Analyzing the adaptability of MoWebA to different environments

Magalí González T.[1], **Luca Cernuzzi**[1]

[1]Departamento de Electrónica e Informática
Universidad Católica "Nuestra Señora de la Asunción" (UC) – Asunción – Paraguay

{mgonzalez,lcernuzz}@uc.edu.py

Abstract. *Some of the major challenges facing Web applications are those of portability, adaptability, and evolution, not only in the environment in which they run but also in the way in which they must be developed, often requiring different languages, frameworks, tools, environments, platforms, etc. MDD takes into account these issues. However, achieving portability, adaptability, and evolution depends to some extent on the degree of independence that the models adopt. This paper focuses on the Architecture Specific Model (ASM) of MoWebA (Model Oriented Web Approach). It analyses its impact on portability and adaptability across different architectures. A case study is presented to validate this issue by extending MoWebA to three different architectures. In such extensions, we analyze the grade of adaptability of MoWebA and automation of PIM-ASM, as well as the grade of independence of the PIM metamodel.*

1. Introduction

Current Web engineering methods center on developing techniques and/or models needed to define the design processes, and on providing tools to support them, following the MDD (Model Driven Development) approach in many cases [Brambilla et al. 2017]. Various quantitative and qualitative studies show how MDD practices contribute to increasing the efficiency and effectiveness in software development [Panach et al. 2021] [Farshidi et al. 2021], and others propose challenges that need to be addressed by the Web and MDE community [Bordeleau et al. 2017][Rossi et al. 2016].

The main challenges facing applications today include portability, adaptability, and evolution, not only in the environment in which they run but also in the way they need to be developed. This often requires different languages, frameworks, tools, environments, platforms, etc. MDD takes these issues into account. However, the degree of independence of the models is critical to achieving such desirable properties.

In previous studies, we proposed the MoWebA approach [González et al. 2016b] [González et al. 2016a]. Some of its features have a positive impact on the portability and adaptability problems. In particular, this study focuses on the analysis of the Architecture Specific Model (ASM), which corresponds to a level of abstraction between the Platform Independent Model (PIM) and the Platform Specific Model (PSM), encapsulating details related to the architecture the developed software will have. This encapsulation makes it possible to improve both the generalization of the modeling and its understanding (by separating concepts). Therefore, it keeps the portability of the PIM regarding the different architectures (e.g., RIA, SOA, Mobile) and facilitates the adaptability and evolution of the method to different architectures.

Different extensions of MoWebA have been presented for various architectures: MoWebA4RIA (extension of MoWebA for RIA functionalities) [Nuñez et al. 2018], MoWebAMobile4FC (extension of MoWebA for mobile applications for functions in the cloud) [Sanchiz et al. 2018], and MoWebAMobile4Persistence (Extension of MoWebA in mobile applications for the persistence layer) [Núñez et al. 2020]. MoWebA prescribes the definition of a new ASM for each extension, along with the respective PIM-ASM transformation rules and ASM-Code generation. Furthermore, each extension presented the validation of the proposal.

This study enriches the previous works by presenting a case study to analyze the role of the ASM phase, both for modeling and transformation processes, based on the three previous experiences. The aim is to determine the degree of adaptability and evolution of the MoWebA method, including the automation of PIM-ASM, as well as the grade of independence of the PIM metamodel, concerning the three different architectures.

The main findings are as follows. All three extensions were successfully developed from the same PIM metamodel with a reduced number of adaptations. Each extension has involved revising the elements defined in the PIM Metamodel and defining the ASM Metamodel for a specific architecture. The ASM-PIM transformation has been completed for the ASM elements that are derived from the PIM by inheritance, with a few manual adjustments.

The rest of the paper is structured as follows. Section 2 presents the related works. Section 3 provides a brief introduction to MoWebA, with some details on the ASM model. Section 4 focuses on the analysis of MoWebA's adaptability to different architectures. Finally, Section 5 concludes and outlines future work.

2. Related Works

The Model Driven Development approach provides various benefits in terms of portability, adaptability, and evolution [Brambilla et al. 2017]. Thus, several web methodologies follow the Model Driven Web Engineering approach [Rossi et al. 2016].

Moreover, several methodologies such as WebML and UWE propose extensions to consider emerging technologies and architectures. Such extensions are generally contemplated with new constructors that are introduced in the proposal's notation by UML stereotypes, or specific DSL included in the original notation for modeling so that the original proposal becomes an enriched one for the extension under consideration. In other words, the adaptation mechanism is based on the inclusion of notational elements in the models, thus reducing their independence from the platform or architecture. This is because the extensions are not defined as a separate modeling layer. The result of this extension mechanism is that the platform-independent modeling (PIM) stage contemplates elements of a specific platform or architecture, combining both concepts in a single final model. Other proposals decided to add architectural-specific information at the PSM level. In this case, architecture and platform details are included at the same modeling level, losing portability at the architectural level for different technologies where it can be implemented (e.g., UWA for RIA [Bernardi et al. 2014]). However, the WebRatio Mobile Platform [Brambilla et al. 2014], an extension of WebML, is the only one that considers an ASM to some extent. None of the others capture the requirements for specific architectures at a different level of abstraction, as proposed by MoWebA. As a counterpart, to offer

a greater reusability of the PIM facilitating the architectural evolution of the Web applications, this mechanism of extension requires some additional effort, including the need for metamodels and the definition of the corresponding transformation rules, to achieve automatic transformations on the proposed architecture or platform. But with this separation, a clear distinction is made between what would be the problem space, presenting a model that is completely independent of the target architecture or platform; and the solution space, through the specific extension oriented to architecture, platform, and the final code.

For a more detailed analysis and discussion of the role of ASM in the portability, adaptability, and evolution of the methodology, we refer the interested reader to section 2.2.1 of [González 2022].

3. MoWebA at a Glance

Figure 1 presents the dimensions of MoWebA that cover its modeling and transformation processes. MoWebA adopts the MDA approach by identifying three different phases related to modeling and transformation activities (the *Phases* dimension): i) the problem space, covered by CIM (Computational-Independent Model), and PIM; ii) the solution space, covered by ASM, and PSM; and, iii) the source code definition, covered by the ISM (Implementation-Specific Model) and manual code adjustments.

The *Levels* dimension deals with complementary perspectives to be considered in every phase (content, business logic, navigation, presentation, users). Finally, the *Aspects* dimension addresses structural and behavioral considerations for each perspective [González et al. 2016b].

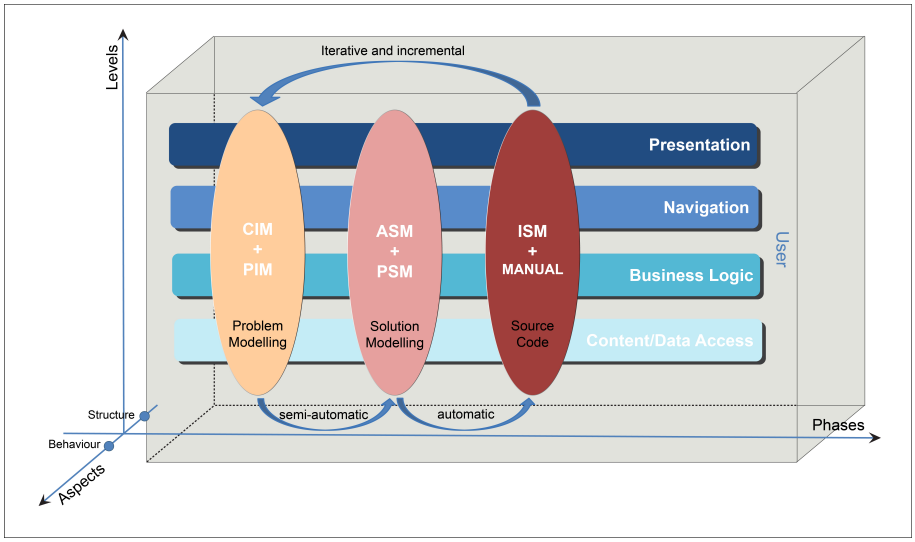

Figure 1. Dimensions of MoWebA that cover its modeling and transformation processes.

To develop applications, MoWebA defines two main complementary processes: the first related to modeling activities and the second one related to transformation activities. To formalize the modeling and transformation processes, it adopts the MOF language for the definition of the abstract syntax, and the UML profile extension for a precise definition of the modeling language (i.e., concrete syntax).

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

In this line of thought, MoWebA can be extended to other architectures through the definition of the ASM, used to complement the PIM with information about a specific architecture. To utilize an ASM model, it must first be defined. This involves specifying the corresponding metamodel, among other steps. The ASM metamodel definition follows the recommendation of Brambilla et al. [Brambilla et al. 2017]. Furthermore, MoWebA complements the suggested process with additional steps that go from the definition of the concrete syntax until the generation of the final code of an application [González et al. 2016a]. The steps of this process are synthesized below:

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

Indeed, the MoWebA approach requires some additional effort as a counterpart to improving the portability of the PIM and facilitating the architectural evolution of applications. This additional effort includes the need for the specification of ASM metamodels and the definition of the corresponding transformation rules, to achieve automatic transformations on the proposed architecture. In any case, it should be noted that these steps will only be executed once when targeting a new architecture for the first time.

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

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

Figure 2 illustrates the proposed process for the development of applications for a specific architecture. In the figure, we can see that the process begins with the PIM definition with the MagicDraw tool (http://www.nomagic.com/products/magicdraw.html) using the corresponding UML profiles. The PIM model is then exported to an XMI file, which is imported into the EMF (https://www.eclipse.org/modeling/emf/) tool, where, through the mapping of the defined M2M transformation rules, it is transformed into a new XMI with diagrams corresponding to the ASM. The new XMI is imported into the Acceleo tool. The latter uses the transformation rules to perform transformations from

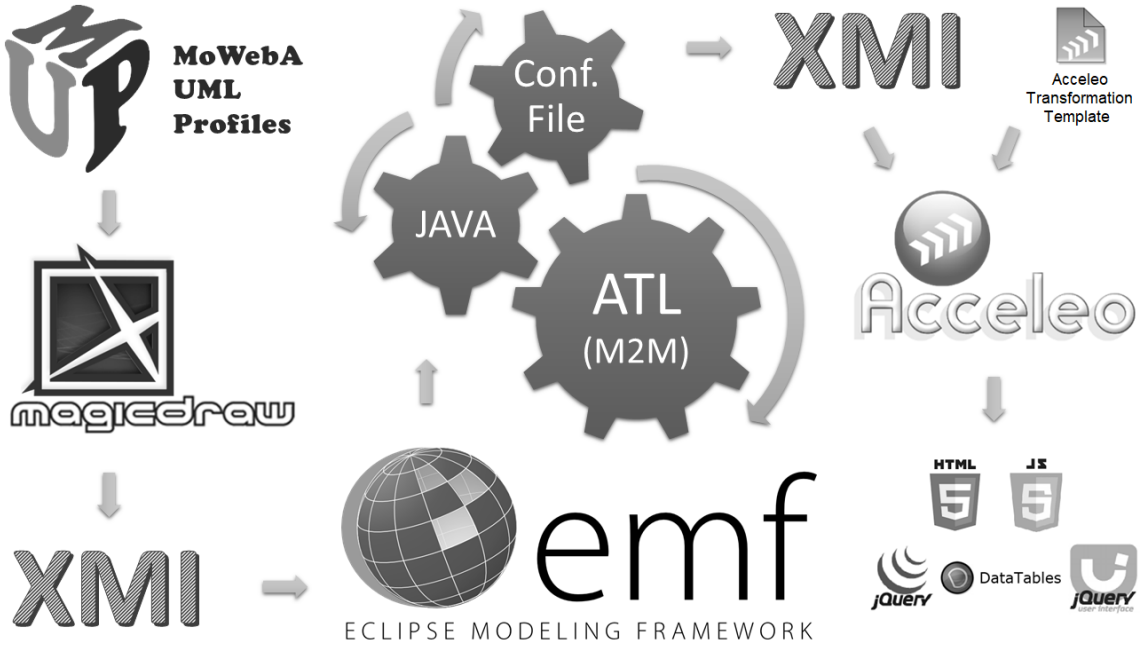

Figure 2. MoWebA development process

model to text, generating the code in different languages depending on the architecture specified.

We claim that the ASM extension mechanism proposed by MoWebA can improve the adaptability and evolution of MoWebA. To support this assertion, we present below a case study that aims to analyze this aspect.

4. Analyzing the Extension of MoWebA to other architectures

The focus of the study is on the ASM phase, both for modeling and transformation processes. The experience was structured taking into account a framework that Runeson et al. [Runeson et al. 2012] have defined for case studies. The case study was done as part of a research project at the Catholic University of Asuncion called "Mejorando el Proceso de Desarrollo de Software: Una propuesta basada en MDD" (https://www.dei.uc.edu.py/proyectos/mddplus/), Paraguay.

In this experience, the extensions were made to three different architectures: MoWebA4RIA (extension of MoWebA for RIA functionalities) [Nuñez et al. 2018], MoWebAMobile4FC (extension of MoWebA for mobile applications for functions in the cloud) [Sanchiz et al. 2018], and MoWebAMobile4Persistence (Extension of MoWebA in mobile applications for the persistence layer) [Núñez et al. 2020].

Using the GQM [Basili et al. 1994] template, the main purpose of this validation was described as follows: "**Analyze** the MoWebA method **for the purpose of** determining the grade of adaptability and evolution of the method **with respect to** the architecture **from the point of** view of the researcher **in the context of** a research project with students, professionals, trainee researchers, and MDD experts".

Based on this general objective, we propose the following research questions:

- **RQ1:** To what extent are the evolution and adaptability of the extension mecha-

nism proposed by MoWebA achieved to incorporate new architectures?
- **RQ2:** How independent is the MoWebA PIM for use in the modeling stage before ASM?
- **RQ3:** To what extent automation can be obtained with MoWebA's model-to-model and model-to-code transformation rules?

This study was developed by a research team composed of 11 participants (five undergraduate students, one master's degree student, two PhD students, and three MDD PhD experts). The final-year undergraduate students involved in the project knew MDE and MDD, in addition to previous knowledge and experience with the MoWebA proposal. One of the undergraduate students was in charge of performing the PIM-ASM mappings and developing the M2M transformation rules from PIM to ASM and the others have worked independently in the definition of the metamodels and the code generation rules for each extension.

4.1. Design of validation

The follow-up of the development of the extensions was accompanied by weekly meetings of approximately 2 hours duration. The weekly meetings were held for one year (including a series of validation experiences for each extensions that are not included in this paper for reason of space). The meetings were oriented to follow the experiences done in the period, complemented with open interviews, observations, and focus groups.

The extension proposals have been divided into stages, considering the process proposed by MoWebA for ASM extension and other additional activities. The activities carried out for the development of the extensions are: i) Definition of the scope to be considered for extensions; ii) Revision and adjustments to the PIM taking into account the architecture established for the extension; iii) ASM metamodeling using MOF; iv) PIM-ASM mapping; v) PIM-ASM transformation rules definition with ATL; vi) Definition of code generation rules; vii) Proof of Concept development; viii) Validation of the extension. For each activity, we identified sources of data to be considered during the experience analysis stage.

In the following subsections, we will present a synthesis of the metamodels proposed for the three architectures, and then focus on items (iv), (v), (vi), and (viii). The details of each of the extensions can be found in the following articles [Nuñez et al. 2018] [Sanchiz et al. 2018] [Núñez et al. 2020].

4.1.1. ASM Metamodel and UML Profile Definition

The **MoWebA4Ria** metamodel incorporates the functionalities to model the features of client data, client business logic, and asynchronous communication between client and server. To save data in a web client, specifically in a web browser, there were created two concepts, a *ClientValueObject*, which extends the value object of the original MoWebA logic diagram, and a *ClientStaticObject*, which extends a static object, a new element introduced in the logic diagram to represent sets of statically defined values as properties of the class. In terms of processes executed on the client, there were created three elements considered in the presentation page, the *RichForm*, the *RichTextInput*, and the *RichTable*.

To allow asynchronous communications between client and server there were created the element called *AsynchronousCall*.

The **MoWebAMobile4FC** metamodel extends the logic diagram of MoWebA to enable the definition of logic processes. Network communication is viewed as a logic process. The logic diagram comprises the logic processes (i.e., *TProcesses* and the procedures for doing a specific task (i.e., *Services*). Also, the logic diagram has *ValueObjects*, which group attributes of entities and enable access to the entities' data. The extensions for obtaining the ASM are based on the REST architecture and the four types of network communication functions: CloudServer, ResourceInterface, RestProcess, and Request type (lightdata, download, upload, loadImage).

The **MoWebAMobile4Persistence** metamodel focuses on the data access level of mobile applications. The definition of the mobile ASM for persistence started with the extension of MoWebA's Entity Diagram at the PIM to take advantage of conceptual elements of the structural type. In this extension, we can define mobile applications with local data persistence and data provider components. Regarding data persistence, we introduce elements to identify what data will be stored on the device (*persistentEntity* tag value), and what type of persistence will be used (*persistentType* tag value). Concerning data providers, we introduce the following interfaces: *WebServiceInterface*, *HardwareDeviceInterface*, and *MobileAppDataInterface*, for the representation of external providers (via web-services), internal providers (through commonly supported sensors among platforms) and interoperability with other applications (data types defined for communication between applications), respectively.

4.1.2. PIM-ASM Mapping Rules

A visual analysis of the defined profile was used to perform the mapping. During the mapping phase, we noticed that, in general, when the relation between elements from source and target profiles is an inheritance, the transformation that allows us to obtain the corresponding target element is very simple and consists of the application of the correct stereotype.

In **MoWeba4Ria** mapping, we identified the following mapping: i) A *Table* element of the PIM, becomes a *RichTable* in the ASM; ii) A *Form* element of the PIM, becomes a *RichForm*; iii) A *TextInput* of the PIM, is transformed into a *RichTextInput*; iv) A *StaticObject* element of the PIM, becomes a *ClientStaticObject*; v) A *ValueObject* of the PIM, is transformed into a *RichValueObject*.

In **MoWebaMobile4FC** mapping, we identified the following mappings: *tProcess* from the PIM becomes a *restProcess* in the ASM, a *service* from the PIM becomes *cloudService* and a *valueObject* from the PIM becomes a *cloudValueObject*.

In **MoWebAMobile4Persistence** mapping, we could detect that a class Entity in the PIM model, becomes a *PersistentEntity* class in the ASM model and that property *entityProperty* in the PIM model is transformed into a *persistentEntityProperty* in the ASM model.

When the relation between two elements in one profile is not an inheritance, the mapping becomes a bit more difficult. In the case of the RIA profile, we found two rela-

tions of this type, one between the *ServiceState* and *AsynchronousCall* elements, and the other between the *ServiceState* and *RichTable* elements. Analyzing these relationships, we could conclude that the *AsynchronousCall* must be created when it is related to a *ServiceState*, which, in addition, must represent an asynchronous service. This condition can be detected automatically, allowing the automation of the transformation.

4.1.3. PIM to ASM Transformation Rules

For the definition of the M2M transformation rules, we have chosen ATL (http://www.eclipse.org/atl/) (Atlas Transformation Language) over QVT (http://www.omg.org/spec/QVT/About-QVT/). This choice was based, above all, on the fact that ATL is considered one of the most widely used transformation languages, both in academia and industry, and a mature tool support is available [Brambilla et al. 2017].

Another significant choice was the selection of the engine execution mode of the ATL transformation, which has two modes of execution: default and refining [García Rubio et al. 2013]. When the source and target metamodels are different, it is mandatory to use the default execution mode, but when the metamodels of the source and target models are the same you can opt for either of the two execution modes [García Rubio et al. 2013]. In our case, both, the metamodels of source and target models are the same because MoWebA's implementation is based on profiles [González et al. 2016b]. Moreover, the M2M transformation from PIM to ASM fits conceptually better to the refining mode, since the PIM is justly refined to obtain the ASM. For all this, we opted for the refining execution mode. This choice has greatly reduced the number of necessary transformation rules. However, this also led to some complications, since the application of profiles in ATL is performed in the imperative block and turns out this option when using refining mode. For the above reason, we were forced to change the compiler. The default compiler in Eclipse is the EMF-specific Virtual Machine (EMF-specific VM), but it also provides other compilers. We opted for the EMF Transformation Virtual Machine (EMFTVM). Although the main reason for this choice was that it allows the use of the imperative block in the refining mode, which is necessary to work with profiles, there are some other advantages. For example, its performance is roughly 80% better than the EMF-specific Virtual Machine, and allows us to invoke native Java methods.

Another important aspect of the transformation rules definition is the Configuration file. Configuration files allow the designer to control some transformation aspects, allowing the achievement of two important capabilities: on the one hand, the possibility of capturing specific design decisions of the system under design that would otherwise not be automated, and on the other hand, the possibility that the designer has some level of influence over the transformation rules. Two different configuration files have been considered. One that allows to indicate which elements of the model are transformed and which are not, called "ArchConfTransformacion.yaml", and another one that allows to specify some properties for the classes created automatically when executing the M2M transformation rules, called "ArchConfAsynchronousCall.yaml".

Among the challenges of the M2M transformation that have been tackled, we can mention:

1. The inability to work with profiles in the refining mode, forced us to use a compiler different from the one proposed by default.
2. The asynchronous services identification, is necessary for automating the creation of asynchronous classes.
3. The correct configuration of the IDE to access native Java methods was created expressly for the processing of the configuration files.

The *called rules* is another vital section of transformation rules. The called rule named *applyMobileStereoTypes* allows changing the stereotypes in the target model.

4.1.4. ASM to Code Generation

After the generation of the ASM of the application, it is possible to go through the process of generating code. To do this, we have defined transformation rules from model to text using the Acceleo tool for the three extensions.

These transformation rules follow a template-based approach in which text templates are specified with entries for data to be extracted from the model diagrams. The MTL (http://www.omg.org/spec/MOFM2T/1.0/) language was used for the definition of the templates, and OCL to make queries to the model. In addition, services defined in Java were used to extend MTL with greater functionalities.

In the case of **MoWebA4RIA**, these rules are responsible for transforming elements defined in the logic and content diagrams to HTML5, Javascript, jQuery, jQuery code, and Datatables libraries.

In **MoWebAMobile4FC** we have built a service in Java to extend the functions of the MTL. We have built the transformation rules based on the classes, properties, and operations characterized by the respective stereotypes, tag values, and enumerations defined in the ASM's profile. In this sense, such rules perform a mapping between the model elements defined and the target code to be generated. The generation for both sides, mobile and cloud, depends on each combination of a *CloudServer*, a *RestProcess*, *ResourceInterface* and *CloudRequestHandler*. The target code generated consists, on one side, of native mobile code written in Java for Android, in Swift for iOS, and on the other side, in open source code written in Javascript with Node.js for the Openshift and Amazon Web Services platforms. Additionally, our cloud 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 for easing the portability of cloud applications.

In **MoWebAMobile4Persistence**, the generated target codes for Android and Windows Phone are Java and C#, respectively. Additionally, GUI code is also generated for Android (XML) and Windows Phone (XAML). This generated code is ready to be executed for both mobile platforms, Android and Windows Phone, with previous compilation in their respective IDEs.

The tools defined for the three MoWebA extensions, which include the metamodels, UML profiles, PIM-ASM transformation rules, and ASM-code transformation rules, are available on the *MDD+ project* website (http://www.dei.uc.edu.py/proyectos/mddplus/herramientas/).

4.2. Data Collection

The data collection can be classified as first-degree since we were in direct contact with participants and we collected data in real-time using different methods. The main sources of data were results/outputs expected for each stage.

We considered as **information sources** for data collection:

1. the *project documentation*, which includes PIM metamodels, ASM metamodels, PIM-ASM transformation rules, ASM-Code generation, proof of concept, and all the documentation generated for each validation extension;
2. the work sessions' *timesheets* of each validation extension.

The **quantitative data** were collected from these information sources. On the other hand, the **qualitative data** were obtained from the comments and opinions of the participants.

In the first place, the **project documentation** allowed us to determine the success rate of the participants. Therefore, the **timesheets** permitted us to determine the completion time of each process in the validation extensions. Table 1. presents a summary of data collected during the experience.

Table 1. Summary of data collected during the experience.

Data Collection method	**Materials**	**Outputs**
1. Definition of the scope to be considered for extensions.		
Focus Group, Open, and semi-structured interviews	Scientific documentation (articles, proceedings, books)	Theoretical framework and state of the art related to each architecture; Document with scope definition.
2. Revision and adjustments to the PIM taking into account the architecture established for the extension.		
Observation (category 3); Archival data	Magic Draw; MoWebA specification; Internet	PIM metamodel modified; PIM modification report.
3. Metamodel and UML profile: definition of architecture metamodels, considering the Brambilla et al. framework [Brambilla et al. 2017]. Definition of UML profiles.		
Observation (category 3); Archival data	Magic Draw; Brambilla specification; UML profile specification	MOF definition for each architecture; UML profile definition for each architecture; Explanatory report
4. PIM-ASM mapping.		
Observation (category 3); Archival data	Magic Draw	Mapping rules identification
5. PIM-ASM transformation rules definition with ATL.		
Observation (category 3); Archival data	ATL tool	Transformation rules in ATL; Explanatory report
6. Definition of transformation rules from the ASM model to the final code. Generation of source code.		
Observation (category 3); Archival data	Acceleo tool	Transformation definition in Acceleo; Documentation

Table 1. Summary of data collected during the experience.

Data Collection method	Materials	Outputs
7. Proof of Concept development.		
Focus group; Open and semi-structured interviews	MagicDraw; ATL tool; Acceleo	PIM Models; ASM models, code generated

4.3. Data Analysis

The analysis carried out consists of a qualitative analysis and judgment for each research question, based on the data collected and the monitoring throughout the process. We begin the section with table 2, which presents a summary of the activities carried out and the achievements reached for each of the extensions.

Table 2. Resume of activities of ASM extensions development

Task	RIA	Mobile for Function and the Cloud	Mobile for Persistence
Scope for the extensions	client data, client business logic, and asynchronous communication between client and server	light-data, load-image, download-files, upload-files	data persistence mechanisms: databases, files, and key-value pairs providers: external, internal, and other applications
Modifications to the PIM metamodel	Logic Diagram: *staticObject* and *value* attribute. Content Diagram: *requestType* attribute of Form element, *name* attribute of List element, *submitButton* as a specialization of Button	none	Entity Diagram: *EntityProperty* element and *Datatype*
PIM Diagrams extended	Logic and Content Diagrams	Logic Diagram	Entity Diagram
PIM elements extended into the ASM metamodel	6 elements: *valueObject* and *staticObject* into the Client Data; *table*, *form* and *textInput* into the Client Business Logic; and *service* in the Asynchronous Communication)	3 elements: *valueObject*, *service* and *TProcess*	2 elements: *persistenceEntity* and *persistenceEntityProperty*
Number of elements defined in the ASM metamodel	11 elements: 3 for Client Data, 4 for Client Business Logic and 4 for Asynchronous Communication	18 elements: for Mobile Cloud Communications	17 elements: 5 for data persistence y 12 for data providers

Table 2. Resume of activities of ASM extensions development

Task	RIA	Mobile for Function and the Cloud	Mobile for Persistence
PIM-ASM Mapping	5 direct mapping elements from inheritance and 1 mapping from associations	3 direct mapping elements from inheritance	2 direct mapping elements from inheritance
PIM-ASM Transformation	ATL with refining mode and 2 configuration files	manually	ATL with refining mode and 1 configuration file
M2T Code Generation	**M2T Tool:** Acceleo **Final code:** HTML5, Javascript, jQuery, jQuery UI, and Datatables libraries	**M2T Tool:** Acceleo **Final Code:** Java (Android), Swift (iOS), Node.js , Docker	**M2T Tool:** Acceleo **Final Code:** Java (Android) and C# (Windows), GUI code in XML (Android) and XAML (Windows)

Below we discuss each of the research questions.

RQ1: To what extent are the evolution and adaptability of the extension mechanism proposed by MoWebA achieved to incorporate new architectures?

From the experience with the three different extensions, we can conduct the following analysis:

- All three extensions have been successfully developed, starting from the same PIM metamodel, with a reduced number of adjustments to the already defined PIM level. However, we had to make some decisions for the development of the extensions, which have involved some minor adjustments to the PIM metamodel, and changes in the environments to be used for the definition of the transformation rules (e.g. the ATL tool used for M2M transformations explained in section 4.1.3).
- The MoWebA proposal for adaptation to other architectures can be done by users with knowledge of MDD and the use of standards. This observation is made because the extensions have been developed by undergraduate students whose knowledge and experiences have been the fundamentals, tools, and standards of MDD. They have learned the MoWebA proposal during their degree studies and as part of the work carried out in the project.
- One aspect to consider is that the greater the number of elements of the ASM metamodel that are not related to elements of the PIM metamodel (either through inheritance or association), the degree of PIM-ASM automation decreases. Therefore, more effort is required in manual adjustments to the ASM model before code generation. However, the configuration files have allowed to introduction of design decisions before the transformation processes and thus increased the degree of automation.
- Finally, regarding the number of elements added in the ASM metamodels, we consider that it corresponds to a reasonable and manageable amount to include

new concepts in a methodological proposal (11 for RIA, 18 for Cloud Communications, and 17 for persistence). On the other hand, the fact of considering the modeling of a specific architecture (ASM) in a different level of abstraction is not mandatory and should be included only in case the architecture must be specified in the modeling process. Doing so, the PIM remains independent of the proposed extensions.

RQ2: How independent is the MoWebA PIM for use in the modeling stage before ASM?

The extensions made to MoWebA have involved revisions to the elements defined in the PIM metamodel and the definition of the ASM metamodel for a specific architecture. In some cases, these revisions implied a more detailed specification of certain existing elements, and in other cases the inclusion of new no contemplated elements into the PIM. In all three cases, the extensions were made based on diagrams already defined in the PIM, since each extension aims to consider aspects of an architecture that includes one or more layers of an application. It should be noted that for the three metamodels defined, there are elements that correspond to specializations of elements from the PIM (e.g. *richForm* is a specialization of *form* in the RIA metamodel, *CloudService* as a specialization of *Service* in the FC extension, or *persistenceEntity* as a specialization of *entity* in the persistence ASM).

Table 2 shows some details of how the extensions were made. From this Table we have conducted the following analysis:

- The first has to do with the extended diagrams. In this regard, it should be noted that the three extensions have been based on diagrams already existing in the PIM, which leads us to believe that the PIM has the necessary independent modeling elements to carry out these extensions.
- Another aspect worth mentioning is that specializations of existing concepts have also been made in the three extensions, i.e., elements have been redefined or specialized to orient the elements towards specific architectures. It gives us evidence that the generic or independent concepts are included in the PIM metamodel.
- In two of the three extensions, new elements have been proposed, or, existing elements have been redefined in the PIM metamodels. This is because, in addition to being required for the extensions, they have been considered as generic concepts, i.e., independent of the architecture, so we decided to include them in the PIM metamodels (e.g., the *staticObjects* identified in RIA, or *EntityProperty* added to the PIM during the persistence extension). It should be noted, however, that the number of new elements included in the PIM has been minimal (2 classes and 3 attributes in RIA, no elements in Cloud, 1 class and 1 enumeration in Persistence), which again confirms the fact that the PIM is independent enough to be extended, but at the same time has the necessary elements to PIM modeling.

RQ3: To what extent automation can be obtained with MoWebA's model-to-model and model-to-code transformation rules?

ASM-PIM transformation has been completed for ASM elements derived from PIM by inheritance.

In cases where additional information has been required from the user to include

design decisions, the configuration files have allowed to enter such information before performing the transformations, thus improving the degree of automation.

The PIM-ASM transformation experiences described in [Núñez et al. 2020] have shown that although it is possible to include manual adjustments to the ASM models generated from the PIM, the configuration files have helped to reduce these percentages considerably.

It should also be noted that the following manual adjustments have been made by using the extensions: i) including certain elements in the ASM that could not be derived from the PIM; ii) modifying the final code related to the user interface of the applications and other adjustments for the generation of the application from the code generated in Acceleo; iii) modifying the final code and the models for validation purposes.

5. Conclusions and Future Works

The purpose of this study was to analyze the role of the Architecture Specific Model (ASM) phase in both modeling and transformation processes. Specifically, we examined its use in extending MoWebA for three different architectures: MoWebA4RIA (an extension for RIA functionalities), MoWebAMobile4FC (an extension for mobile applications with functions in the cloud), and MoWebAMobile4Persistence (an extension for mobile applications with a persistence layer). MoWebA prescribes the definition of a new ASM for each extension, along with the respective PIM-ASM transformation rules and ASM-Code generation. The development of the extensions was followed up through weekly meetings over one year. These meetings were complemented by open interviews, observations, and focus groups.

The results show that MoWebA provides an interesting degree of adaptability and automation to the PIM-ASM, as well as independence from the PIM meta model. Experience has shown that all three extensions were successfully developed from the same PIM metamodel with a reduced number of adaptations. Each extension has involved revising the elements defined in the PIM metamodel and defining the ASM metamodel for a specific architecture. Two of the extensions have included the PIM-ASM transformation rules with configuration files. While in the other extensions (i.e., (MoWebAMobile4FC), the PIM-ASM transformation required a few manual adjustments.

We identified several opportunities to improve and extend the proposal. In the modeling dimension, it is relevant the development of a modeling tool to simplify the use of MoWebA for PIM and ASM modeling, along with the revision and improvement of Model-to-Model and Model-to-Code transformation rules. Considering the need for adaptation of the methodologies, we envision the extensions to other environments and architectures. Finally, usability experiences with MoWebA and further validation of the proposal (e.g., formal experiments or case studies) in industrial or commercial contexts are necessary steps to consolidate the approach.

References

Basili, V. R., Caldiera, G., and Rombach, D. H. (1994). *The Goal Question Metric Approach*, volume I. John Wiley & Sons.

Bernardi, M. L., Lucca, G. A. D., and Distante, D. (2014). Model-driven fast prototyping of rias: From conceptual models to running applications. In *2014 International*

Conference on Advances in Computing, Communications and Informatics (ICACCI), pages 250–258.

Bordeleau, F., Liebel, G., Raschke, A., Stieglbauer, G., and Tichy, M. (2017). Challenges and research directions for successfully applying MBE tools in practice. In *Proceedings of 20th International Conference on Model Driven Engineering Languages and Systems (MODELS 2017), Austin, TX, USA, September, 17, 2017*, pages 338–343.

Brambilla, M., Cabot, J., and Wimmer, M. (2017). *Model-Driven Software Engineering in Practice, Second Edition*. Synthesis Lectures on Software Engineering. Morgan & Claypool Publishers.

Brambilla, M., Mauri, A., and Umuhoza, E. (2014). 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*, pages 176–191.

Farshidi, S., Jansen, S., and Fortuin, S. (2021). Model-driven development platform selection: four industry case studies. *Software and Systems Modeling*, pages 1–27.

García Rubio, F. O., Vara Mesa, J. M., and Chicote, C. V. (2013). *Desarrollo de Software Dirigido por Modelos: Conceptos, Métodos y Herramientas*. Ra-Ma Editorial.

González, M., Cernuzzi, L., Aquino, N., and Pastor, O. (2016a). 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*, pages 1–11. IEEE.

González, M., Cernuzzi, L., and Pastor, O. (2016b). A navigational role-centric model oriented web approach - MoWebA. *Int. J. Web Eng. Technol.*, 11(1):29–67.

González, M. (2022). *A Navigational Role-Centric Model Oriented Web Approach MoWebA*. Phd thesis, Polictecnic University of Valencia, Valencia, Spain.

Nuñez, G., Bonhaure, D., González, M., Aquino, N., and Cernuzzi, L. (2018). A model-driven approach to develop rich web applications. *CLEI Electron. J.*, 21(2).

Núñez, M., Bonhaure, D., González, M., and Cernuzzi, L. (2020). A model-driven approach for the development of native mobile applications focusing on the data layer. *J. Syst. Softw.*, 161.

Panach, J. I., Dieste, O., Marín, B., España, S., Vegas, S., Pastor, O., and Juristo, N. (2021). Evaluating model-driven development claims with respect to quality: A family of experiments. *IEEE Trans. Software Eng.*, 47(1):130–145.

Rossi, G., Urbieta, M., Distante, D., Rivero, J. M., and Firmenich, S. (2016). 25 years of model-driven web engineering: What we achieved, what is missing. *CLEI Electron. J.*, 19(3):1.

Runeson, P., Höst, M., Rainer, A., and Regnell, B. (2012). *Case Study Research in Software Engineering - Guidelines and Examples*. Wiley.

Sanchiz, E., González, M., Aquino, N., and Cernuzzi, L. (2018). Moweba mobile: Modeling and generation of the communication of mobile apps with their functions in the cloud. In *Proceedings of the XXI Iberoamerican Conference on Software Engineering, Bogota, Colombia, April 23-27, 2018*, pages 312–325.