by its development environment, specifically its modeling and transformation tools, all of them based on standards like UML and MDA. Considering a general view, we could divide the development process in three phases:

- Problem modeling.
- Solution modeling.
- Implementation or source code generation.

About the problem modeling, we highlight here the PIM specification. This PIM could be based on five models, offering a strong separation of concerns that is in line with some of the three aspects studied in this work:

- A navigation model, including particular diagrams to model the transitions among the different parts of an application, enabling the possibility to obtain a function-oriented navigation, independent of the data model.
- A domain model for the data layer modeling, where it is possible to model the structure and representation of data.
- A presentation model, where all the user interface design and its details are encapsulated.
- A logic model for behavior design, isolated from the presentation layer details.
- A user model for defining roles and assign specific functions to them.

About the solution modeling, we highlight the inclusion of the ASM as an additional level of models, and as a previous stage of the PSM, or directly, as a previous stage of the specific implementation, the application generation. Therefore, from the PIM and through model to model transformation rules, this new proposal should generate, with manual adjustments if it is necessary, an ASM that incorporates the concepts, patterns and functions coming from the emerging architectures.

About the implementation phase, either from the ASM or PSM, it should be possible the generation of code and other artifacts (e.g., documentation) that constitute the generation of a working application.

This new proposal could also take profit of the proposals studied in this work. The new proposal could be enriched by:

- The use of standard modeling languages. For instance, WebRatio [4] uses UML graphic models while SIMON [46] uses XML for textual models.
- The support for modeling and implementation of settings related to security policies in the cloud, that can be derived from SIMON [46].
- The adoption of the MVC schema, that can be derived from MobiCloud [47], MD^2 [45] and Steiner et al. [48].
- The combination of MDD and the open source approach, that can be derived from WebRatio [4].
- The use of REST as a uniform and portable communication interface, that can be derived from WebRatio [4], MD2 [45], SIMON [46], MobiCloud [47].

3.5.3 Complementary Issues

First, as a complement of modeling and generation in the cloud, there are particular configurations and functions that are peculiar of the clouds. In Di Martino et al.[42], those configurations and functions are presented as patterns. In their analysis, they summarize patterns independently of providers and platforms. Some of these patterns are about sharing, scaling and elasticity patterns, data management and storage, network security, identity and access management. As we can see, they are not related to the design of applications specifically, but they are closely related and they contribute to alleviate the vendor lock-in problem [42]. This could be an interesting option to enrich the modeling of applications since, it should contribute to improve the portability of the models.

Second, considering the clear separation of the presentation layer regarding the navigation and behavior layers, the principle of Separation of Concerns can be identified, which facilitates the building, understanding and expressiveness of model languages. In this sense, as future work, it would be interesting to perform an study about the separation of concerns in the specification of mobile applications, since these languages that apply the mentioned principle could be used in the MDD process and, at the same time, it could enrich that process.

Third, it would be interesting to confront and to compare different proposals, that combine MDD with other methodologies or approaches, regarding the qualities of portability and efficiency. This last one, is also relevant, specially for the mobile side, because the restrictions of resources like the battery. For instance, these possible studies could compare the combination of MDD with open source/independent platform generation (e.g., Apache Cordova, hybrid applications) against MDD with a proprietary and/or native generation. The open approach is considered as a more flexible way to improve the portability, while the native or proprietary approach is considered better to get more efficiency. It would be interesting to know which combination presents the better trade-off between the mentioned qualities. In the same line, keeping the analysis regarding portability and efficiency, it would be interesting an study that compares the implementation of the REST interface communication, which is uniform and portable, against a native interface communication, which is better about efficiency.

Fourth, about the development environment. MD^2 presents a unified modeling language and tool to model and generate MobileApps-FC. It could be considered as an advantage comparing MD^2 against WebRatio, since WebRatio presents different languages and tools to model and generate the mentioned applications. It would be interesting to analyze or perform experiments to see if there are significant differences between these both alternatives or which of them present the better properties. For such a study, the learning curve and the tools and language power could be considered as parameters for analysis.

As summary, considering the portability problem of the MobileApps-FC from the analyzed works, we have found that there is a low consideration of the aspects of interest that could improve the portability. Nevertheless, we have identified initial positive evidences supporting MDD for the development of the mentioned applications, regarding the portability issue. These positive evidences encourage future research to get more information and even more evidences in order to enable more general conclusions about the benefits of MDD on the portability problem, at least for the kind of applications that we consider here. Future works could adopt our aspects of interest. This adoption could be done either proposing new approaches or enriching the selected ones. Some other complementary research lines were proposed considering some principles and qualities of the software engineering and which are somewhat related to our main subject.

3.6 Threats to Validity

This section describes some threats that must be improved in future replications of this study. We consulted experienced researchers to validate the research design and its understanding. Their feedback and the trials contributed to reduce threats.

3.6.1 Threat of Missing Literature

Since in [44] it is mentioned that in some cases a simple search string could be as effective as a complex one, we tried to use a search string that is not very complex. Our search string included frequently used terms related to our areas of interest, according to our experience and to what we saw in the literature. Nevertheless, in future studies, in order to try to get more results from the sources, our search string could be enriched with additional terms, synonyms, abbreviations (e.g., "platform independent model", "platform agnostic model", "mobile app", "mobile device", services, server). In this study, we could not use the same string for all search engines because some of the tools didn't give us any results. Therefore, we slightly modified our search string according to the search facilities of the different information sources. The string suffered changes specially in DBLP (as it is shown in Table 3.2). This search tool is not as flexible as the other ones. Despite of such limitations, we opted to keep DBLP because, in general, it gives interesting results of our area of study. Another threat of missing literature is related to the fact that we have defined many exclusion criteria that could have rid off studies about mobile applications that do not consider the cloud but could have been interesting to analyze.

3.6.2 Threat of Selection Bias

As we have explained previously, during the study selection procedure, we gradually applied the selection criteria according to the different stages. In the first stage, we considered all the selection criteria, except the exclusion criteria number 5 and 6 for being more particular issues. These criteria refer to cloud and portability. Since we cannot obtain enough information from title, we assumed that, at least, the articles in which we were interested, should mention the MDD approach and the mobile environment. In the stage of applying the selection criteria on abstracts and keywords, we focused on selecting the articles which include the portability as the main problem and exclude those which do not filled that criteria. In this stage, we considered all the selection criteria except the number 5, which refers to the cloud. We did things in that way because we were searching for works that focus on the portability problem. We assumed that in the abstracts and keywords should be a mention of the problem. Finally, in the stage of doing the analysis on the full text of the articles, all the selection criteria were considered. Additionally, we mention that just one researcher did the search and selection of articles. Hence, we did not do a parallel search and selection which could have helped the validation of such processes.

3.6.3 Threat of Inaccuracy of Data Extraction

About the data extraction and summary, these stages were not planned in advance. Nevertheless, they were performed in a structured way, gathering all the necessary information to answer the research questions. Finally, we have not made an evaluation about the quality of the selected works. In Genero et al. [40], they explained that the inclusion of this point is not essential, since, in a SMS both theoretical studies as well as empirical ones can be included, and therefore, it can be difficult to define a common evaluation mechanism for all the included works.

3.7 Summary of the Chapter

This chapter presents a SMS of MDD proposals for the development of MobileApps-FC. The process has resulted in the identification of a small number of proposals that deal with the subject of interest. This suggest that the area should progress further. At the same time, the analyzed proposals agree in pointing some possible positive aspects about the use of MDD regarding the portability issues in the context of the development of MobileApps-FC. These evidences are still early and they need to be consolidated through further studies. In this sense, the analysis of this SMS have identified possible research lines which include the proposal of new methodologies or improving existing ones to incorporate the desirable analyzed aspects. Moreover, the discipline needs to increment the empirical evidences in order to generalize the consideration about the usefulness of MDD regarding the portability issues for the development of MobileApps-FC.

Chapter 4
Proposed Solution

In this chapter, we present our proposed solution to face the effects of the portability difficulty for the development of Mobile Applications with Functions in the Cloud (MobileApps-FC). We have based our proposal on the Systematic Mapping Study (SMS) presented in Chapter 3. In the mentioned chapter, we have done an analysis about future studies (see Section 3.5). In such analysis, one of the alternatives for future works was the proposal of a new approach. In this sense, we propose the adoption of MoWebA as a new approach for the development of the MobileApps-FC. Following, we describe briefly the content of this chapter. First, we list the aspects of MoWebA on which its adoption is based. Subsequently, we specify the scope of the implementation of this work, which is the network communication between the mobile applications with their functions in the cloud. Similarly, we present the extensions on MoWebA's PIM in order to obtain an ASM for the MobileApps-FC. In addition, we describe the transformation rules, the code generated and the code related to user interfaces. Afterwards, we explain the process of modeling and generation phases of our approach. Finally, we show an example for the design and the generation process of the implemented models and rules, respectively.

4.1 Adoption of MoWebA for the Design and the Generation of MobileApps-FC

In this section, we propose the adoption of MoWebA for designing and generating MobileApps-FC. We explain MoWebA's aspects and additional concerns which support our proposal. First, we cite the three aspects which could have positive impact for the portability design. Second, we present some common aspects identified from the studies selected in the SMS. Third, we describe additional concerns coming from the SMS. Finally, we described our proposal based on what we learned from the state of the art.

Unlike all the proposals identified from the state of the art, MoWebA includes natively the three aspects on which our SMS is based. As we described in Chapter 2, such aspects could help to save effort in the design phase. From an overall perspective of the Model Driven Development (MDD) process, such saving could help to alleviate the extra effort introduced by the challenge of the platform portability. Therefore, such aspects could have positive impact in the design portability, and consequently, they could help alleviate the effects of the platform portability challenge. Following, we cite these three aspects considered by MoWebA:

- Architecture Specific Model (ASM) as a new modeling lawyer. The ASM is a layer between the PIM and the Platform Specific Model (PSM). The aim of the ASM is to keep the portability of the PIM regarding the different architectures (e.g., RIA, SOA, mobile architecture). In other words, the ASM enables the reuse of the PIM to the different architectures. Such reuse could mean a saving of effort in the design phase.
- Clear separation of the presentation layer with regard to the navigation and behavior layers. The presentation layer is the one that presents more difficulties related to portability. Hence, a clear separation of the presentation layer details regarding details of other layers would prevent additional complications to the presentation. This prevention could help the portability of the presentation to be improved, as well as for the other layers.
- Function-oriented navigation approach. The function-oriented approach prevents the modification of the navigation design caused by implementation changes (e.g., a data-oriented navigation

has to be modified if there are changes in the data model structure). Such prevention means a saving of effort, where this effort would have required to modify the navigation design.

For more details about the analysis of such three aspects, see Chapter 2, Section 2.6.2.

Moreover, there are some aspects which were adopted by most of the studies of our SMS. We call to them common aspects. The common aspects identified were:

- A MVC schema, natively adopted by MoWebA.
- REST as a uniform and portable communication interface, adopted as an extension.

In this sense, the SMS, through a common adoption, supports the inclusion of such aspects in our proposal. In addition, among the selected studies there are other interesting aspects, like:

- The use of an standard modeling language. For example UML[1], a standard language for modeling.
- The generation of native mobile application. As we already explained in the introduction of this book, since, the better capabilities for using the mobile resources, the native mobile applications are preferred over the hybrid and web ones[2].
- The combination of MDD and the open source approach. The open source approach, specifically for the cloud side presents more flexibility to address the portability of applications among the cloud service provider platforms (as mentioned in Section 3.5, answer to RQ5).

In spite of all the characteristics mentioned so far, we consider some additional concerns from the SMS. For these concerns, our proposal presents alternatives of solutions:

- There is a low consideration of graphical modeling. Only two of the six studies consider a graphical syntax. In this sense, MoWebA introduces a new graphical modeling language for the MobileApps-FC.
- Most of the selected studies of the SMS consider JAVA for the implementation in the cloud. Any of them consider Node.js, which is an alternative widely used today. In our case, we propose to Node.js as an alternative for the implementation of cloud applications.

In summary, for the modeling and the code generated we present new alternatives according the evidence of the state of the art.

Following we describe our proposal based on the aspects we learned from the state of the art. The adoption of MoWeba as a new approach for the development of MobileApps-FC. MoWebA is based on the standard UML (a graphic modeling language) and considers natively the three aspects studied in our SMS. Such aspects could have a positive impact in the portability of the design of such applications. In this way, MoWebA for the development of the MobileApps-FC can be considered for continuing the study of the mentioned aspects regarding portability. In this case, we extend MoWebA for a unified modeling and generation of the network communication functions of the MobileApps-FC. We based the design of the communication on the REST architecture. From the modeling, we consider the generation of native mobile applications and open source cloud applications. In this sense, the native code for mobile platforms is preferred over other alternatives (e.g, hybrid, web applications) due to their better properties. Similarly, an open source approach for the development of cloud applications helps to improve the portability of those applications[3].

Since, we are dealing with mobile applications we call to our proposal MoWebA Mobile, which is an extended version of MoWebA.

4.2 MoWebA Mobile for Modeling and Generating the Network Communication

In general, we propose MoWebA Mobile as a solution for the development of the MobileApps-FC. Nevertheless, due to the scope of a work to be submitted for an engineer degree, we focus our implementation on one aspect of such applications, the network communication of the MobileApps-FC. Such aspect integrates both sides, mobile and cloud. At the same time, it integrates all the platforms of each side. In other words, working with the communication of the MobileApps-FC, implies working with several platforms with different programming languages, libraries and other

[1] UML, link: `http://www.uml.org/`

[2] Native vs hybrid application, link: `https://goo.gl/JH1u7K`

[3] Open source approach - Portability, link: https://goo.gl/mOCr8P

differences already mentioned in Chapter 2. In this sense, the modeling and the generation of the communication help to understand the relevance of the abstraction presented by a MDD approach.

Therefore, we focus on the network communication itself. Basically, our goal is to achieve the exchange of data between the mobile and the cloud sides. We do not include aspects like the security, synchronization methods and others related issues. Following, we explain the concrete functions we consider in our implementation.

Starting from the official documentations of Android[4] and iOS[5] (currently, the most popular platforms[6]) we focused on four main types of network communication functions we identified. Those communication functions set the scope of this work for the design and the generation implementation. Next, we describe the mentioned types of such functions:

- **Light-data:** In this communication functions, the data exchange does not include files (like images, documents, video or audio). In this sense, the data to exchange is light. For example, integers, floats, strings, list of strings, etc.
- **Load-image:** Download and load of images in memory. This function allows to download an image from the server (in the cloud) and to load it on the screen of the mobile device, always, keeping the image in memory.
- **Download-files:** Download of files in background. It allows to get files from server (in the cloud). Generally in this case, the files have considerable size. After the download the files are stored into the mobile device.
- **Upload-files:** Upload of files in background. It allows to send files to the server (in the cloud). The files are stored into the server.

The mentioned functions include implementations in both sides, the mobile and the cloud, respectively. Such implementation are based on the REST architecture.

Since, we propose the adoption of MoWebA for the modeling and generation of the network communication, we have worked on one diagram of the MoWebA PIM's models, which is the logic diagram. We extended and added elements to the metamodel and profiles of the mentioned diagram, based on the REST architecture and the four types of the network communication functions mentioned above. From such extensions, we got the Architecture Specific Model (ASM), which extends the logic diagram. A we saw in Chapter 2, Section 2.6.3, the ASM is basically the PIM enriched with concepts and elements of an specific architecture. In this case, we consider the architecture of the MobileApps-FC for the network communication. In this sense, the ASM enables the modeling of the network communication functions.

4.3 ASM Definition for the Network Communication

The definition process of the ASM for extending MoWebA for different architectures consists in the following steps [29]: i) define the metamodel using MOF[7]; ii) define the corresponding UML[8] Profile; iii) define transformation rules for elements that can be obtained in an automatic way; iv) apply transformation rules in order to obtain the first version of the ASM from the PIM; v) make necessary manual adjustments to complete the ASM model; vi) generate the target code applying transformation rules; vii) include manual adjustments, if necessary.

The scope of this work has focused, on one side, on the steps i) and ii) for defining the ASM metamodel and profile and, on the other side, on the step iii) and vi) for defining model to text transformation rules and to apply them to get the target code from the ASM. It was not necessary the step vii). We have left for future works the automatic transformation from the PIM to the ASM, which includes the steps iv) and v).

In this work, we have defined the metamodel and profile for obtaining the network communication ASM. Such metamodel and profile extend the logic diagram of MoWebA. The logic diagram of MoWebA enables the definition of logic processes. In this sense, we consider the network communication as a logic process. The logic diagram contain *TProcesses*, which are the logic processes defined. Such processes include *Services*, which are procedures doing a specific task. Also, the logic diagram has *ValueObjects*, which group attributes of entities and enable the access to the entities'

[4] Android, `https://goo.gl/LwMLXn`

[5] iOS, `https://goo.gl/3FlZqW`

[6] Popularity of Android and iOS, `https://goo.gl/ZAu8Ho`

[7] MOF, link: `http://www.omg.org/mof/`

[8] UML, link: `http://www.uml.org/`

data. For more details about the metamodel and the profile of the logic diagram, see Chapter 2, Section 2.6.3.

Following we describe the extensions we made on the logic diagram in order to get the ASM.

The extensions for obtaining the ASM are based on the REST architecture and the four types of network communication functions we have identified through the official documentation of the most popular mobile platforms, iOS and Android. Following, we describe the elements of the ASM we included. Such elements belong to the metamodel showed in Figure 4.1.

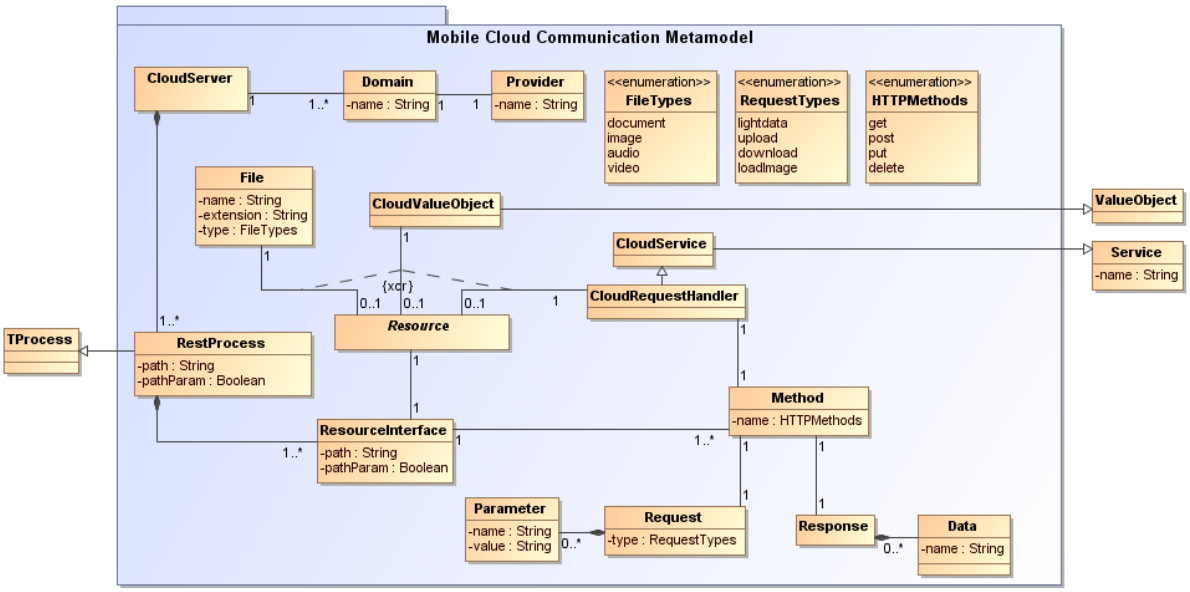


Fig. 4.1: Metamodel for the network communication ASM

Firstly, it is defined a *CloudServer*, which is accessed through a *Domain*. The *Domain* is provided by a service *Provider*. The provider setting enables to distinguish the configurations for each provider of cloud service. Then, each *CloudServer* contains a set of logic processes, where each one of them is called *RestProcess*. Such processes modularize the network communication design. Subsequently, each *RestProcess* includes a set of resource interfaces, where each *ResourceInterface* is associated with a *Resource*, one at a time. A *Resource* can be a MoWebA's *ValueObject* called *CloudValueObject* because it is residing in the cloud, a *File* stored or to be store in the cloud or a MoWebA's *Service* called *CloudRequestHandler*, which is executed in the cloud.

Each *ResourceInterface* is associated to a set of methods, where each *Method* defines the operation to be performed on a *Resource*. At the same time, each *Method* is associated to the object *Request* and, in some cases, to the object *Response*. For one side, each *Request* can be associated to a set of parameters, which contains each *Parameter* of the request. For the other side, each *Response* is associated to a set of data expected to receive from a *light-data Request*, operated by the *Method get*. If a *CloudValueObject* is a *Resource* and the *Method* is *get*, then, there is no need to specify the data types in the *Response*. In case of a *CloudRequestHandler* be itself a *Resource*, then, the data type has to be specified.

Following, we will explain the attributes of each element described so far. The *RestProcess* has a relative *Path*, which makes the process accessible, and a boolean flag which establish if the *Path* is used as additional data in each *Request*. For instance, the *Path* could be a user identification. The same case is for the attributes of the *ResourceInterface*. About *File*, its attributes specify the *name*, *extension* and the file *type*. The *Name* of *Method* defines the type of HTTP method (*get, post, put, delete*). The four HTTP methods considered are the most popular. The *Request type* (*lightdata, download, upload, loadImage*) defines the call to be done. The *name* and the *value* of the *Parameter* describe the different parameters of the *Request*. Finally, the *name* of *Data* specify the data to be receive from the *Request lightdata* and the *Method get*. Such *name* specifies an attribute of the *CloudValue Object* associated to the respective resource interface.

The described elements of the ASM are represented in the profile shown in Figure 4.2. As we saw in Chapter 2, Section 2.6.3, the profile contains the definition of concrete elements, which enable the modeling.

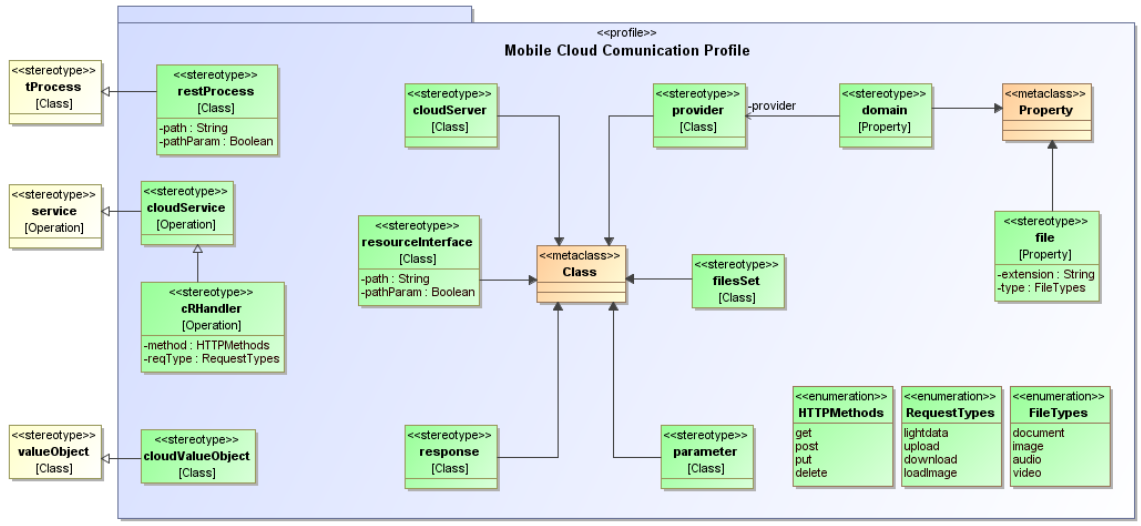


Fig. 4.2: Profile for the communication network ASM

4.4 Transformation Rules

We have defined model to text (M2T) transformation rules in order to generate the target code from the ASM. We used Acceleo as the tool of development, see a screen-shot of the the tool in Figure 4.3. Also, we used the same tool to run the rules on the imported model in order to generate the target code. The first step to achieve the generation of the code is to import the

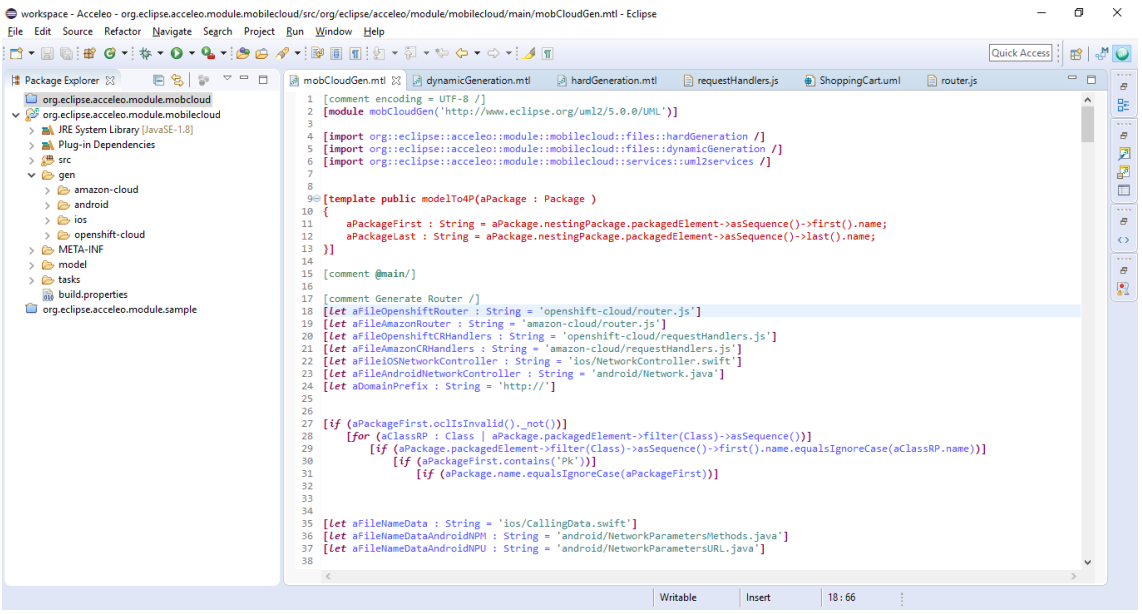


Fig. 4.3: Acceleo for the building and execution of transformation rules

model, designed in MagicDraw, in EMF UML2 (v2.x) XMI format. Then, it is necessary to verify the run settings. As we see in Figure 4.4, we must set our project's name, the main class of our transformation rules definition, the input (imported) model and the target directory. This last will contain the generated code. Afterwards, we can run the rules in order to obtain the target code.

In order to built the transformation rules, we have followed an approach based on templates. We have used the Model Transformation Language (MTL) for defining the templates. Similarly, we have used OCL for doing queries to the model. Furthermore, we have built a service in Java to extend the functions of the MTL.

We have built the transformation rules based on the model stereotypes and tag values of the classes, properties and operations (such stereotypes are the elements defined in the profile of the

ASM, see 4.2). In this sense, such rules perform a mapping between the model elements defined and the target code to be generated. Basically, the generation for both side, mobile and cloud, depends on each combination of a *cloudServer*, a *restProcess*, *resourceInterface* and *cRHandler*. About the cloud, according to such combination there are created the routing to the URL (*cloudServer*'s domain + *restProcess*'s path + *resourceInterface*'s path) and the respective request handler function for handling the requests on such URL. Depending on the *reqType* value, the request handler function is implemented for light-data, load-image/download files or upload files. Also, the parameters and responses are generated if they were specified. Similarly, for the mobile side, from the mentioned combination there are created the clients to send and receive the data from the server in the cloud.

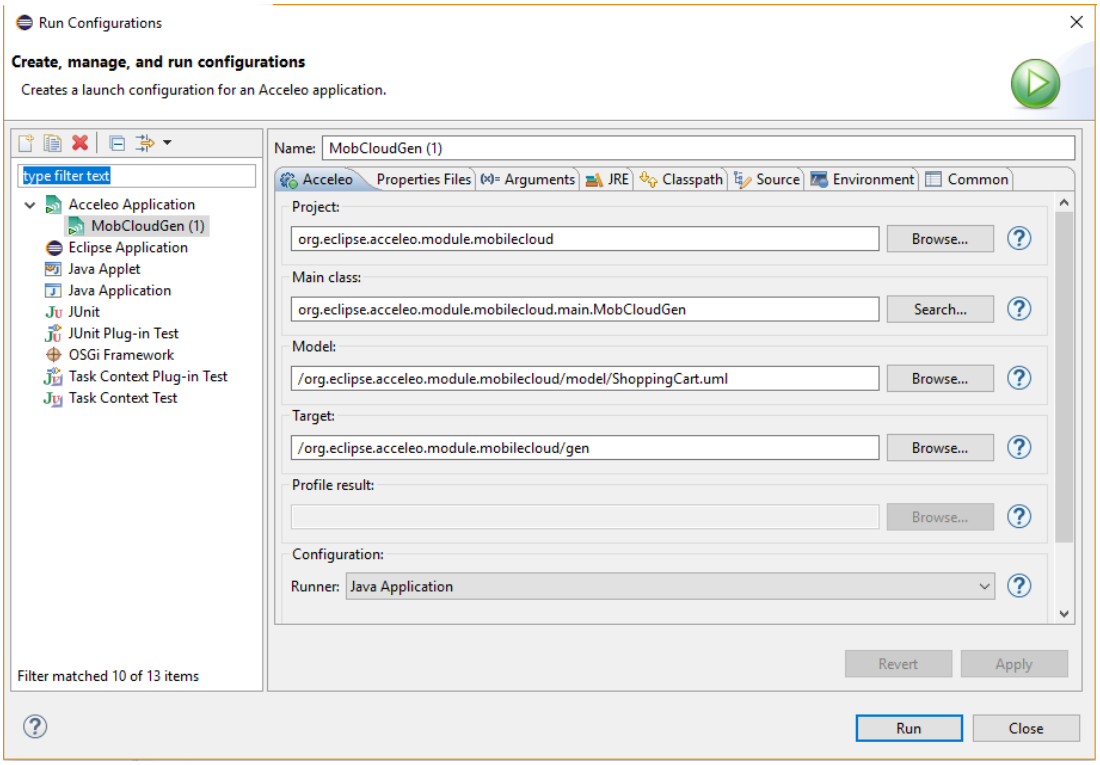


Fig. 4.4: Acceleo run configuration

4.4.1 Code Generated from the Transformation Rules

The transformation rules map the model with the target code. In this section we describe such target code.

From the ASM through the transformation rules, we generate, on one side, native mobile code written in Java[9] for Android, in Swift[10] for iOS, and on the other side, we generate open source code written in Javascript[11] with Node.js for Openshift and Amazon. About Android, we have used the Volley[12] library for light-data and load-image functions, the *DownloadManager*[13] system service for download in background and HttpURLConnection[14] library for upload in background. We generate four Java files. First, Network.java, in which are defined the functions by type of communication (light-data, load-image, download and upload files). Second, NetworkParametersURL.java, which contains all the parameters defined. VolleyCallback.java, which is an interface

[9] Java, link: `https://goo.gl/hGBggw`
[10] Swift, link: `https://developer.apple.com/swift/`
[11] Javascript, link: `https://www.javascript.com/`
[12] Volley, link: `https://goo.gl/5yRYhM`
[13] Download Manager, link: `https://goo.gl/eP4WfQ`
[14] HttpURLConnection, link: `https://goo.gl/wY72DF`

for the asynchronous calls of the Volley library. MySingleton.java, which allows the reuse of the instantiation of the network connection. On the side of iOS, we have considered Alamofire for light-data and load-image, and URLSession[15] for download and upload files in background. We generate two files, NetworkController.swift, in which are defined all the communication functions and CallingData.swift, which contains all the call parameters. About the cloud, we generate Javascript code based on Node.js using the Express[16] framework. For Openshift and Amazon we generate six files. The main file called index.js, config.js for IP and port configurations, requestHandler.js for containing the definition of all the request handler functions, router.js for mapping the URLs and request handler functions, server.js, which contains the code to set up and run the server and package.json with the list of the application dependencies. Our cloud implementation is based on Docker[17], which is a container where an app runs. Moreover, Docker is an emerging method developed by the open source community for ease the portability of software. Docker allows the automatic deployment of the execution environment, and it contributes to reuse the code in differents cloud providers which support dockers[18]. In case of Amazon, there is an additional file called Dockerfile, which specify the environment requirements of the application. Nevertheless, in case of Openshift, there is no need of a Docker file since it is managed automatically by the platform. Furthermore, about the requestHandlers.js there are some differences of implementation between Openshift and Amazon, due to additional storage restrictions of the second[19].

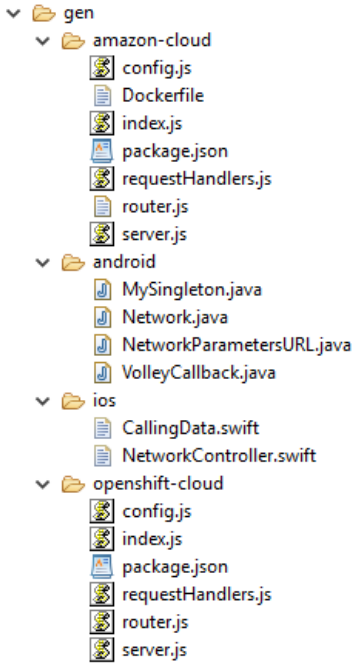


Fig. 4.5: Directory structure of the files which contain the code generated

4.4.2 Code for the User Interface

Besides the code generated from the transformation rules, we have developed a set of user interfaces to show the network communication functions. Therefore, there are two projects, one belongs to Android Studio[20], the framework of Android, and the other to Xcode[21], the framework of iOS. From these two projects the code generated is imported.

[15] URLSession, link: `https://goo.gl/KHw8S1`

[16] Express framework `https://expressjs.com/`

[17] Docker, `https://www.docker.com/what-docker`

[18] List of docker hosting providers`https://goo.gl/5a16vm`

[19] For storing files, it is mandatory the using of the S3 service, link: `https://aws.amazon.com/es/s3/`

[20] Android Studio, link: `https://developer.android.com/studio/index.html?hl=es-419`

[21] Xcode, link: `https://developer.apple.com/xcode/`

Basically, the mobile application has four screens. Each one of them for every type of network communication, light-data, load-image, download and upload files.

The interface for the light-data communication consists in a URL selector, a method selector, a text area to show the communication result and a connection button to perform the communication.

For load-image, there is an image view to show the loaded image, a URL selector, a text area to show a communication result message and the button to get (to perform the communication) and to load the image.

For download, there is a URL selector, a download progress element, a text area to show the result of the communication, a button to perform the communication and a button to cancel the download. In case of Android, the downloaded file can be accessed by the default file explorer of the phone. In case of iOS, once the download is finished, the file is showed automatically.

For upload, there is a URL selector, a text area to show the local path of the file selected to upload, an upload progress element, a button to select and to upload a file and a button to cancel the upload. Once the *select and upload* button is touched, a file explorer is pop up in order to select the file. Once the file is selected, the upload starts automatically.

These user interfaces are pretty general and similar among Android and iOS platforms.

From the controller of the user interface, it is made a reference to access to the network communication implementation, including the communication functions and parameters among other code generated.

In Section 4.6, through an example we can see the described screen interfaces.

4.5 Modeling and Generation with MoWebA Mobile

For modeling we have used MagicDraw[22], which is a modeling tool of general purpose. From MagicDraw we can import the MoWebA's profile and the extended profile. Once the profiles import is finished, it is possible to model the network communication based on the REST architecture and the functions described in the previous section. Once the model is done, it is exported as XMI files. Such files are imported from Acceleo[23], which Acceleo is a modeling tool, based on Eclipse. Acceleo allows the definition of transformation rules templates and the generation of code from models. Therefore, in Acceleo we have defined our templates of transformation rules. Then, we have taken our imported model (in XMI format) as input in order to generate the code for the different platforms, thanks to the transformation rules defined previously. In this sense, the generated code corresponds to the network communication functions modeled.

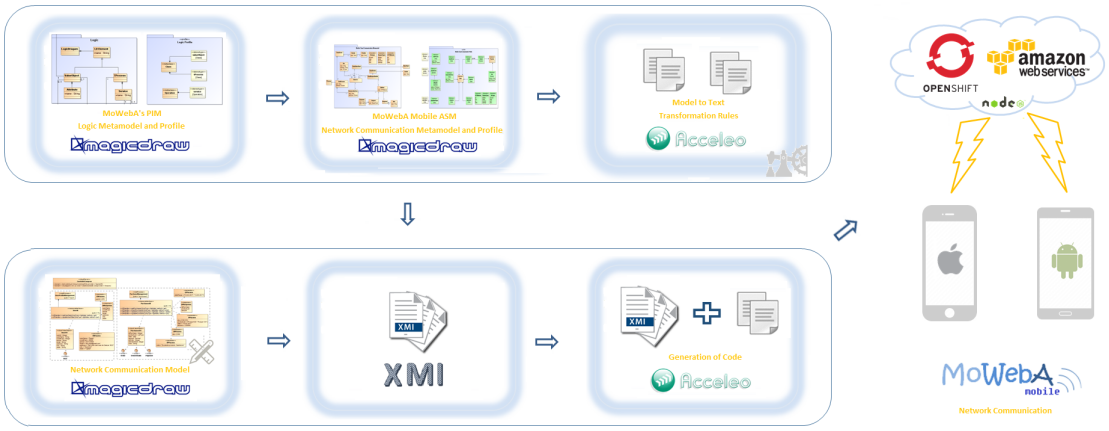


Fig. 4.6: Process for the design and the generation of the network communication between the mobile applications and their functions in the cloud

[22] MagicDraw, link: `https://goo.gl/mLPvur`

[23] Acceleo, link: `https://goo.gl/jgCZhu`

The implementation includes the code for the mobile platforms Android and iOS. In case of the cloud, the code includes the cloud service providers, Openshift[24] and Amazon Web Services[25]. The generated code is ready to be executed for both mobile platforms and for both cloud service providers, already mentioned.

In Figure 4.6 we can see an illustration of the design process.

4.6 Example for the Modeling of the Network Communication

With the aim of improving the understanding of the design process, we present an example of modeling an application. We use MoWebA and the extensions and definitions added to it for designing the network communication.

The application consists in a virtual shop, which includes a set of products to be offered and a set of users, who are the potential purchasers of such products. This application requires the implementation in mobile and cloud platforms.

In our case, the task is focused on the functions of network communication for the data exchange. Basically, the data to be exchanged is about the user (e.g., name, phone, email, address) and about the products (e.g., name, provider, pictures).

Fig. 4.7: Example of an application for offering products. Providers diagram

This example consists in the design of the network communication between a mobile application for offering products and its functions in the cloud. We show the use of the ASM obtained from the extended and new elements of the logic diagram of MoWebA's PIM.

First of all, it should be modeled the service providers in the cloud. To do this, we create a class and assign to it the stereotype *provider*. In this case, there are available two service providers, Openshift and Amazon. Subsequently, the defined providers can be selected from the class stereotyped as *cloudServer*. The defined providers are shown in Figure 4.7.

Following, the *cloudServer* is modeled (see Figure 4.8) through a stereotyped class. As attributes, we specify the *domain* and the *provider* for each service provider. For each *domain* is selected a provider from those defined previously. In this case, we consider Openshift as the cloud service provider (see Figure 4.8). After the *cloudServer*, there are defined the communication processes based on REST, we call to each process *restProcess*. For each *restProcess* and thanks to tag values it is possible to specify a relative *path*, which makes accessible the *restProcess*. For instance, the *restProcess Customer* has as *path "/customer"* (see Figure 4.8). Since, each *restProcess* is composed by a set of resource interfaces, each *resourceInterface* is related to *restProcess* through a composition. Similarly, for each *resourceInterface* is specified the relative *path*. For example, the case of *UserRI* it is specified the *path "/"* (see Figure 4.8). The definition of the *path*'s names depends on the designer criteria. Before each name must be specified the prefix *"/"*.

Each *resourceInterface* is related to a resource. Such relation could be done in three ways. First, through a dependency relation with the *cloudValueObject*, where it becomes to a resource. See the case of *CustomerVO*, in Figure 4.8. Second, through a *cRHandler*, which becomes itself a resource. To illustrate, see the case of *CurrencyRI*, in Figure 4.9. Third, through a dependency relation, from a *cRHandler* to a *fileSet*. This *fileSet* contains several files, where each one of them is a resource. For instance, see the case of *ImageRI*, in Figure 4.9. In this example, *iphone7Frente.png* is a resource. In this way, the final URL is composed of the *cloudServer*'s *domain*, the *restProcess*'s *path*, and the *resourceInterface*'s path. For example, in case of *Purchases*, in Figure 4.8, the URL is *"test-mddcloud.rhcloud.com/purchases/"*. In case of *download* and *loadImage*, the file's name is concatenated in order to get the final URL. As an example, *"test-mddcloud.rhcloud.com/products/images/iphone7Frente.png"* for *ImageRI*, in Figure 4.8. In some

[24] Openshift, link: `https://www.openshift.com/`

[25] Amazon Web Services, link: `https://aws.amazon.com/es/`

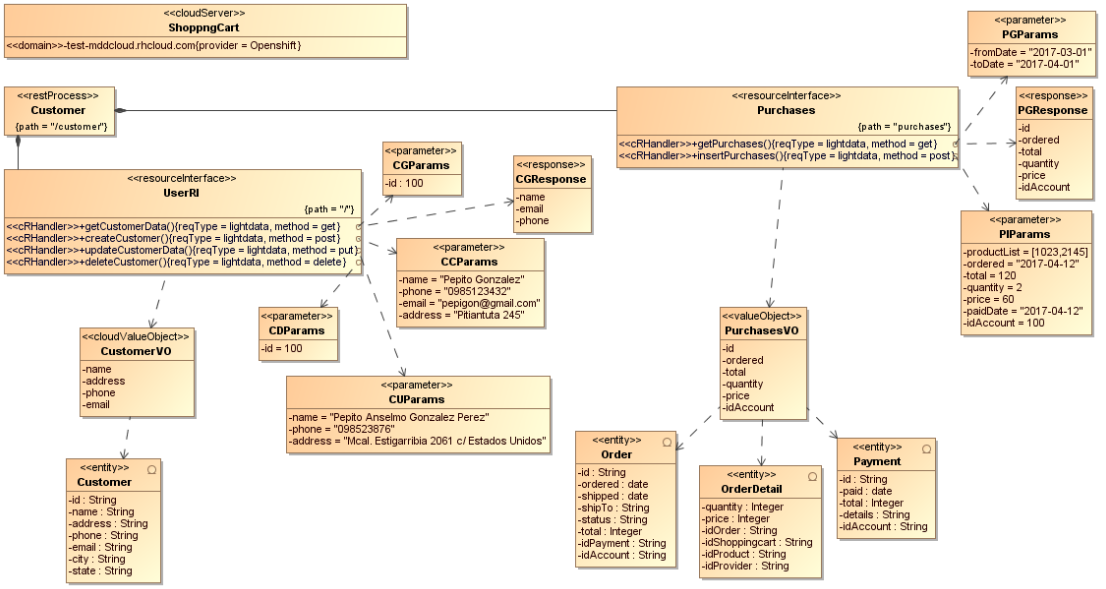


Fig. 4.8: Example of an application for offering products. Customer, purchases diagram

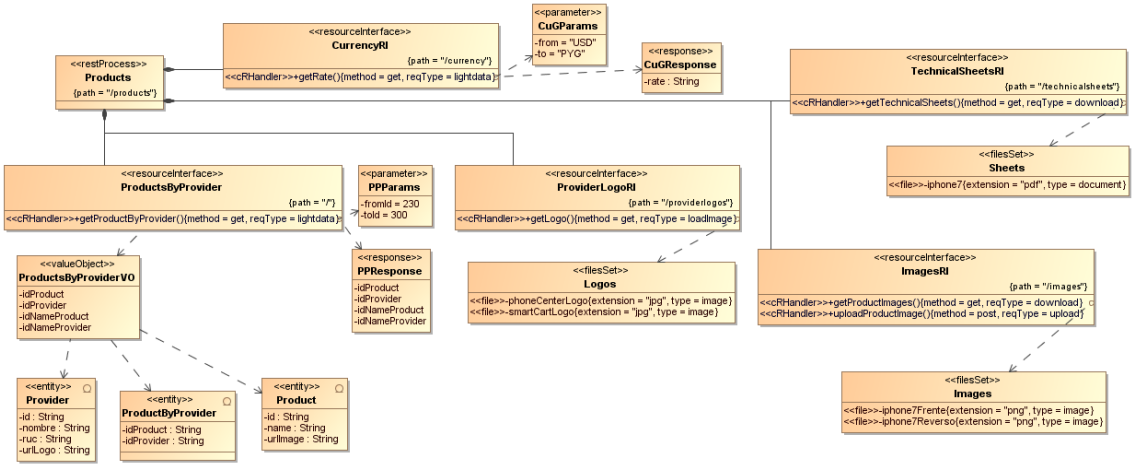


Fig. 4.9: Example of an application for offering products. Products

cases, the *cRHandler* could be a resource. Nevertheless, it always has to handle the requests on its respective URL. In other words, for each URL there exists at least a *cRHandler*. Furthermore, for each combination of a URL and a defined method, there is a *cRHandler*. Each *cRHandler* is set as an operation of a *resourceInterface*. For instance, see all the *cRHandler* in Figures 4.8 and 4.9.

For each *cRHandler* it must be defined the request type or *reqType*. The possible values of *reqType* are *lightdata, loadImage, upload or download*. If the value is *lightdata*, then it also must be defined the *method*, which can be *get, post, put or delete*. The *reqType* and *method* are specified through tag values. For instance, see the case of *Purchases*, in Figure 4.8.

In case of *get method*, it is necessary to specify the *response* to be received. Moreover, if the resource is a *cloudValueObject*, the response is specified according such resource without put again the data types. For instance, see the case of the *cRHandler getProductByProvider* for *Products-ByProvider*, in Figure 4.9.

If a *cRHandler* is a resource, then, the data type in *response* is specified. For instance, see the case of *CurrencyRI*, in Figure 4.9. Similarly, for each *cRHandler* it can be defined request *parameters* in order to send additional information. When *reqType* takes the value of *loadImage*, *download* or *upload*, the *parameters* are delimited to native types (e.g., integer, string). Generally, in practice, such parameters do not need complex representations. Furthermore, in case of *loadImage*, *download* or *upload* there is no need for assigning a method since they have already default ones.

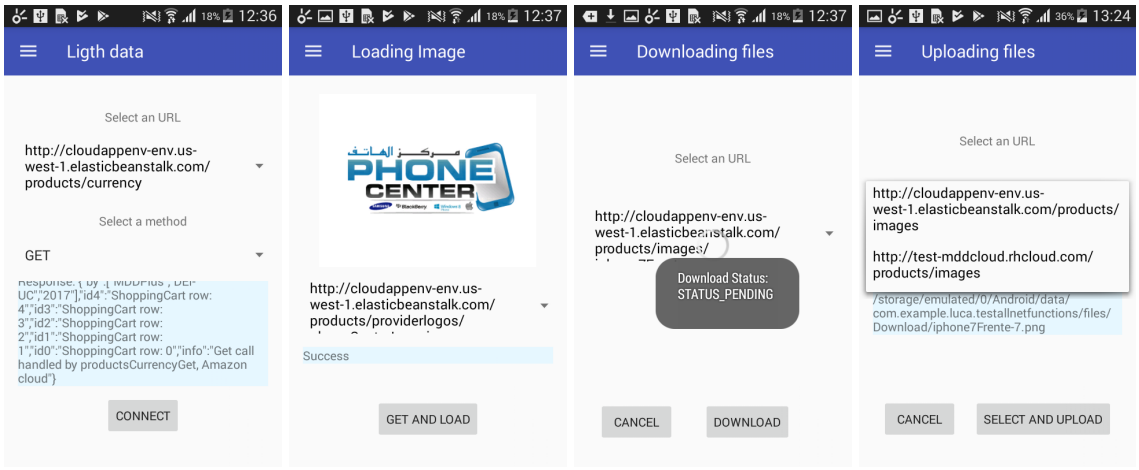
Once the modeling is finished, we proceed to the generation stage. First, it is necessary to export the model from MagicDraw. We export in EMF UML2 (v2.x) XMI format. Then, from Acceleo we

import those XMI files. The import could be done by copying the files from the directory where they are saved and pasting them into a directory inside the Acceleo project. This directory is the same established for the model in the run configurations, see Figure 4.4.

Once the import is finished, we have to set the main model URI property (generally, the main model have the name of the project defined in MagicDraw) to "http://www.eclipse.org/uml2/5.0.0/UML". Then, we can run the rules. The files generated will appears as is showed in Figure 4.5.

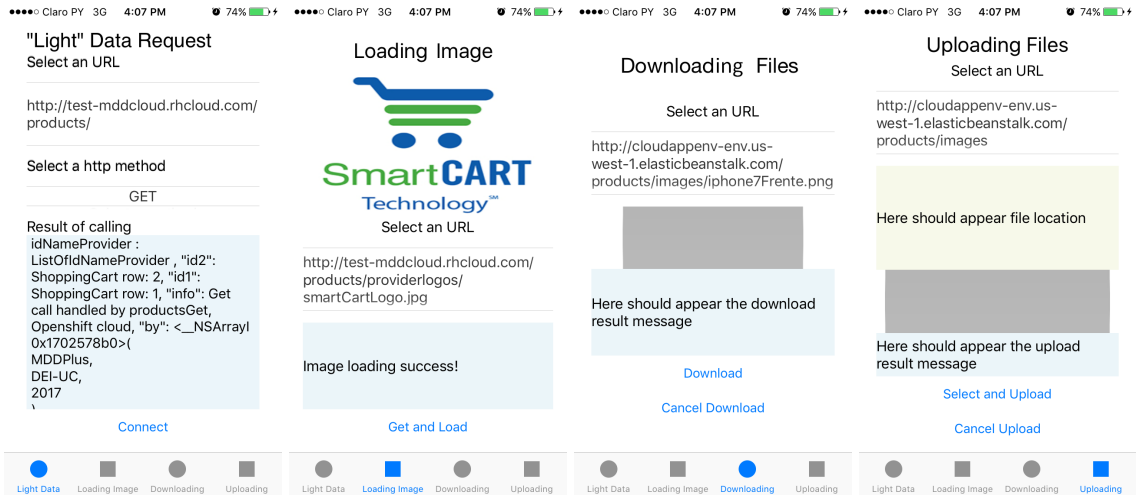
Afterwards, the generated code can be imported from the respective tools of each platform. For the mobile side, the code is imported from the already existing projects (see Section 4.4.2) of Android Studio and Xcode, respectively. In the case of Openshift, the code is deployed via git, being the remote repository a directory in the Openshift cloud. In case of Amazon[26] we do the deployment through its web dashboard, from which it is possible to upload the application files and to do the deployment on Docker. The dashboards of the mentioned cloud service providers are shown in Figure 4.12 and 4.13.

The user interface of the application generated is shown in Figures 4.10 and 4.11, for Android and iOS, respectively. Also, in Figures 4.12 and 4.13 are shown the cloud application dashboards of Openshift and Amazon, respectively.



(a) Light data (b) Load images (c) Download files (d) Upload files

Fig. 4.10: Mobile Application User Interfaces. Android



(a) Light data (b) Load images (c) Download files (d) Upload files

Fig. 4.11: Mobile Application User Interfaces. iOS

[26] Amazon Web Services - Beanstalk, link: https://aws.amazon.com/es/elasticbeanstalk/

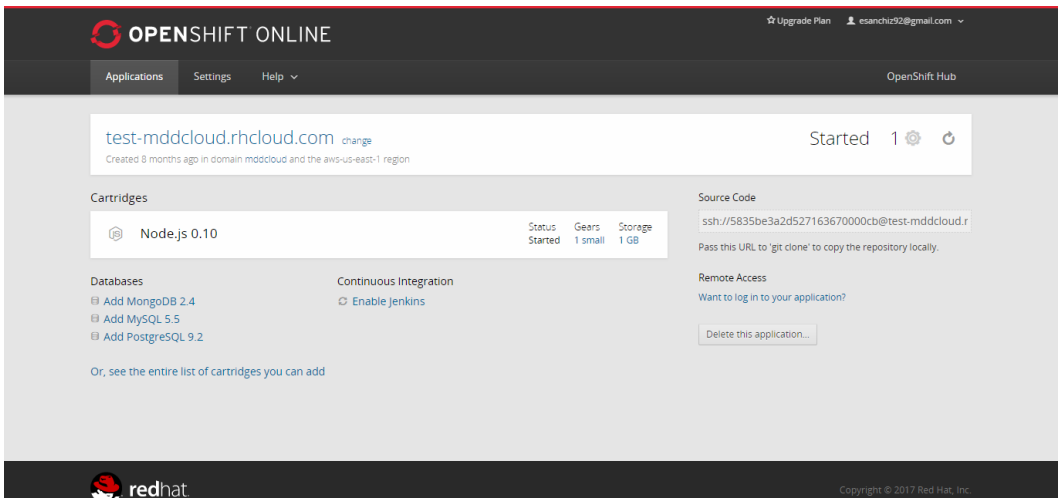


Fig. 4.12: Cloud application. Openshift dashboard

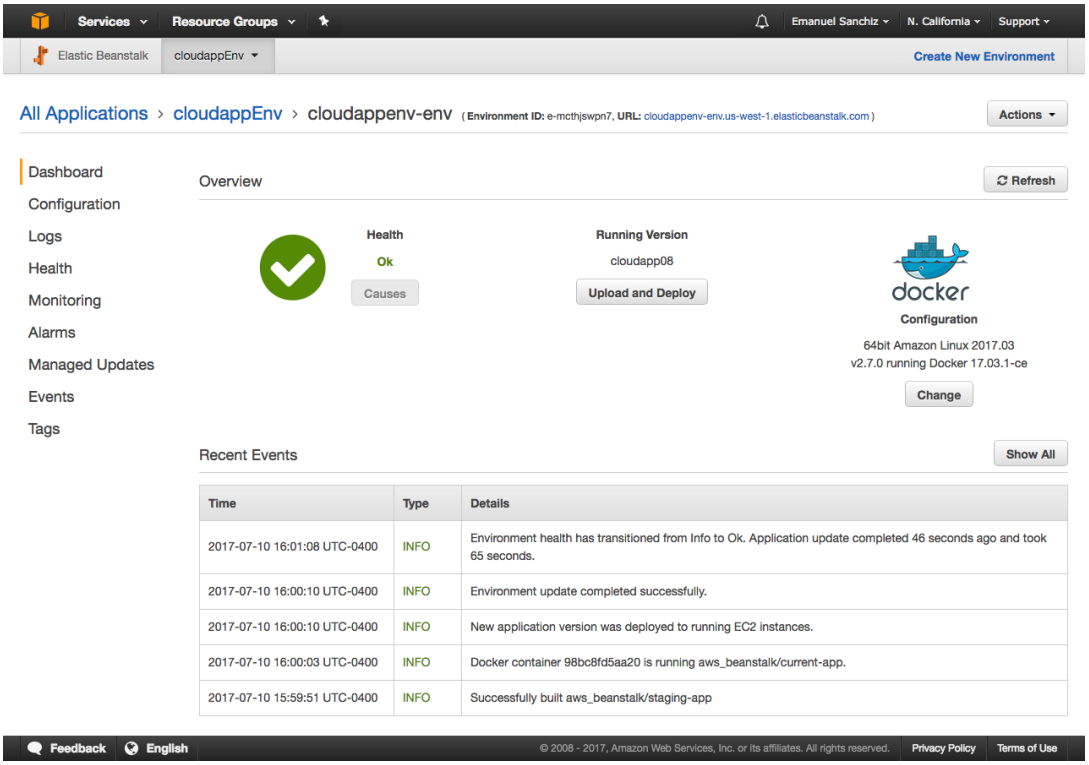


Fig. 4.13: Cloud application. Amazon dashboard

4.7 Summary of the Chapter

In this chapter we have presented our proposal based on what we learned from our SMS of the state of the art. We propose an extension of MoWebA for a unified modeling and generation of the network communication functions of the MobileApps-FC. We have presented the ASM's metamodel, profile and transformation rules for applying the mentioned extension. We based the design of the communication on the REST architecture and in four types of network communication functions: light-data, load-image, download-files and upload-files. From the model, we consider the generation of native mobile applications and open source cloud applications. In this sense, the native code for mobile platforms is preferred over other alternatives (e.g, hybrid, web applications) due to their better properties. Similarly, an open source approach for the development of cloud applications helps to improve the portability of those applications. In addition, we based the cloud implementation on Node.js and Docker. On one side, Node.js is a popular runtime environment which use the most popular programming language, Javascript. Node.js and Javascript are proposed as an alternative

to the implementations based on Java, which is adopted by most of the studies we studied in our SMS. On the other side, Docker is an emerging method developed by the open source community for improving the portability of applications among cloud service providers. Furthermore, we have presented the modeling and generation process and an example for illustrating the modeling and generation of our proposal, which we call MoWebA Mobile.

Chapter 5
Comparative Studies with MoWebA Mobile

In this chapter, we present two comparative studies with MoWebA Mobile. In the first study, we have compared the manual development approach against our proposal.The aim was to get the difference of effort in the development of the network communication functions, among both approaches. More specifically, the development time was analyzed in each case. In this sense, we have used the time to measure the effort, which is one of the main issues when we talk about the difficulty of portability. In this work, the difficult of portability is the addressed problem. Therefore, the interest was to analyze the difference of development times to see if our proposal has a significant improvement in saving effort.

Moreover, in the second study, we have carried out an experience in order to compare MoWebA Mobile against another proposal of the state of the art. For this purpose, according the SMS we presented in Chapter 3, we have selected the most complete and industry used model driven tool, WebRatio Mobile Platform. In this case, the aim of such experience was to identify differences about the modeling and the generation of mobile applications with functions in the cloud (MobileApps-FC). In other words, the interest was to identify differences of MoWebA Mobile regarding one of the most strongest MDD frameworks.

First, we present the comparison between manual development and MoWebA Mobile. Second, we describe the comparative experience between MoWebA Mobile and WebRatio Mobile Platform, including the research questions defined to guide the experience and the respective results. Finally, in the summary, we present a final discussion according the results of the comparative studies.

5.1 MoWebA Mobile vs Manual Development

As we saw in Chapter 4, the transformation rules are part of the implementation of our proposal. In order to built such rules, previously, we had to implement manually the network communication functions. For taking advantage of such manual implementation, we registered the times of development. In this sense, the objective was to compare effort between the manual approach and MoWebA Mobile to get the mentioned functions. In this case, we measured the effort through time.

The network functions implemented included the four types we considered in our work: light-data, load-image, download and upload files. In order to get such functions, we considered an application as an example. Basically, the application is a virtual shop for selling products. The application includes several platforms (Android, iOS, Node.js and the platforms offered by Openshift and Amazon Web Services). Specifically, the mobile application is going to be deployed on tablets and cell phones for field agents, i.e., salesmen that are assigned some geographical areas and go to customers for selling the products. Requirements include the need of periodic or upon-request synchronization of the product catalog with the centralized copy of the data (including product information, photos, and technical sheets). This example is based on the one presented in Brambilla et al. [4]. We have focused on the network communication aspect. Therefore, the considered application was simplified to the network communication functions described in Table 5.1. Such table describes each network communication function to be implemented. The first column refers to the data to be exchanged in the communication. The second column specifies the different methods to be applied on the respective data. The third column refers to the parameters included for each method. The fourth column specifies the response in case of get methods on light-data. Finally, the fifth column presents the specification of files in case of load-image, download and upload files functions. The pair *data (first column) - method(second column)* determines a single function. Just for the purpose of illustration, we show screen-shots of the application type considered in this ex-

ample (see Figure 5.1). The real user interface of the network communication functions developed and generated are shown in Chapter 4, Section 4.6.

Table 5.1: Specification of network communication functions to be developed

Network communication function		**Data to be exchanged in the communication**		
Data to be exchanged	**Method**	**Parameters**	**Response**	**Files**
Customer data (light-data)	get	id = 100	name:string, creditCard:string phone:string, email:string, address:string, city:string, state:string	-
	put (update)	name = “Pepito Anselmo Gonzalez Paiva”, creditCard=“XXXXXXXX” phone = “0985523876”, email=“don.pepi@gmail.com” address = “San Jose casi Juan de Salazar 3022” city=“Asuncion”, state:“Capital”	-	-
Purchases data (light-data)	get	fromDate=“2017-02-14”, toDate=“2017-03-14”	idPurchase:integer, productList:list, amount:numeric	-
	post (insert)	productList = [“100”, “101”], productList = [“Product100”, “Product101”], date=“2017-04-12”, amount=“1250000”	-	-
	delete	id = 100	-	-
	put (update)	reviewNote=“Excelente producto. Satsifecho”	-	-
Products by provider data (light-data)	get	idProvider=230	idProduct:string, idProvider_string, NameProduct:string, NameProvider:string	-
Provider logo images (load-image)	get	-	-	phoneCenterLogo(type = image, extension = jpg), smartCartLogo(type = image,extension = jpg)
Product images (download and upload)	get (download) post (upload)	-	-	iphone7Frente(type = image, extension = png), iphoneReverso(type = image, extension = png)
Technical sheets (download)	get	-	-	iphone7 (type= document,extension=pdf)
Currency data (light-data)	get	From=”USS”, to=”PYG”	currencyRate:numeric	-

The participant, who developed the application, was a bachelor in computer science with no experience in the mobile and cloud development areas, but with two years of experience in modeling. In case of the manual development, the registered times include the learning curve of each platform.

Following, we describe the documentation and tutorials followed for the manual development. In case of the cloud (Node.js, Javascript), the participant consulted The Node Beginner Book[1]. Similarly, the participant followed the official documentations of Node.js [2] and Express.js[3], which is the framework we used. In case of the mobile platforms, for Android[4] (Java) and iOS[5] (Swift), it was followed the official documentations and some additional tutorials from the web.

Following we describe the process we followed to do the comparison. First, the considered application was implemented by means of manual development. Development times were registered

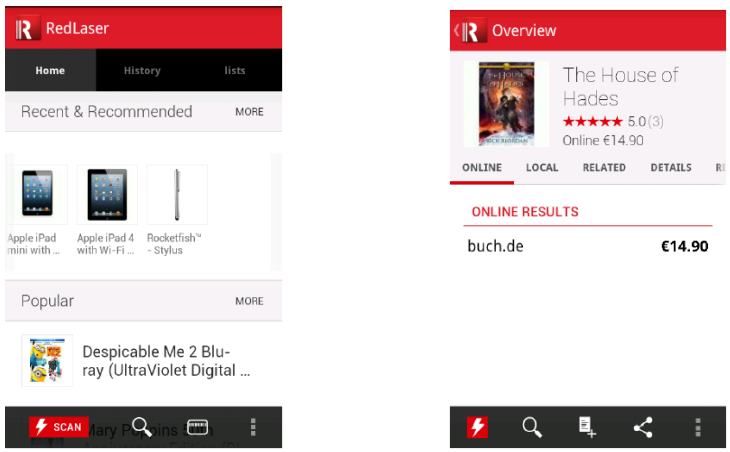
[1] The Node Beginner book, link: `https://leanpub.com/b/node-beginner-and-craftsman-bundle`

[2] Node.js, link: `https://nodejs.org/es/`

[3] Express.js, link: `https://expressjs.com/`

[4] Android developer, link: `https://developer.android.com/training/basics/network-ops/index.html`

[5] Apple developer, link: `https://developer.apple.com/`



(a) Red-Laser. Screen 1 (b) Red-Laser. Screen 2

Fig. 5.1: Red-Laser app. Example of the type of application considered. Taken from Brambilla et al. [4]

by weeks in a spreadsheet. At the end of the week, the hours dedicated each day were calculated and the respective sum was registered in the spreadsheet. Since, the application includes several platforms, the week hours were divided by them. The registered hours are shown in Table 5.2. Second, the same application was modeled and generated using MoWebA Mobile. In this case, the times of modeling were registered by type of functions. For each function, a start time and an end time of modeling were registered. The times were measured in minutes. The summary of the modeling times are shown in Table 5.3. About the generation, it took no more than five minutes.

As a result, the time comparison between the manual approach against MoWebA Mobile, gives a considerable difference of time, as it was expected. Using our proposal takes approximately 1 hour 20 minutes (1:15 minutes of modeling, see Table 5.3 plus 5 minutes of generation) for obtaining the same application. About the manual approach, it takes 356 hours (see Table 5.2). We analyze such numbers. On one side, the manual development was performed by a non-expert developer. Consequently, there is a learning curve time to take into account. Moreover, since the application development includes several platforms and cloud service providers, the participant had to face difficulties which slowed down the manual development. Following we describe some of those difficulties:

- The heterogeneous environments (mobile and cloud) and platforms (mobile: iOS and Android, cloud: Openshift and Amazon) made hard the switching of context. We are talking about different programming languages, libraries, frameworks and operating systems. Switching from one environment or platform to another one results confusing due, for example, to:
  - The way of use of their respective frameworks (e.g., the navigation).
  - The particular programming languages.
  - The restrictions on operations like list initializations, among other issues.
- Dependency of both environments (mobile and cloud) in the communication. The development of the communication needs a synchronization. For example, setting the same url in both sides to do requests to the server from the mobile side and to attend the requests on the server. Also, setting the code in both sides to send and receive parameters according to the type of http method.
- Testing of the implementation. First, we had to test and to fix bugs by platform (module). Once we did the integration (modules integration), we have tested the application as a whole.

On the other side, the participant had already experience in the modeling and generation of application following the model driven approach. In this sense, such situation helped to accelerate the modeling and generation process. Nevertheless, besides such observations, there are clear properties of our approach which support the difference of time among both approaches. Firstly, the automatic generation of the code from the built model. Second, we get an already tested code, so the probability to spend time in fixing bugs decreases. Third, the code of both sides (mobile and cloud) and all the platforms are generated from a unified model which abstracts the individual settings and the respective differences among the environments and platforms. If we have domain-experienced designers and developers, maybe the time of manual development would be

reduced, but due to the mentioned properties the difference among both approaches would remain considerable.

Table 5.2: Manual development times

Environment	Platform	September				October				November				December				January				February				March				Total per Platform
Cloud	Node.js	20	20	4	4																									48
	Amazon																					20	12				4			36
	Openshift					12	12						12										12				4			52
Mobile	iOS												20	20	20	20		20	4				4			12	4			124
	Android																			32	32			20	4	4	4			96
Total per week		20	20	4	4	12	12	0	0	0	0	0	32	20	20	20	0	20	4	32	32	20	28	20	4	16	16	0	0	
Observations: Time in hours - Months divided by weeks																										Total				356

Table 5.3: Modeling times registered using MoWebA Mobile

Tool	Light-data functions	Load-image functions	Download-upload files functions	Total
MoWebA Mobile (A)	Cloudserver: 3 min. Lightdata-customer: 23 min. Light-data purchases: 11 min. Light-data product-ByProvider: 11 min. Light-data currency, function as resource: 5 min. Subtotal: 53 min.	Load-image logo: 6 min. Subtotal: 6 min.	Download-images: 8 min. Download-sheets: 4 min. Upload-images: 4 min. Subtotal: 16 min.	1:15 min.

5.2 MoWebA Mobile vs WebRatio Mobile Platform

Following, we present as well a comparison experience. In this case we compared MoWebA Mobile against WebRatio Mobile Platform, a model driven framework of the state of the art which is used in the industry.

Our objective was to do a comparison experience between two tools, MoWebA Mobile (A) and WebRatio Mobile Platform (B), for resolving the modeling and the generation of the communication of the mobile applications with functions in the cloud (MobileApps-FC). The purpose is to see if our proposal has considerable modeling and generation differences with one of the most representative MDD tools of the state of the art (according our SMS). If there are such differences, we could understand if our proposal is well routing towards becoming an alternative for the modeling and generation of the MobileApps-FC.

In this case, the same application described in the previous section was considered, with the respective network communication functions specified in Table 5.1. Similarly, the participant was also the same described in section 5.1.

We have defined four research questions used to guide the comparison. Following we cite the research questions:

- **RQ1**- What differences of network communication modeling are between A and B?
- **RQ2**- What kind of code do A and B generate?
- **RQ3**- Respectively, how many platforms do A and B generate code for (mobile and cloud)?
- **RQ4**- How much time do A and B require to model and to generate the network communication?

Through the defined research questions, we emphasized on the differences among the proposals. In this case, the modeling and generation are the main aspects. Therefore, RQ1, RQ2 and RQ3 refer to the modeling and the generation differences. About RQ1, we have looked for any difference in the modeling approach. For RQ2, we have compared the two kind of code generated, for analyzing the pros and cons. In case of RQ3, we analyzed the target platforms included. Finally, RQ4 refers to

the time of modeling to understand and to analyze the differences about the effort in the modeling process. Similarly, in this case, we have measured the effort through time.

Following we cite the documentation, which supported and guided the modeling and generation process:

- Chapter 4 of this Book: Proposed Solution, MoWebA Mobile, for the modeling and generation of the network communication functions.
- Online documentation for WebRatio Mobile Platform[6]. Documents and specifications available online[7].

We have modeled and generated an application focused on communication using both tools, MoWebA Mobile (A) and WebRatio Mobile Platform (B). In the following, for each research question, we present the results. Each result includes the difference among A and B, following by the analysis/explanation (X).

RQ1- What differences of communication modeling are between A and B?

We divided the answers in four parts. First part, the communication based on data model as resource, where the data exchanged comes from the data model. Second part, the communication based on remote functions as resources, where those functions produce a result which is consumed by the mobile side. Third part, communication based on files as resources, where the data exchanged are files and those files are stored in directories. Fourth part, we describe the differences from a general perspective.

Communication based on data model as resource

A: The modeling starts from value objects, which represents an additional level of abstraction on the data model (entities and relationships). **B:** The communication is modeled directly on the entities. **X:** As an additional level of abstraction on the data model, the value objects of A allow the grouping of attributes of one or more entities. Further, the value objects prevent the building of a service for every entity (containing data to be created, updated, read or deleted remotely) to be used in the communication process. In case of B, if the data of an entity is going to be accessed, modified or deleted remotely, then a data service have to be assigned to that specific entity. For instance, if we have two entities *order* and *order detail*, using B it is necessary to build a service for each entity. In case of A, it is possible to define a value object *order data*, which can contain the respective attributes of *order* and *order detail*. Consequently, the service is built only on *order data*.

A: It does not consider yet an automatic synchronization method among entities. **B:** In case of having in the mobile side a partial replica of the remote database, B presents an option for an automatic synchronization between entities of the back-end data model and entities of the mobile side data model. The data to be synchronized varies from simple data types to files. **X:** In case of having a partial replica of the remote database in the mobile application, B presents a method of synchronization, where all the functions and settings for synchronizing the entities (local and remote) are generated automatically. In this sense, WebRatio abstracts the developer to do the synchronization with such method. In contrast, in MoWebA Mobile we have not considered yet synchronization methods since we were focused on the data exchange itself.

Communication based on remote functions as resources

A: The modeling of remote functions is based on REST. Nevertheless, it does not consider web services concepts. It does not include the modeling of the function's body. **B:** It defines remote functions through web services. The modeling includes the function's body. **X:** The modeling and the generation of the communication to remote functions allow the access to data generated or processed remotely. For example, some of the remote functions could be those obtaining statistical data, average exchange rate (which consists in obtaining several rates from different services and then do an average of them), or others which do complex or resource consuming operation. Using A, the designer has to set parameters and the response to model the communication without setting any web service configuration. In addition, the modeling is done trough a single model. In contrast, B defines remote functions through web services. Therefore for implementing a remote function it is necessary to manage concepts of web services (e.g., XSD Provider, XSD Resource, Service description) and configure them to get the complete modeling of the function. Also, with B, the communication has to be modeled from the back-end (exposing web service) side and for

[6] WebRatio Learn, link:`https://my.webratio.com/portal/content/es/learn`

[7] WebRatio - Resources, link: `https://www.webratio.com/site/content/es/recursos`

the mobile side (invoking web service). Nevertheless, the modeling of B includes the definition of the function's body, which implies a complete modeling of the remote function. In A, we have not covered such aspect yet.

Communication based on files as resources (files stored in directories)

A: It does all the modeling and configurations in a unified project and model. **B:** There are, on one side, configurations to do in the back-end project and, on the other side, modeling in the mobile project. In this case, the communication has to be done manually in B (without the automatic synchronization method, mentioned above). It is necessary to model the invocation of the REST service. **X:** Using A, all the design for the network communication, for file exchange, is done in a single model. Nevertheless, using B there are settings to do in the back-end project, for exposing the respective service, and a modeling to invoke the service exposed in the mobile project. Working with a single model there is an abstraction for the designer from the different environments.

A: Through the specification of REST processes and resource interfaces, it is possible to customize the access paths to the files. **B:** From the back-end side for a manual communication communication, there are default paths for download and upload files. **X:** The customization of the path allows to organize the access to files by type (e.g., document, images, audio, video), by category (e.g., products images, provider logos), and so on. In A, it is possible to get such customization. In case of B, there are two default paths, for downloading and for uploading files. These paths could be customized from a setting. Nevertheless, those paths are limited to two and it is not possible to customize them according to specific type or categories of files. There will be necessary additional (besides the modeling) adjustments to do such customization.

Differences, in general

A: All the modeling process is performed with the same model and project. **B:** The modeling process is divided in two models and projects. One project for the mobile side, another project for back-end side. **X:** In A, the purpose of working with the same project and model for designing the communication is to abstract the developer from individual settings and modeling by each side (mobile and cloud). In other words, since the communication implies two sides (mobile and cloud), from a unified model, at the moment of generation, the design and settings are replicated in both sides. Therefore, the developer "works once" instead of twice or more. For instance, if a url is modified, then, a developer does not modify it for the cloud side and for the mobile side, the url modification is made only once in the model. Afterwards, thanks to the transformation rules, the change is replicated in both, the mobile and cloud sides.

A: There is no authentication. **B:** WebRatio requires authentication for connecting to the cloud. **X:** B includes the generation of the authentication process. In A, we have not work yet on the security aspect, it is a pending item since for production environments the authentication is a necessary requirement. Nevertheless, during the development process, in case of a prototype, the authentication could be not necessary yet. May other functional requirements have priorities at the stage of prototyping. In this sense, A could be considered to get a prototype. Nevertheless, it is still a pending item for A.

A: There is no need to connect to a database to test the communication. It is possible to add data in the model for testing. **B:** The connection to a database is a requirement to test the communication. **X:** Since, at this stage, A is focused exclusively on the communication there is no need of a connection to a database for testing the communication functions. Such test could be done with test values specified in the models. Without the need of database connection, the communication can be tested as an independent module. Nevertheless, the database integration is also a pending item for A.

A: We model the communication allowing the grouping of the communication functions through resource interfaces and REST processes. **B:** In case of the manual communication, in B the web services modeling is done individually for each service. **X:** In A, the communication functions are grouped by resource interfaces (or URL). This means, if there is more than one communication function on a resource interface, from the second one it is possible to reuse part of the modeling made for the first one. At the same time, the resource interface is grouped by REST processes. In contrast for A, if a URL have more than one service associated, then, for each service the designer has to model the service invocation. Therefore, there is no such similar reusing in the modeling.

RQ2- What kind of code do A and B generate?

Mobile Side

A: Generation of native code for each platform (iOS and Android). Such code can be imported, respectively, from each native IDE (Xcode and Android Studio). **B:** Generation of hybrid imple-

mentation. The models generates source code in HTML5, CSS3, Javascript and then, the Apache Cordova compiler is used for generating the hybrid mobile applications (for iOS and Android). **X:** Native applications have the best performance, highest security, and best user experience. Therefore, native applications are preferred by users over the hybrid ones[8]. With the generation of native code for the communication, A is following a process to build an application 100% native in contrast to B, which generates hybrid applications. The generation of hybrid applications is a way to alleviate the effects of the portability issues. Therefore, in this case the Apache Cordova[9] takes the code generated by B and it generates the application for running on the target platforms. In other words, the Apache Cordova abstracts the differences among platforms. Nevertheless, if we are using an MDD (Model Driven Development) approach, the models also abstracts the designer or developer from platform differences. In this sense, with A, we take advantage of such MDD property and we generate native applications, which are preferred over the hybrid ones.

Cloud Side

A: Generation of open source Node.js server. **B:** Generation of open source Java EE server application. **X:** Besides B, other of the proposals selected in our SMS consider the generation of a Java Server application. None of them considers other emerging and already popular alternatives as Node.js[10], which has interesting properties[11], like:

- The use of the most popular programming language, Javascript[12].
- It is lightning fast. Node.js is primarily a JavaScript runtime that is powered by V8, developed by Google for use in Chrome. V8 has the ability to compile and execute JavaScript at lightning fast speed.
- It is lightweight. Typically, Node.js uses a basic event-driven architecture. This means everything executed on it, including every single operation and call, is a set of asynchronous callbacks. This enables Node to run on a single thread as unlike other web technologies where a new thread is spawned for every client request.
- High Performance. There are statistics which presents positive numbers for the performance of an application built in Node.js. PayPal uses Node.js, and has reported doubling the number of requests per-second and reducing the response time by 35%.

In this sense, A presents an cloud implementation alternative based on a popular technology, which have interesting properties.

RQ3- Respectively, how many platforms do A and B generate code for (mobile and cloud)?

Mobile Side

A: Two (iOS and Android). **B:** Two (iOS and Android). **X:** Both A and B consider the most popular mobile platforms, iOS and Android. In case of B, they get hybrid mobile applications trough the Apache Cordova.

Cloud Side

A: Two (Openshift and Amazon). The generation is based on Dockers[13], which ease the portability among different cloud providers. **B:** One (their own cloud platform). **X:** B generates a running Java application for its own cloud service platform. Thanks to its development tools, the application can be automatically deployed in the cloud. In case of A, it generates an implementation to run in two different clouds (Openshift and Amazon). Furthermore, the implementation is based on Docker, which is a container where an application runs. Moreover, Docker is an emerging method developed by the open source community to ease the portability of software, in this case of cloud applications. Docker allows the automatic deployment of the execution environment, and it contributes to reuse the code in different cloud providers (which support dockers[14]). In the case of A, the cloud application is modularized into five modules; four of them are directly ported. In contrast, some adjustment according to the service supported by the provider should be performed in one of the modules (requestHandlers.js, in the case of considering Openshift and Amazon Web Services).

---

[8] Native vs hybrid application, link: `https://goo.gl/JH1u7K`

[9] Apache Cordova, link: `https://cordova.apache.org/`

[10] Node.js statistics, link: `https://goo.gl/EBmNKS`

[11] Node.js properties, link: `https://goo.gl/TCbTgC`

[12] Javascript popularity, `https://goo.gl/H6tJmA`

[13] Docker, link: `https://www.docker.com/what-docker`

[14] List of docker hosting, link: `https://goo.gl/SPM4Km`

Besides the code of the application, the dependencies and the environment are built automatically. We generate code for two cloud providers considering the case where different clients require the same application but have already different cloud providers. In addition, the generation for several providers could be interesting if the client requires a replica of its application in the cloud into other service provider[15], or if they require a migration from the current service provider to an alternative provider.

RQ4- How much time do A and B require to model the network communication?

Table 5.4: Differences between MoWebA Mobile (A) and WebRatio (B). Answer to RQ4, modeling time.

Tool	Communication based on data model as resource	Communication based on remote functions as resources	Communication based on files as resources (storing files in directories)	Time to connect to the database	Total
A	Cloudserver: 3 min. Lightdata-customer: 23 min. Light-data purchases: 11 min. Light-data product-ByProvider: 11 min. Subtotal: 48 min.	Light-data currency, function as resource: 5 min. Subtotal: 5 min.	Load-image logo: 6 min. Download-images: 8 min. Download-sheets: 4 min. Upload-images: 4 min. Subtotal: 22 min.	No aplica	1:15 min.
B	Set the services for entities: 3 min. Time to replicate database: 5 min. Subtotal: 8 min.	Exposing the service (currency function): 15 min. Invoking the service (currency function): 6 min. Subtotal: 21 min.	Load-image : 5 min. Download-images: 5 min. Download-sheets: 5 min. Upload-images: 5 min. Subtotal: 20 min.	Connect to database: 4 min. Subtotal: 4 min.	53 min.

Table 5.4 presents modeling times.

X: First of all, we explain the times in Table 5.4. According the functions we defined in Table 5.1, there were several communication functions to model. In case of the registered times, such minutes are those to finish the modeling required by the mentioned defined functions.

At first glance, B have a better modeling performance than our proposal. Such situation is due since B has a specific development environment, which eases, makes simple and faster the modeling comparing with our modeling tool. Furthermore, B is a platform with experience in the industry and covers many other aspects besides the communication. In contrast, our proposal is in a stage where the implementation is done through a prototype for modeling and generating the communication. Consequently, we have done a comparison of an experienced specific tool, which is B, against an academic prototype, which is A.

For the communication based on data model as resource, there is an important difference of time since B automatizes the synchronization and abstracts the designer from the modeling of the communication. Just with few clicks and check boxes it is possible to get the communication based on the data model. In case of A, there is a modeling to do. Such modeling implies the adding of classes, properties, tag values and relations, which through a general purpose tool (MagicDraw, used by A) takes a considerably time to get the communication design on the data model.

About the communication based on remote functions as resources, our proposal is made with a single modeling. In case of B, there is a modeling to do from the back-end project (for exposing the service) and another one from the mobile project (for invoking the service). Furthermore, the function is implemented trough web services. Therefore, in the modeling of the service exposing there are included concepts and properties of web services. In case of A, those additional details are not considered since we based the communication on REST but not on web services. In this case, B takes more time because the process of exposing the service includes the modeling of the service's body. In case of A, it does not include yet modeling of the service's body. In this case, it is necessary to add to A the service's body modeling in order to get a fair and a better comparison. In case of the communication based on files as resources (storing files in directories), the times are quite similar. For A, the modeling for each time of function for transferring files (for download, upload and load-image) are similar to the remote function modeling. In case of B, there is no modeling in the back-end. Instead of modeling there is a setting to do. From the mobile side it is necessary to model the invocation of the services.

[15] Problems with service providers, `https://goo.gl/bbNRXZ`

For the time to connect to the database, in case A there is no need to establish a connection to a database. In the modeling it is possible to define temporal data for testing the communication implementation. For B, it is necessary to connect to a database to test the communication.

In summary, B's times are the 70% of A's times. At first glance, the difference is not so great, considering that we are comparing an academic prototype with a tool used in the industry. Nevertheless, we have to consider that in the modeling of the network communication, unlike A, B includes others aspects like the modeling of the service's body and the connection to database. In this sense, such aspects implies a considerable time of modeling. Therefore, if A would include such aspects, its modeling time will increment. In such case the time difference will be greater. Although, as we already mentioned, A is a prototype, so, if as future work we improve the modeling tool and other modeling aspects, then, also the modeling time will be reduced.

5.3 Summary of the Chapter

Finally, we present a final discussion related to the comparative studies that were carried out.

First, regarding the manual development, MoWebA Mobile offers a reduction of development effort. This effort reduction can be seen from the differences in hours of the development process using the manual approach and our proposal, MoWebA Mobile.

Second, we cite those aspects covered by our proposal and non-covered by WebRatio Mobile Platform:

- The design of the communication based on the data model is not restricted to entities; it is based on value objects, which allow working with data from several entities. In this sense, there is no need to build a communication function for each entity.
- It is possible to work with a unified modeling for the mobile and cloud sides.
- It is possible to model and generate a communication to access to remote functions without using web service concepts.
- The customization of the path, in the model, allows organizing the access to files.
- The communication functions can be tested independently, since, there is no need to connect to a database. It is possible to add testing data in the model.
- Generation of native implementations for mobile platforms, iOS and Android.
- Implementation of the cloud application based on Docker, which is an emerging open source method to improve the portability of the application through the cloud service providers.
- Implementation of a cloud application built in Node.js, an emerging and popular runtime environment with interesting properties, as an alternative to Java.

In this sense, we have identified these aspects as differences in the modeling and generation approach of our proposal regarding WebRatio Mobile Platform, which is one of the most representative MDD tools of the state of the art, according our SMS presented in Chapter 3. Therefore, we could say that our proposal presents some interesting characteristics and is well routing towards becoming an alternative for the modeling and generation of network communication of the MobileApps-FC. Nevertheless, to ensure that, it will be necessary more formal validations of our proposal in order to identify more aspects to be improved.

Finally, as we listed the aspects covered by our proposal and non-covered by WebRatio Mobile Platform, following, we list the main additional aspects only covered by WebRatio:

- An automatic synchronization of data between the mobile data and the remote one.
- The modeling and generation of the remote functions' body.
- The modeling and generation of the authentication function, which is a requirement in the production environment.
- The modeling of the connection to database.
- The building of a specific tool which integrates the process of modeling and generation. Furthermore, the tool should help to make simpler and faster the modeling process.

In this sense, it will be interesting if in future works we integrate such aspects to MoWebA Mobile.

Chapter 6
Conclusions

We have presented the proposal of a Model Driven Development (MDD) approach for the development of the Mobile Applications with Functions in the Cloud (MobileApps-FC), MoWebA Mobile. Specifically, we have proposed the modeling and generation of the network communication between the mobile applications and their functions in the cloud. In this chapter, we present a summary of the contributions of our work. In addition, we describe several limitations and, consequently, we propose a series of issues for future works.

6.1 Summary of contributions

In summary, following, we present the contribution of our work.

First, a systematic mapping study (SMS) of MDD approaches, focused on the portability problem, for the modeling and the generation of the MobileApps-FC. This SMS was published at the Latin American Informatics and Computer Science Forum (CLEI 2016) [11]. Furthermore, it has been accepted for publishing at the CLEI Electronic Journal 2017[1] [12].

Second, the proposal of the adoption of MoWebA, a MDD approach, for the modeling and generation of the MobileApps-FC based on three aspects: i) incorporation of an Architecture Specific Model (ASM) as a new modeling layer; ii) clear separation of the presentation layer with regard to the navigation and behavior layers; and iii) definition of the navigational structure according to a function-oriented approach. None of the selected studies in our SMS, includes at the same time such aspects as MoWebA does. Such aspects could have a positive impact on the design portability. Nevertheless, more studies and research are necessary to get a clear assess such positive impact. In that way, this work is intended to offer an opportunity to further explore the benefits of the mentioned aspects. We call to our proposal, MoWebA Mobile.

Third, an extension (Architecture Specific Model - ASM) of MoWebA for the unified modeling and the generation of the network communication between the mobile applications and their functions in the cloud. The unified modeling is based on the REST architecture and on four types of network communication functions: light-data, load-image, download and upload files. From the modeling of the ASM, it is possible to get running code for mobile platforms (native code for Android and iOS) and cloud platforms (open source code for Openshift and Amazon, based on Docker). For obtaining the running code, previously, we have built transformation rules, which map the model to the respective code.

Fourth, two comparative studies with MoWebA Mobile. First, the traditional (manual) development against our approach. In this case, the metric was the effort, which was measured through the time of development (hours). Second, WebRatio Mobile Platform, an MDD platform used in the industry, against our approach. For this comparison, we have focused on the differences related to modeling and the generation. Furthermore, in the first study, we have taken a common example to do the comparison. We have registered the times of manual development of the network communication. These times were registered in clock hours. The hours correspond to the development of each type of communication function considered in this work. Then, we have done the modeling and generation of the same functions using our approach. The difference of time among the two approaches was considerable. Consequently, in this case, we have concluded the saving of effort using MoWebA Mobile, as expected. In the second study, we have carried out a comparative experience between WebRatio Mobile Platform and MoWebA Mobile. We have taken the same

[1] https://www.clei.org/clei-electronic-journal/

example of the first study. We have identified differences at the modeling and generation levels. We highlight the properties of our proposal of having a unified modeling, which generates native code for mobile platforms and open source code for the cloud platforms. In addition, we highlight the main aspects, covered by WebRatio an non-covered by MoWebA Mobile, in the modeling and generation of the network communication. Some of such aspects are: an automatic synchronization of data between the mobile data model and the remote one, the modeling and generation of the authentication function, the modeling of the connection to database, the building of a specific tool which integrates the process of modeling and generation and makes faster and simpler such process.

6.2 Limitations of the Proposal

In this section, we summarize the main limitations of our work:

- Considering the complete extension process of MoWebA, there is pending the building of transformation rules between the PIM and ASM.
- The network communication functions we considered are limited to the data exchange itself. Nevertheless, there are other aspects closely related to the network communication which has to be taking into account for a real application. Some of those aspects are: a synchronization method, the authentication of the connection, the database integration, among others.
- For the modeling and the generation, we have focused on a specific aspect of the MobileAppsFC. There are other fundamental aspects like the user interface and the data model, which has to be considered.
- Since our proposal is an academic prototype, the modeling of MoWebA Mobile is done through a general purpose tool, which is MagicDraw. Furthermore, the generation is carried out in a second tool, Acceleo. This situation make tedious and slow down the modeling and the generation process.
- The comparative studies we have done were preliminary validation. In order to have a more formal validation and conclusions, there are necessary experiments and further studies.

  Based on the mentioned limitations we propose the future works in next section.

6.3 Future Works

For concluding, considering the limitations presented previously, we propose the next future works which can address them:

- Build the transformation rules for the automatic transition from the PIM to the ASM, which includes the steps iv) and v) of the MoWebA extension definition [29].
- Add to the communication modeling and generation, security aspects like the authentication process, required for production environments.
- Add other extensions for MoWebA Mobile, for covering other aspects of an application (e.g., user interface).
- Build a specific tool for making simpler and faster the modeling. For instance, the creation of classes, properties, relations, tag values are 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 built in MagicDraw and import them from Acceleo to run the transformation rules and, consequently, obtaining the code.
- An experiment with the specific tool mentioned in the previous point. It will be possible to do a comparison with other tool of the industry (e.g., WebRatio), considering a population of students or professionals. It could be compared the usability of the tools based on standard questionnaires among other measures.
- Integrate our work with the proposal of Núñez et al. [50]. Núnez et al. proposes an extension of MoWebA for the modeling and generation of the data persistence of mobile applications. In this sense, it is an aspect of a mobile application which we do not cover through our proposal.

References

1. V. March, Y. Gu, E. Leonardi, G. Goh, M. Kirchberg, and B. S. Lee, "Mcloud: Towards a new paradigm of rich mobile applications," *Procedia CS*, vol. 5, pp. 618–624, 2011. [Online]. Available: https://doi.org/10.1016/j.procs.2011.07.080
2. D. Sahu, S. Sharma, V. Dubey, and A. Tripathi, "Cloud computing in mobile applications," *International Journal of Scientific and Research Publications*, vol. 2, no. 8, pp. 1–9, 2012. [Online]. Available: https://pdfs.semanticscholar.org/dd37/7136624a0535bd7d3777536ba6e543a3efa9.pdf
3. S. Abolfazli, Z. Sanaei, A. Gani, F. Xia, and L. T. Yang, "Rich mobile applications: genesis, taxonomy, and open issues," *Journal of Network and Computer Applications*, vol. 40, pp. 345–362, 2014. [Online]. Available: https://doi.org/10.1016/j.jnca.2013.09.009
4. M. Brambilla, A. Mauri, and E. Umuhoza, "Extending the interaction flow modeling language (IFML) for model driven development of mobile applications front end," in *Mobile Web Information Systems - 11th International Conference, MobiWIS 2014, Barcelona, Spain, August 27-29, 2014. Proceedings*. Springer International Publishing, 2014, pp. 176–191. [Online]. Available: https://doi.org/10.1007/978-3-319-10359-4_15
5. Z. Sanaei, S. Abolfazli, A. Gani, and R. H. Khokhar, "Tripod of requirements in horizontal heterogeneous mobile cloud computing," *CoRR*, vol. abs/1205.3247, 2012. [Online]. Available: http://arxiv.org/abs/1205.3247
6. P. Gupta and S. Gupta, "Mobile cloud computing: The future of cloud," *International Journal of Advanced Research in Electrical, Electronics and Instrumentation Engineering*, vol. 1, no. 3, pp. 134–145, 2012.
7. E. A. N. da Silva, R. P. Fortes, and D. Lucrédio, "A model-driven approach for promoting cloud paas portability." in *CASCON*, 2013, pp. 92–105. [Online]. Available: http://dl.acm.org/citation.cfm?id=2555523.2555535
8. H. Heitkötter, S. Hanschke, and T. A. Majchrzak, "Evaluating cross-platform development approaches for mobile applications," in *Web information systems and technologies*. Springer, 2013, pp. 120–138. [Online]. Available: https://doi.org/10.1007/978-3-642-36608-6_8
9. C. Pons, R. Giandini, and G. Pérez, *Desarrollo de Software Dirigido por Modelos*, E. de la Universidad Nacional de La Plata, Ed. Editorial de la Universidad Nacional de La Plata (EDULP)/McGraw-Hill Educación, 2010.
10. M. Brambilla, J. Cabot, and M. Wimmer, *Model-Driven Software Engineering in Practice*, M. &. Claypool, Ed. Morgan & Claypool, 2012. [Online]. Available: https://doi.org/10.2200/S00441ED1V01Y201208SWE001
11. E. Sanchiz, M. González, N. Aquino, and L. Cernuzzi, "Mobile cloud applications development through the model driven approach: A systematic mapping study," in *Latin American Computing Conference, CLEI 2016*. IEEE, oct 2016, pp. 605–614.
12. E. Sanchiz, M. González, N. Aquino, and L. Cernuzzi, "Development of mobile applications with functions in the cloud through the model driven approach: A systematic mapping study," *CLEI electronic journal*, vol. 20, no. 3, December 2017.
13. S. Casteleyn, I. Garrig'os, and J.-N. Maz'on, "Ten years of rich internet applications: A systematic mapping study, and beyond," *ACM Transactions on the Web (TWEB)*, vol. 8, no. 3, p. 18, 2014.
14. N. Fernando, S. W. Loke, and W. Rahayu, "Mobile cloud computing: A survey," *Future Generation Computer Systems*, vol. 29, no. 1, pp. 84–106, 2013.
15. Z. Sanaei, S. Abolfazli, Member, A. Gani, and R. Buyyag, "Heterogeneity in mobile cloud computing: taxonomy and open challenges," *Communications Surveys & Tutorials, IEEE*, vol. 16, no. 1, pp. 369–392, 2014.
16. M. Shiraz, A. Gani, R. H. Khokhar, and R. Buyya, "A review on distributed application processing frameworks in smart mobile devices for mobile cloud computing," *Communications Surveys & Tutorials, IEEE*, vol. 15, no. 3, pp. 1294–1313, 2013.
17. A. ur Rehman Khan, M. Othman, S. A. Madani, and S. U. Khan, "A survey of mobile cloud computing application models," *Communications Surveys & Tutorials, IEEE*, vol. 16, no. 1, pp. 393–413, 2014.
18. Y. Wang, R. Chen, and D.-C. Wang, "A survey of mobile cloud computing applications: Perspectives and challenges," *Wireless Personal Communications*, vol. 80, no. 4, pp. 1607–1623, 2015.
19. P. Mell and T. Grance, "The nist definition of cloud computing," 2011.
20. H. T. Dinh, C. Lee, D. Niyato, and P. Wang, "A survey of mobile cloud computing: architecture, applications, and approaches," *Wireless communications and mobile computing*, vol. 13, no. 18, pp. 1587–1611, 2013.
21. S. Vaupel, G. Taentzer, J. P. Harries, R. Stroh, R. Gerlach, and M. Guckert, "Model-driven development of mobile applications allowing role-driven variants," in *Model-Driven Engineering Languages and Systems*. Springer, 2014, pp. 1–17.
22. R. T. Fielding and R. N. Taylor, "Principled design of the modern web architecture," *ACM Transactions on Internet Technology (TOIT)*, vol. 2, no. 2, pp. 115–150, 2002.
23. L. Richardson, M. Amundsen, and S. Ruby, *RESTful Web APIs*. "O'Reilly Media, Inc.", 2013.
24. C. Riva and M. Laitkorpi, "Designing web-based mobile services with rest," in *Service-Oriented Computing-ICSOC 2007 Workshops*. Springer, 2009, pp. 439–450.
25. K. Mohamed and D. Wijesekera, "Performance analysis of web services on mobile devices," *Procedia Computer Science*, vol. 10, pp. 744–751, 2012.
26. B. Upadhyaya, Y. Zou, H. Xiao, J. Ng, and A. Lau, "Migration of soap-based services to restful services," in *Web Systems Evolution (WSE), 2011 13th IEEE International Symposium on*. IEEE, 2011, pp. 105–114.
27. S. Kumari and S. K. Rath, "Performance comparison of soap and rest based web services for enterprise application integration," in *Advances in Computing, Communications and Informatics (ICACCI), 2015 International Conference on*. IEEE, 2015, pp. 1656–1660.

28. M. González, L. Cernuzzi, and O. Pastor, "A navigational role-centric model oriented web approach - moweba," *Int. J. Web Eng. Technol.*, vol. 11, no. 1, pp. 29–67, 2016. [Online]. Available: http://dx.doi.org/10.1504/IJWET.2016.075963
29. M. González, L. Cernuzzi, N. Aquino, and O. Pastor, "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*, 2016, pp. 1–11. [Online]. Available: http://dx.doi.org/10.1109/RCIS.2016.7549344
30. T. Mikkonen, R. Pitkänen, and M. Pussinen, *On the Role of Architectural Style in Model Driven Development.* Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 74–87. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-24769-2_6
31. N. Koch, A. Knapp, G. Zhang, and H. Baumeister, "Uml-based web engineering," in *Web Engineering: Modelling and Implementing Web Applications.* Springer, 2008, pp. 157–191.
32. S. Ceri, P. Fraternali, and A. Bongio, "Web modeling language (webml): a modeling language for designing web sites," *Computer Networks*, vol. 33, no. 1, pp. 137–157, 2000.
33. J. Fons, V. Pelechano, O. Pastor, P. Valderas, and V. Torres, "Applying the oows model-driven approach for developing web applications. the internet movie database case study," in *Web Engineering: Modelling and Implementing Web Applications.* Springer, 2008, pp. 65–108.
34. C. Cachero, J. Gómez, and O. Pastor, "Ooh-method: un método de diseño de lugares web," in *Conference Proceedings, IDEAS 00. Cancún*, 2000, pp. 133–144.
35. A. Adamko, "Modeling data-oriented web applications using uml," in *EUROCON 2005-The International Conference on" Computer as a Tool"*, vol. 1. IEEE, 2005, pp. 752–755.
36. M. Winckler and V. Jean, "Vanderdonckt: Towards a user-centered design of web applications based on a task model," in *In Proceedings of 5th International Workshop on Web-Oriented Software Technologies (IWWOST).* Citeseer, 2005.
37. P. A. Laplante, *Dictionary of computer science, engineering and technology.* CRC Press, 2000.
38. S. Howard, J. Hammond, and G. Lindgaard, *Human-Computer Interaction: INTERACT 97.* Springer, 2013.
39. T. Neil, *Mobile Design Pattern Gallery: UI Patterns for Smartphone Apps*, 2nd ed., I. O'Reilly Media, Ed. O'Reilly Media, Inc., 2014.
40. M. Genero, J. Cruz-Lemus, and M. Piattini, *Métodos de Investigación en Ingeniería del Software*, R.-M. Editorial, Ed. RA-MA, 2014.
41. A. Ribeiro and A. R. da Silva, "Survey on cross-platforms and languages for mobile apps," in *Quality of Information and Communications Technology (QUATIC), 2012 Eighth International Conference on the.* IEEE, 2012, pp. 255–260.
42. B. Di Martino, G. Cretella, and A. Esposito, "Methodologies for cloud portability and interoperability," in *Cloud Portability and Interoperability.* Springer, 2015, pp. 15–44. [Online]. Available: https://doi.org/10.1007/978-3-319-13701-8_2
43. E. Umuhoza and M. Brambilla, "Model driven development approaches for mobile applications: A survey," in *International Conference on Mobile Web and Information Systems.* Springer, 2016, pp. 93–107.
44. B. Kitchenham and S. Charters, "Guidelines for performing systematic literature reviews in software engineering," Keele University and University of Durham, Tech. Rep., 2007, version 2.3.
45. H. Heitkötter, T. A. Majchrzak, and H. Kuchen, "Cross-platform model-driven development of mobile applications with md 2," in *Proceedings of the 28th Annual ACM Symposium on Applied Computing.* ACM, 2013, pp. 526–533. [Online]. Available: http://hdl.handle.net/10915/26667
46. N. Chondamrongkul and N. Chondamrongkul, "Model-driven framework to support evolution of mobile applications in multi-cloud environments," *International Journal of Pervasive Computing and Communications*, vol. 12, no. 3, pp. 332–351, 2016.
47. A. H. Ranabahu, E. M. Maximilien, A. P. Sheth, and K. Thirunarayan, "A domain specific language for enterprise grade cloud-mobile hybrid applications," in *Proceedings of the compilation of the co-located workshops on DSM'11, TMC'11, AGERE! 2011, AOOPES'11, NEAT'11, & VMIL'11.* ACM, 2011, pp. 77–84.
48. D. Steiner, C. Turlea, C. Culea, and S. Selinger, "Model-driven development of cloud-connected mobile applications using dsls with xtext," in *EUROCAST (2)*, ser. Lecture Notes in Computer Science, vol. 8112. Springer, 2013, pp. 409–416.
49. A. Ruokonen, L. Pajunen, and T. Systä, "On model-driven development of mobile business processes," in *SERA.* IEEE Computer Society, 2008, pp. 59–66.
50. M. Núñez, M. González, N. Aquino, and L. Cernuzzi, *Un Enfoque MDD para el desarrollo de Aplicaciones Móviles Nativas enfocadas en la Capa de Datos*, 2017.