Generated Psm Multi-Layered Model Using Mda Approach

Aziz Srai, Fatima Guerouate, Hilal Drissi Lahsini

Abstract*: Model-driven engineering is a paradigm in which modeling is considered the central element of software. This paradigm puts the model at the center of the concerns of designers and analysts. In that sense, several specifications have been proposed such as the MDA approach proposed by the OMG. MDA (Model-Driven Architecture) is a standard, launched by the OMG (OMG, 2003), which is based on the MDE (Model Driven Engineering), providing a set of guidelines and architecture for the design of software systems. The MDA approach provides the opportunity to understand complex systems and the real world through an abstraction of them. This abstract point of view of the system is elaborated in a conceptual framework as well as a number of standards provided by the OMG allowing defining the Models, their relations as well as their transformations, e.g. UML (Unified Modeling Language), MOF and XMI (XML Metadata Interchange). After a detailed description of the standards on which MDA is based, such as MOF, QVT. Our effort will be focused on developing a meta-model respecting the n-tier architecture and on the definition of transformation rules allowing automatic conversion from UML models to n-tier models.

***Index Terms*: MDA, QVT, UML, MODELS.**

I. INTRODUCTION

In software engineering, several software development approaches, essentially procedural, object, components and services, have been successfully applied. The common goal of these approaches was to facilitate the implementation of computer systems on execution platforms while ensuring the quality of the systems produced. These execution platforms do not stop to evolve, to diversify and to become more complex. In order to separate the business part from the specific part of the execution platform, a rise in power models has been proposed, which means models are not just used to represent and document systems but have become productive which means they are used to generate the code of the systems. It is the essence of the new approach called Model Driven Engineering (MDE). As a consequence, Model Driven Engineering is a specific software engineering approach which invites to center the software development on the models. It defines a theoretical framework for generating code using successive transformations of models. In this context, an important approach of Model Driven Engineering has emerged, it's the MDA approach (Model-Driven-Architecture) of the OMG (Object Management Group). MDA is a particular variant of MDE which is proposed by the OMG in 2001. It concerns an approach for using models in software development.

In this work we try to apply the MDA approach to generate a multi-layered application composed of three layers, a presentation, a business layer, and a data access layer.

This paper is organized as follows: we begin in the first section with an introduction. The section 2 is dedicated to the related work. The section 3 introduces MDA architecture. The section 4 introduces mobile platforms. The section 5 presents uml and n-tier meta-models and the different transformation algorithms for generating the n-tier model. The last section concludes this paper and presents some perspectives.

II. RELATED WORKS

The authors in [4] propose an application of the MDA approach on mobile platforms, to generate as a final result a model that respect the n-tier architecture. The authors also propose in this work the necessary meta-models to generate the presentation layer, the business layer, and the data access layer. The different model-to-model transformation rules have been defined. It remains to be noted that, the authors chose the ATL model transformation language to translate these transformation rules. But the transformation algorithm with the ATL language was not mentioned.

In the work [1], the authors propose the application of the MDA approach on web services platforms. Authors have proposed effectively a meta-model that represents the web service platform, a meta-model that represents UML. The different rules have been cited using the ATL language.

The authors in [2] propose an algorithm based on the QVT (Query / View / Transformation) model transformation language, in order to generate a PSM model respecting the multi-tier architecture dedicated to E-learning platforms.

Recently, the work [3] is arrived to generate a PSM model for EJB platforms.

Manuscript published on 30 June 2019.
***** Correspondence Author (s)
Aziz Srai, LASTIMI Laboratory, Superior School of Technologies of Sale, Mohammadia School of engineering, Mohamed V University city of Rabat, Morocco.
Fatima Guerouate, LASTIMI Laboratory, Superior School of Technologies of Sale, Mohammadia School of engineering, Mohamed V University city of Rabat, Morocco.
Hilal Drissi Lahsini, LASTIMI Laboratory, Superior School of Technologies of Sale, Mohammadia School of engineering, Mohamed V University city of Rabat, Morocco.

III. MODEL DRIVEN ENGINEERING (MDE)

Model Driven Engineering (MDE), has allowed several significant improvements in the development of complex systems by allowing focusing on a more abstract concern than conventional programming. It is a form of generative engineering in which all or part of an application is generated from models.

A. Model Driven Architecture

MDA (Model Driven Architecture) is a particular variant of the MDE which is proposed by the OMG in 2001. This is an approach for using models in software development.

MDA's three main goals are portability, interoperability and reusability through separation of concerns. The MDA starts from the idea of separating the specification of a system from the details of its implementation on a specific platform and provides an approach to:

- Specify a system regardless of the platform that supports it.
- Specify platforms.
- Choose a particular platform for a system.
- Transform the specification of a system into a new specification specific to the chosen platform.

The techniques used are mainly modeling techniques and model transformation techniques. Transformations can be horizontal (PIM to PIM or PSM to PSM) or vertical (PIM to PSM or PSM to PIM).

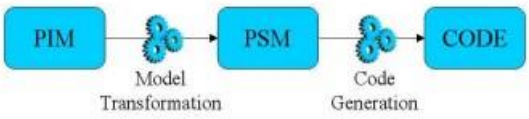

Fig. 1. MDA process

- CIM: Expression of needs by a model independent of the design.
- PIM: In MDA, a Platform Independent Model (PIM) is a model of a high level of abstraction which does not take into account aspects related to implementation technologies.
- PSM: The Platform Specific Model (PSM) is a model that describes the technical solution to achieve. It is used to translate the PIM by explaining the implemented technologies.

B. Meta-Object Facility (MOF)

With the development and the evolution of a wide variety of different and incompatible meta-models, there was an urgent need to provide an integration framework for all meta-models. The answer was to offer a definition language for meta-models or a meta-meta-model. The MOF is the metamodel of the OMG that allows establishing meta-models allowing themselves to express models. The MOF standard is at the top of a four-layer modeling architecture:

- M3, the meta-meta-model MOF (self-descriptive layer).
- M2, the meta-models.
- M1, models.
- M0, the real world.

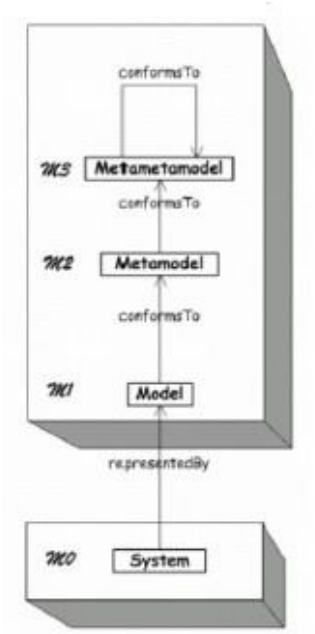

Fig. 2. 3 + 1 adapted architecture (Bézivin, 2005)

The meta-meta-model of the highest level (level M3) is the MOF. Meta-models are therefore compliant with MOF and belonging to the M2 level. The M2 level defines the modeling language and representation grammar of the M1 models. The UML meta-model that defines the internal structure of UML models is part of the M2 level. UML profiles that extend the UML meta-model also belong to this level M2. The M1 level (or model) is composed of information models that describe the informations of the level M0.The UML models, the Platform Independent Model (PIM) and the Platform Specific Model (PSM) belong to this level.M1 models conform to the M2 meta-model. The M0 level (or instance) is the lowest of the architecture and corresponds to the real world.

C. Query, Views and Transformations (QVT)

Model transformation is a central operation of Model Driven Engineering. For this reason, several transformation languages have been defined. Typically, transformation languages apply MDE techniques to transformations themselves. The principle is to offer a meta-model that allows building transformation models. As part of the MDA approach, the OMG introduced the QVT language (Query, View, Transformation).

Query is a query that takes as input a model and selects specific elements of this model. View is a model that derives from others models. Transformation takes an input model to modify it or to create another one.

The QVT standard has a hybrid character by supporting three transformation languages. The declarative part of QVT is defined by two languages of different levels of abstraction: QVT-Relations and QVT-Core.

QVT-Relations is a user-oriented language for defining transformations at a high level of abstraction. It has a textual and graphical syntax.

Published By:
Blue Eyes Intelligence Engineering
& Sciences Publication (BEIESP)

QVT-Core is a low-level technical language, defined by textual syntax. This language is used to specify the semantics of the QVT-Relations language, given as a Relations To Core transformation.

The imperative component of QVT is supported by the language Operational Mappings. This language extends the two declarative languages of QVT by adding imperative constructs (sequence, selection, repetition, etc.) as well as OCL constructs. Finally, QVT offers a second extension mechanism called Black Box to specify transformations; it is about invoking implemented transformations features in an external language.

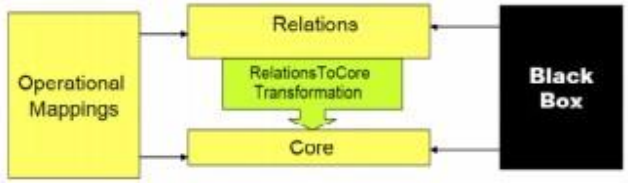

Fig. 3. QVT Architecture as shown in the OMG standard

IV. PROPOSED METHODOLOGY

The development of mobile applications is currently positioned as a major software engineering activity. The development of mobile applications is currently positioned as a major software engineering activity. However, the developers are faced with certain difficulties in choosing the technology for developing such applications because of the limitations imposed by the platforms on which these applications will be executed. Generally, each developed application is dedicated to a specific platform (Android, Windows Phone and iOS ...).

To answer to these constraints, we propose in this paper a valid model-driven approach in order to develop a mobile application independently of its execution platform; this application will play the role of the presentation layer in a final application respecting the n-tier architecture which is the objective of this work.

The study of architectures mobile platforms allows us to establish a PIM metamodel for this type of platforms. Apparently, the study of all architectures is a difficult task. However, most mobile platforms respect the MVC, MVP or the MVVM model. This is our basis for designing a mobile PIM that represents the presentation layer.

In the following, we present some components of android and iOS applications and how they work.

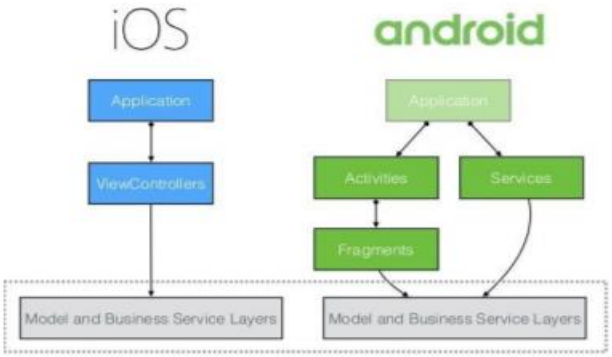

Fig. 4. Simplified architecture for Android and iOS platforms

An Android application is written in Java. Contrary to Java applications standard, an Android application can have multiple entry points. Specifically, an Android application can have multiple components and each of them may be an entry point into the program. There are four types of components: Activity, ContentProvider, Service, and BroadcastReceiver.

The iOS applications work on Objective-C / Swift. This type of application respects the MVC, MVP and MVVM models. The different windows constituting an iOS application are called UIViewControllers. Contrary to Android where an activity cannot hold a sub activity while a UIViewController can hold another UIViewController. The ViewController plays the role of control; it interprets and formats the model informations to pass them to the view.

It is important to note that several mobile platforms allow and encourage the use of architecture MVC, MVP and MVVM. To structure these applications. This chapter is dedicated to the presentation of these design patterns.

A. Layered architecture (N-Tiers)

Layered architecture responds to programmers' intuitive tendency to divide functionality into data presentations and data access. Classical topology divides different sets of features into three distinct layers: presentation, logic, and data.

1) *Presentation layer*: Corresponding to the display, the restitution on the workstation, the dialogue with the user.
2) *Business layer*: Business data processing, corresponding to the implementation of all management rules and business logic.
3) *Data access layer*: Corresponding to the data that are intended to be permanently retained.

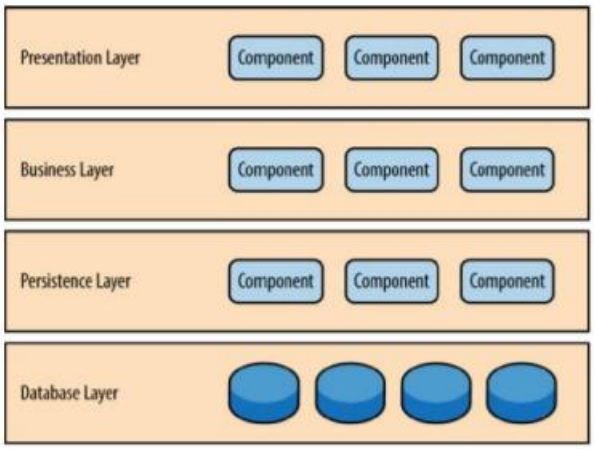

Fig. 5. Minimal representation of the layers of an n-tier architecture

In this approach, the layers communicate with each other through an exchange model, and each of them offers a set of services. The services of a layer are made available to the upper layer. The role of each layer and their communication interface are well defined, the functionalities of each of them can evolve without inducing changes in the other layers.

Published By:
Blue Eyes Intelligence Engineering
& Sciences Publication (BEIESP)

V. RESULT ANALYSIS

A. UML Meta-Model

To develop the algorithm of transformation between the source and target model, we consider the class diagram (see Figure 7), this class diagramme is used as a case study to validate our transformation.

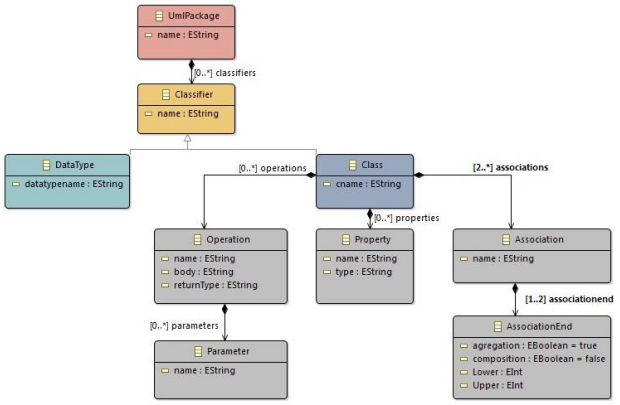

Fig. 6. UML Meta-model

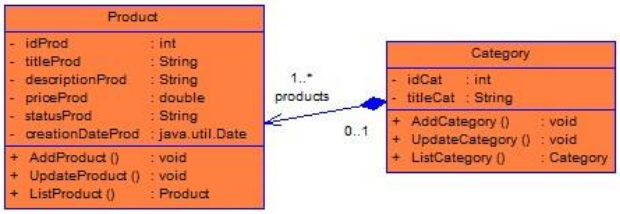

Fig. 7. UML Source Model

B. N-TIER Meta-Model

The n-tier meta-model is composed of three meta-models, a meta-model to generate the presentation layer, a meta-model to generate the business layer and a meta-model to generate the data layer.

1) *Presentation layer generation*: The presentation layer represents a mobile client. We have modeled this mobile application using an advanced development technique which is the MVC design pattern. Indeed, the developers today cut out the application to develop in 3 layers, Model, View and Controller. The distinction of these layers:

- facilitates the organization of project sources;
- allows every trade to work in parallel on the sources dedicated to them;
- Reduces the impact of changes to minimize the risk of error.

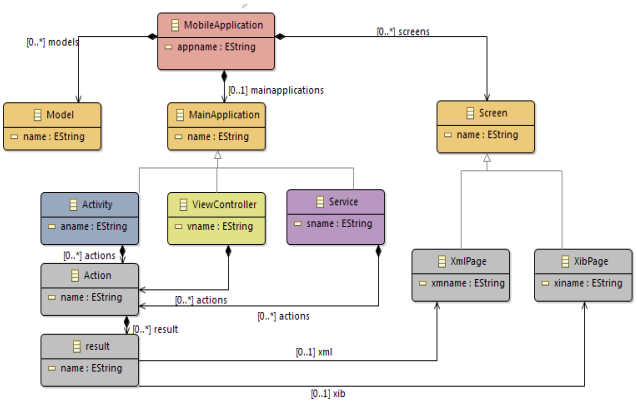

Fig. 8. Meta-model that represents the presentation layer

- MainApplication: each mobile application can contain a main activity that plays the role of the controller, it's called an activity in an Android platform, a ViewController in an iOS platform, it can be a service that also represents an activity without a user interface.
- Model: plays the role of the model in an MVC architecture.
- Activity: represents a class in an android application. An activity can contain one or more actions.
- Action: represents the concept of an Action, the action allows to transfer the answer of the user and to make the answer through the views.
- Result: contains the result generated by the action.
- Screen: represents the view in an MVC architecture, the view can be an xml page (XmlPage) in an Android palateform, an xib page (XibPage) in an iOS platform, we can even extend this meta-model to other platforms like Windows Phone, by adding a generalization (XamlPage) in the Screen part.

The algorithm presented in Fig. 9 translates the different transformation rules to generate the model that represents the presentation layer (see Fig.10).

```
modeltype umlMM uses "http://MetaModel.uml.sr";
modeltype androidMM uses "http://MetaModel.android.sr";

transformation AndroidTransformation(in umlModel : umlMM, out androidModel : androidMM);

main() {
      umlModel.objects()[UmlPackage]->map umlPackage2androidPackage();
}

mapping UmlPackage::umlPackage2androidPackage(): MobileApplication  {
         appname:='-name : '+self.name;
         screens:=umlModel.objects()[Operation]->map OperationtoXmlPage() ;
         var activ :=object Activity {
         name:=' Activity-name : '+self.name;
         actions:=umlModel.objects()[Operation]->map OperationtoAction() ;
        }
        }
mapping Operation::OperationtoAction(): Action  {
        name:='-name : '+self.name;
        var res:=object result{
         name:=self.name+'.xml';
       };
       }
mapping Operation::OperationtoXmlPage(): XmlPage  {
        name:='-name : '+self.name+'.xml';
        }
```

Fig. 9. Algorithm represents the different transformation rules between the uml source meta-model and the presentation layer target meta-model

Published By:
Blue Eyes Intelligence Engineering
& Sciences Publication (BEIESP)

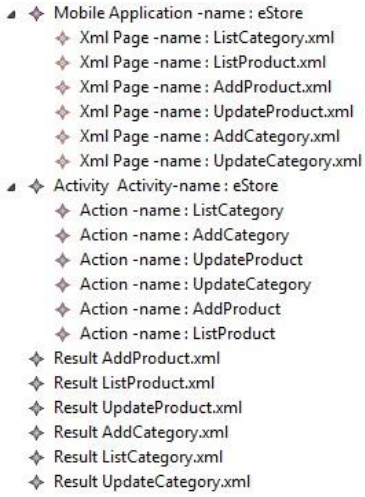

Fig. 10. Generated PSM Android model that represents the presentation layer

2) Business layer generation: To generate the Business Layer we have used the meta-model presented below :

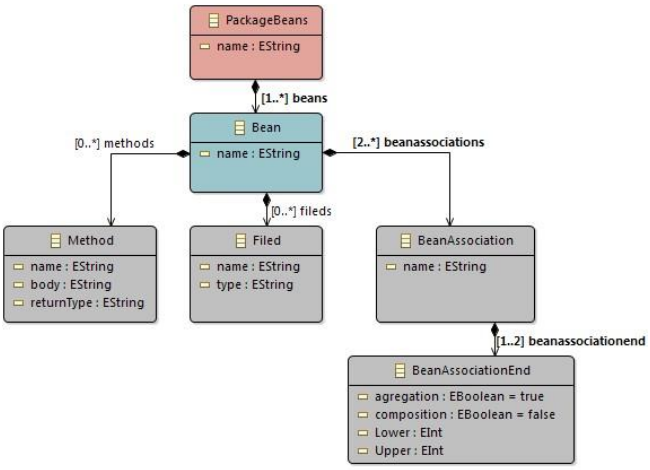

Fig. 11. Meta-model that represents the business layer

The algorithm presented in Fig. 12 translates the different transformation rules to generate the model that represents the business layer (see Fig.13).

```
transformation UmlToBean(in umlModel : umlMM, out beanModel : beanMM);

main() {
      umlModel.objects()[UmlPackage]->map umlPackageTobeanPackage();

}
mapping UmlPackage::umlPackageTobeanPackage(): PackageBeans  {
         result.name:='Bean-package-name : '+self.name;
         beans+=umlModel.objects()[Class]->map ClassToBean() ;
        }
mapping Class::ClassToBean(): Bean  {
        result.name:='Bean-name : '+self.cname;
        result.methods+=self.operations->map operationToMethod() ;
        result.fileds+=self.properties->map propertyTofiled() ;
        result.beanassociations+=self.associations->map AssociationToAssociation() ;
        }
mapping Operation::operationToMethod(): Method  {
        result.name:='Bean-Method-name : '+self.name;
        result.body:='Bean-Method-body : '+self.body;
        result.returnType:='Bean-Method-returnType : '+self.returnType;
        }
mapping Property::propertyTofiled(): Filed  {
        result.name:='Bean-Filed-name : '+self.name;
        result.type:='Bean-Filed-type : '+self.type;
        }
mapping Association::AssociationToAssociation(): BeanAssociation  {
        result.name:='Bean-Association-name : '+self.name;
        result.beanassociationend+=self.associationend->map AssociationEndToAssociationEnd() ;

        }
mapping AssociationEnd::AssociationEndToAssociationEnd(): BeanAssociationEnd  {
        result.composition:=self.composition;
        result.agregation:=self.agregation;
        result.Lower:=self.Lower;
        result.Upper:=self.Upper;
        }
```

Fig.12. Algorithm represents the different transformation rules between the uml source meta-model and the business layer target meta-model

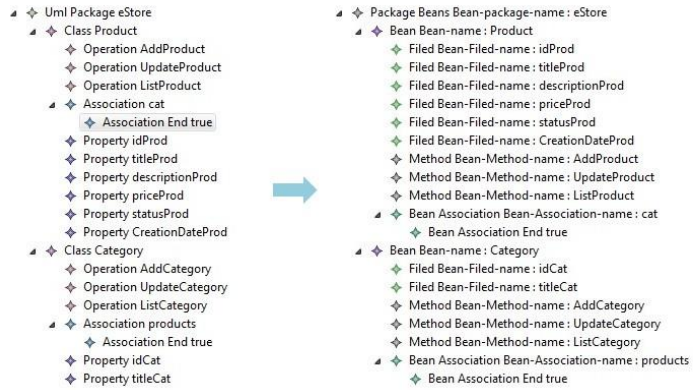

Fig.13. Generated PSM business layer model

3) Data layer generation: To generate the data layer, we have performed a model to model transformation, from our source meta-model Bean (see fig.11) to the target meta-model presented below :

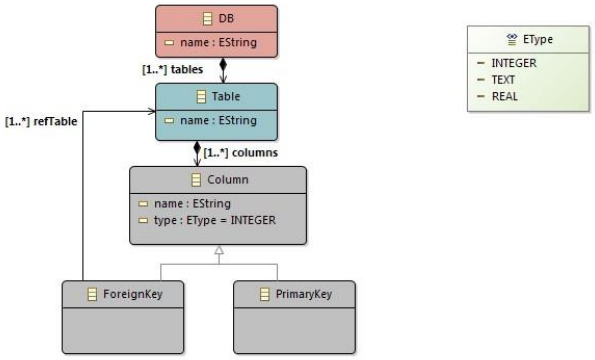

Fig.14. Meta-model that represents the data layer

To generate the model that represents the data layer (see Fig. 16), we have used the algorithm represented in Fig.15.

```
main() {
      beanModel.objects()[PackageBeans]->map beanPackage2dbPackage();

}
mapping PackageBeans::beanPackage2dbPackage(): DB  {
         result.name:='Data-Base-name : '+self.name;
         result.tables+=self.beans->map BeanToTable() ;
        }
mapping Bean::BeanToTable(): Table  {
        result.name:='Table-name : '+self.name;
        result.columns+=self.fileds->map FiledToColumn() ;
        }
mapping Filed::FiledToColumn(): Column  {
        result.name:='Column-name : '+self.name;

       if ( self = PrimitifType::int ) then {
              result.type := EType::INTEGER;
        }else{
             if ( self = PrimitifType::String ) then {
               result.type := EType::TEXT;
            }endif;

        }endif;

        }
```

Fig.15. Algorithm represents the different transformation rules between the Bean source meta-model and the data layer target meta-model

Published By:
Blue Eyes Intelligence Engineering
& Sciences Publication (BEIESP)

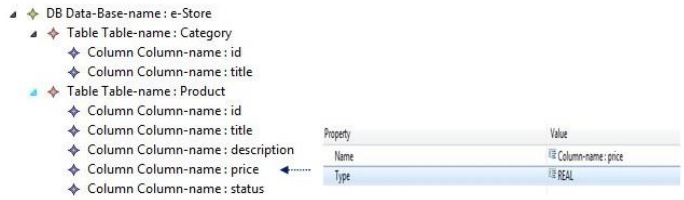

Fig.16. Generated PSM data layer model

Finally, we present in the Fig.17 the GUI (Graphical User Interface) that represents the presentation layer.

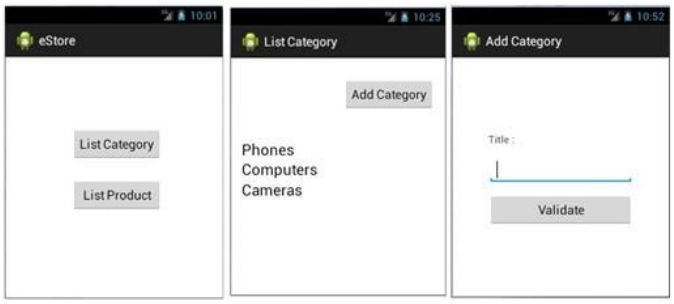

Fig.17. GUI that represents the presentation layer

VI. CONCLUSION

In this paper we have proposed original algorithms to generate, from a uml model, an n-tier model, this model is composed by three layers, a mobile platform which play the role of the presentation layer, a business layer, and a data layer. The transformation rules to generate the data access layer are note cited in this paper. We have demonstrated by this work, the validity of the MDA approach with the n-tier architecture. In a future work we will focus on applying the MDA approach to web service platform in the context of the n-tier architecture.

REFERENCES

1. J. Bezivin, S. Hammoudi, D. Lopes, F. Jouault, “Applying MDA approach for web service platform,” EDOC’04 preceedings of the 8th IEEE International Entreprise Distributed Object Computing Conference, pp. 58-70, 2004.
2. A. Srai, F. Guerouate, N. Berbiche, H. Drissi, “Generated PSM Web Model for E-learning Platform Respecting n-tiers Architecture,” International Journal of Emerging Technologies in Learning (iJET), vol. 12, no. 10, pp. 212-220, 2017.
3. A. Srai, F. Guerouate, N. Berbiche, H. Drissi, “MDA Approach for EJB Model,” 6th IEEE International Conference on Multimedia Computing and Systems (ICMCS’18). DOI: 10.1109/ICMCS.2018.8525924
4. M. Lachgar, “Approche MDA pour automatiser la génération de code natif pour les applications mobiles multiplateformes,” Thèse de Doctorat, 2017.

Published By:
Blue Eyes Intelligence Engineering
& Sciences Publication (BEIESP)