IJCSI International Journal of Computer Science Issues, Vol. 10, Issue 4, July 2013 ISSN (Online): 1694-0814 www.IJCSI.org

Fig.1 N-tiers Layers

Each Layer can be developed independently of the other provided that it adheres to the standards and communicates with the other layers.

4.1 The presentation Layer with MVC2 pattern

The presentation layer of most applications is often critical to the application's success. After all, the presentation layer represents the interface between the user and the application back-end.

Along time ago, web applications were very simple and the technology that was used to develop them was Common Gateway Interface (CGI). As applications became more complex, the defects and limits of this technology have emerged. Slowness and considerable consumption of memory. Therefore, the J2EE platform applies the architecture MVC2 [3]. In this paradigm, the model represents the information system consisting of javaBeans. The view represents the HTML pages returned to the user, and consists of JavaServerPage (JSP). The Controller is the glue between the two and it is composed of servlets. In short, during the early 80's with smalltalk, MVC was widespread in the field of object development. Many frameworks that implements MVC2 pattern have emerged, among them: Struts [1], PureMVC [29], Gwittir [14], SpringMVC [35], Zend [38], ASP.NET MVC2 [5]. Struts remains the most mature solution that has earned the trust of most developers, that is why we have taken it into account in our source meta-model.

4.2 The Business layer with Data Transfer Object and Dependency Injection patterns

Business logic layer is the Layer of abstraction between the presentation layer and persistence layer to avoid a strong coupling between these two layers and hide the complexity of the implementation of business processing to presentation layer. All business treatments will be implemented by this layer. The implementation of this layer is produced by the DTO pattern to render the result of running the service and the DI pattern to ensure a decoupling between objects.

In an article written in early 2004, Martin Fowler asked what aspect of control is being inverted. He concluded that it is the acquisition of dependent objects that is being inverted. Based on that revelation, he coined a better name for inversion of control: dependency injection [19].

In other words, Dependency Injection is a worthwhile concept used within applications that we develop. Not only can it reduce coupling between components, but it also saves us from writing boilerplate factory creation code over and over again. Many frameworks that implements DI pattern have emerged, among them: Spring [35], Symfony dependency injection [37], Spring.NET [36], EJB [30], PicoContainer [31]. (We have used some Spring classes in our source meta-model).

Recently, with the development of mapping o/r tools, it becomes easier to transfer a model object on the client layer (UI), and the distribution of the service layer, other advantage of the DTOs, is privileged in N-tiers modern architectures, that is why we have taken it into account in our work.

4.3 The persistence Layer with DAO pattern

This layer is the entry point to the database. All operations required to create, retrieve, update, and delete data in the database are implemented in the components of this layer.

The Data Access Object (DAO) pattern is now a widely accepted mechanism to abstract the details of persistence in an application. In practice, it is not always easy to make our DAO's fully hidden in the underlying persistence layer.

The advantage of this abstraction is that we can change the persistence mechanism without affecting the logic domain. All we need to change is the DAO layer which, if designed properly, is a lot easier to do than changing the entire logic domain. In fact we might be able to cleanly swap in a new data access layer for our new database or alternate persistence mechanism. Many frameworks that implements DAO pattern have emerged, among them: SpringDao [35], JPA [32], Hibernate [15], iBatis [2], NHibernate [28], EJB [30]. We have used Hibernate in our work because it is the most used solution within the java community.

5. The transformation of MDA models

MDA establishes the links of traceability between the CIM, PIM and PSM models due to the execution of the models' transformations.

The models' transformations recommended by MDA are essentially the CIM transformations to PIM and PIM transformations to PSM. In our work, we perform the second transformation PIM to PSM devoted to N-tires web applications.

5.1 Approach by modeling

Currently the transformations of models can be written according to three approaches: Approach by Programming,approach by Template and approach by Modeling. The approach by Modeling is the one used in the present paper. It consists of applying concepts from model engineering to models' transformations themselves. The objective is modeling a transformation, to reach perennial and productive transformation models, and to express their independence towards the platforms of execution. Consequently, OMG elaborated a standard transformation language called MOF 2. 0 QVT [24]. The advantage of the approach by modeling is the bidirectional execution of transformation rules. This aspect is useful for the synchronization, the consistency and the models reverse engineering [9].

Figure 2 illustrates the approach by modeling. Models' transformation is defined as a model structured according to MOF2.0 QVT meta-model. The MOF 2 0 QVT meta-model express some structural correspondence rules between the source and target meta-model of a transformation. This model is a perennial and productive model that is necessary to transform in order to execute the transformation on an execution platform.

IJCSI International Journal of Computer Science Issues, Vol. 10, Issue 4, July 2013 ISSN (Online): 1694-0814 www.IJCSI.org

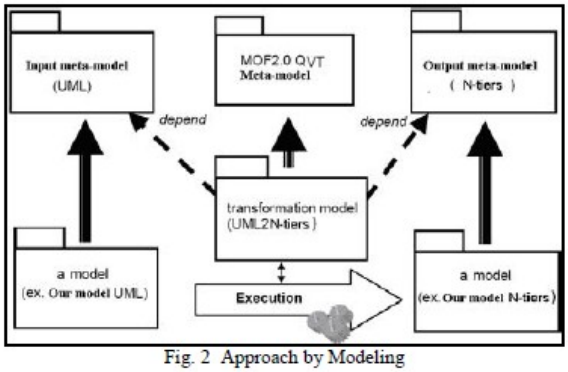


Fig. 2 Approach by Modeling

5.2 MOF 2.0 QVT

Transformations' models are at the heart of MDA, a standard known as MOF 2.0 QVT being established to model these changes. This standard defines the meta-model for the development of transformation model. The QVT standard has a hybrid character (declarative / imperative) in the sense that it is composed of three different transformation languages.

The imperative style languages are better suited for complex transformations including a significant algorithm component. Compared to the declarative style, they have the advantage of optional case management in a transformation. For this reason, we chose to use an imperative style language in this paper.

The imperative QVT component is supported by Operational Mappings language. The vision requires an explicit imperative navigation as well as an explicit creation of target model elements. The Operational Mappings language extends the two declarative languages of QVT, adding imperative constructs (sequence, selection, repetition, etc.) and constructs in OCL edge effect.

This work uses the QVT-Operational mappings language implemented by SmartQVT [33]. SmartQVT is the first open source implementation of the QVT-Operational language. The tool comes as an Eclipse plugin under EPL license running on top of EMF framework. This tool is developed by France Telecom R & D project and partially funded by the European IST Model Ware.

SmartQVT is composed of 3 components:

- **QVT Editor:** helps end users to write QVT specifications.
- **QVT Parser:** converts the QVT concrete textual syntax into its corresponding representation in terms of the QVT meta-model.
- **QVT Compiler:** produces, from a QVT model, a Java program on top of EMF generated APIs for executing the transformation. The format of the input is a QVT specification provided in XMI 2.0 in conformance with the QVT meta-model.

6. UML and N-tiers architecture meta-models

To develop the transformation algorithm between source and target model, we present in this section, the various meta-classes forming the meta-model UML source and the meta-model N-tiers target.

6.1 Meta-model UML source

The source meta-model structures a simplified UML model based on packages containing data types and classes. Those classes contain typed properties and they are characterized by multiplicities (upper and lower). The classes are composed of operations with typed parameters. Figure 3 illustrates the source meta-model.

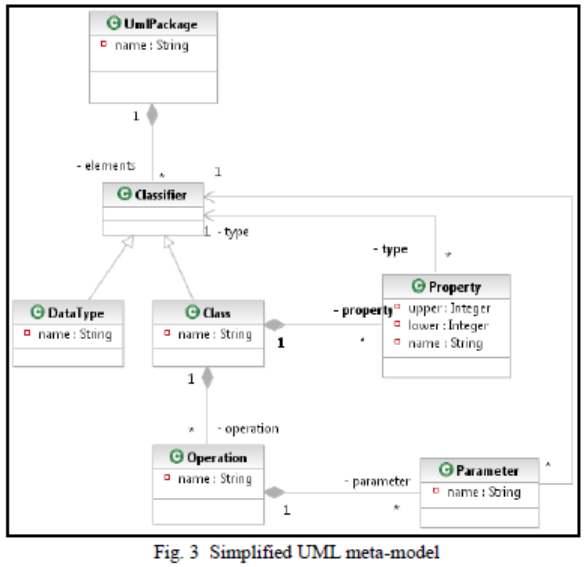


Fig. 3 Simplified UML meta-model

- **UmlPackage:** is the concept of UML package. This meta-class is connected to the meta-class ***Classifier***.
- **Classifier:** This is an abstract meta-class representing both the concept of UML class and the concept of data type.
- **Class:** is the concept of UML class.
- **DataType:** represents UML data type.
- **Operation:** is used to express the concept of operations of a UML class.
- **Parameter:** expresses the concept of parameters of an operation. These are of two types, Class or DataType. It explains the link between Parameter meta-class and Classifier meta-class.
- **Property:** expresses the concept of properties of a UML class. These properties are represented by the multiplicity and meta-attributes upper and lower.

6.2 Meta-model N-tiers target

Our target meta-model is composed of three essential part. Figure 4 illustrates the first part of the target meta-model. This meta-model represents a simplified version of the DAO pattern. It presents the different meta-classes to express the concept of DAO contained in the DaoPackage:

IJCSI International Journal of Computer Science Issues, Vol. 10, Issue 4, July 2013 ISSN (Online): 1694-0814 www.IJCSI.org

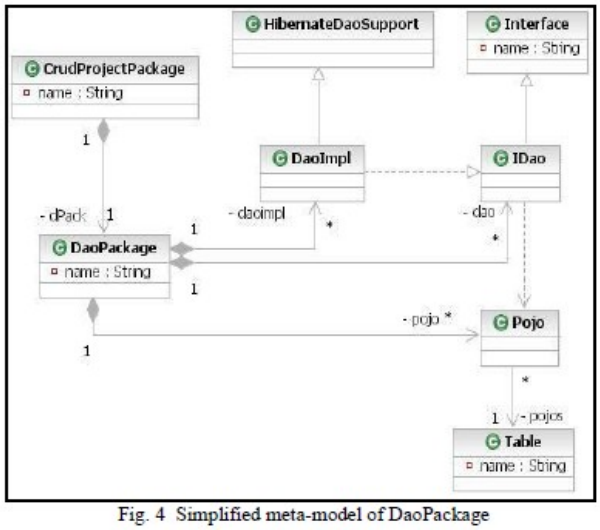


Fig. 4 Simplified meta-model of DaoPackage

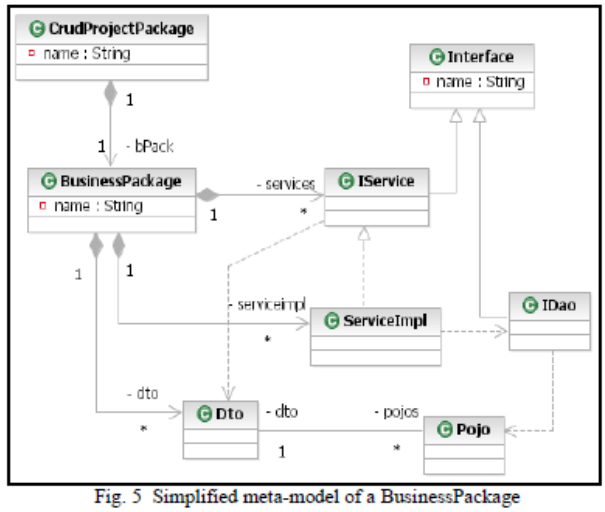


Fig. 5 Simplified meta-model of a BusinessPackage

- **CrudProjectPackage:** represents the project package. This meta-class is connected to the meta-class DaoPackage, BusinessPackage and UIPackage.
- **DaoPackage:** represents package which contains the different meta-classes to express the concept of DAO.
- **HibernateDaoSupport:** expresses the concept of generic class for DAOs, defining template methods for DAO initialization.
- **Interface:** is the concept of UML interface.
- **IDao:** represents the concept of Dao interface containing the methods definition to create, retrieve, update, and delete data in the database**.**
- **DaoImpl:** expresses the concept of Dao implementation, all methods to create, retrieve, update, and delete data in the database are implemented in this meta-class.
- **Pojo:** represents the concept of pojo. The latter extends the meta-class ***Class***. The pojos represents objects in the area of application. These objects communicate with the tables of relational database, which explains the meta-association with meta-class ***Table***.
- **Table:** is the concept of table in the relational databases. It contains a meta-attribute ***name*** which represents the table name in the database. The meta-class is connected by a meta-association to the meta-class ***Column.*** Figure 5 illustrates the second part of target meta-model. This meta-model is the business model of the application to be processed. In our case, we opted for

components such as DTO and DI pattern. Here, we present the different meta-classes to express the concept of DI contained in the Business Package.

This meta-model structures the models representing the business logic of the target application. This logic is essentially made up of DTO components.

- **BusinessPackage:** represents the package which contains the different meta-classes to express the concept of the business logic of target application.
- **Interface:** (already seen at the DaoPackage meta-model)
- **IService:** represents the concept of service interface containing the methods definition**.**
- **ServiceImpl:** expresses the concept of service implementation containing the methods representing in IDao meta-class and declared in IService meta-class.
- **IDao:** (already seen at the DaoPackage meta-model)
- **Dto:** represents the concept of business object that needs to be transferred across a process or network boundary. These objects contain all/some attributes of the pojos, which explains the meta-association with meta-class ***Pojo***.
- **Pojo:** (already seen at the DaoPackage meta-model)

Figure 6 illustrates the third part of the target meta-model. This meta-model represents a concept of MVC2 implementation in the user interface.

IJCSI International Journal of Computer Science Issues, Vol. 10, Issue 4, July 2013 ISSN (Online): 1694-0814 www.IJCSI.org

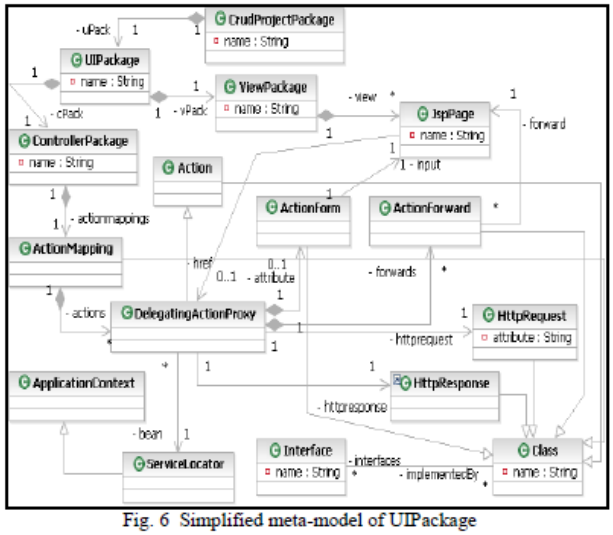


Fig. 6 Simplified meta-model of UIPackage

- **UIPackage:** represents the different meta-classes to express the concept of MVC2. This meta-class is connected to the meta-class ***ViewPackage*** and ***ControllePackage*** which represents respectively View and Controller package.
- **ActionMapping:** Represents the concept of ActionMapping classes. An ActionMapping class contains information to deploy of a class Action. This explains the connection with the meta-class ***Action***.
- **Action:** is the concept of action. Class Action contains its own processing of the application; hence it should be linked to the various beans.
- **DelagatingActionProxy:** represents the concept of Proxy for a Spring-managed Struts WebApplicationContext. The proxy is defined in the Struts-config file, specifying this class as the action class. This class will delegate to a Struts Action that is defined in Action bean in the ContextLoaderPlugIn context.
- **ActionForm:** represents the concept of ActionForm classes. An ActionForm represents a form containing the parameters of the request from the view (ViewPackage). This object is used by Action Class (This is particularly one of the four parameters of the operation execute()), which explains the link with the metaclass Action.
- **JspPage:** represents a Jsp page. An action class may be called from a hyperlink in a Jsp. This explains the link between the Jsp page and Action class. The link between ActionForward and Jsp page is trivial. ActionForm is linked to Jsp page because it contains the information that would be transmitted in the request and then filled in the actionForm. The link between Jsp page and HttpRequest expresses the fact that the Jsp page can use the information contained in an HttpRequest object.
- **HttpRequest:** is the concept of ***HttpServletRequest*** classes.
- **HttpResponse:** represents the concept of ***HttpServlet-Response*** classes.
- **ApplicationContext:** represents the concept of Central interface to provide configuration for an application, An ApplicationContext provides a Bean factory methods for accessing application components and Inheritance from a parent context. Definitions in a descendant context will always take priority. This means, for example, that a single parent context can be used by an entire web application, while each servlet has its own child context that is independent of other servlets.
- **ServiceLocator:** expresses the concept of Service lookup and creation involves complex interfaces and network operations.

This meta-model structures the models representing the view application. In this model, the Servlet invokes the execute( ) method on the instance of the action class. This method completes its processing and then calls the mapping.findforward( ) method with a return to a specified Jsp page.

Annexe 1 shows the global view of our meta-model target.

7. Transformation process from UML to N-tiers implementation

CRUD operations (Create, Remove, Update, and Display) are most commonly implemented in all systems. That is why we have taken into account in our transformation rules these types of transactions.

We first developed EMOF models corresponding to our source and target meta-models, and then we implemented the algorithm using the transformation language QVT Operational Mappings. To validate our transformation rules, we conducted several tests. For example, we considered the class diagram (see Figure 7). After applying the transformation on the UML model, composed by the classes User and advertissment, we generated the target model (see Figure 9).

Fig. 7 UML instance model

7.1 Transformation rules

By source model, we mean model containing the various classes of our business model. The elements of this model are primarily classes.

Main algorithm:

```
input umlModel:UmlPackage
output crudModel:CrudProjectPackage
begin
create CrudProjectPackage crud
create DaoPackage daoPackage
```

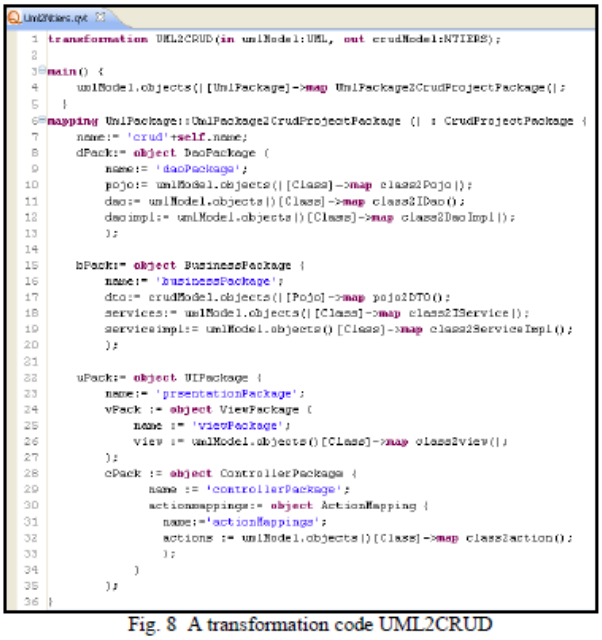
IJCSI International Journal of Computer Science Issues, Vol. 10, Issue 4, July 2013 ISSN (Online): 1694-0814 www.IJCSI.org

```
for each e ∈ source model

x = transformationRuleOnePojo(e)
link x to dp
x = transformationRuleOneIDao(e)
link x to dp
x = transformationRuleOneDaoImpl(e)
link x dp
end for
create BusinessPackage bp;

for each pojo ∈ target model

x = transformationRuleTwoDto(pojo)
link x to bp
end for

for each e ∈ source model

x = transformationRuleTwoIService(e)
link x to bp
x = transformationRuleTwoSrviceImpl(e)
link x to bp
end for
create UIPackage uip;
create ViewPackage vp
vp = transformationRuleThreeView(e)
create ControllerPackage cp
cp = transformationRuleThreeController(e)
link vp to uip
link cp to uip
link dp to crud
link bp to crud
link uip to crud
return crud
end
function
transformationRuleOnePojo(e:Class):Pojo
begin
create Pojo pj
pj.name = e.name
pj.attributes = e.properties
return pj
end

function
transformationRuleOneIDao(e:Class):IDao
begin
create IDao idao
idao.name = 'I'+e.name+ 'Dao'
idao.methods = declaration of e.methods
return idao
end

function
transformationRuleOneDaoImpl(e:Class):DaoImpl
begin
create DaoImpl daoImpl
daoImpl.name = e.name+ 'DaoImpl'

for each e1 ∈ DaoPackage

if e1.name = 'I'+e.name+ 'Dao'
put e1 in interfaces
end if
end for
link interfaces to daoImpl
return daoImpl
end
function
        transformationRuleTwoDto(p:pojo):Dto
begin
    create Dto dto
    dto.name = p.name
    dto.attributes = p.attributes
    return dto
end
function
transformationRuleTwoIService(e:Class):IService
begin
    create IService iservice
    iservice.name = 'I'+e.name+ 'Service'
    iservice.methods = declaration of e.methods
    return iservice
end
function
transformationRuleTwoServiceImpl(e:Class):Service Impl
begin
    create ServiceImpl serviceImpl
    serviceImpl.name = e.name+ 'ServiceImpl'

    for each e1 ∈ BusinessPackage

if e1.name = 'I'+e.name+ 'Service'
put e1 in interfaces
end if
    end for
    link interfaces to ServiceImpl
    return ServiceImpl
end
function
transformationRuleThreeView(e:Class):ViewPackage
begin
    create ViewPackage vp

    for each e ∈ source model

if e.methods.name ≠ 'remove'
create JspPage page
link page to vp
       end if
    end for
    return vp
end
function
transformationRuleThreeController(e:Class):Contro
llerPackage
begin
    create ControllerPackage cp
    create ActionMapping am
    for each page viewPackage
link page to actionForward
create actionForm
create Action action
create ActionForward actionForward
actionForm.input=page
actionForm.attribute=action
link page to actionForward
link actionForward to action
put action in am
    end for
    link am to cp
    return cp
```

IJCSI International Journal of Computer Science Issues, Vol. 10, Issue 4, July 2013 ISSN (Online): 1694-0814 www.IJCSI.org

```
end
```

Figure 8 represents the first part of the code of the transformation of UML model source to N-tiers target model.



```
transformation UML2CRUD(in umlModel:UML, out crudModel:NTIERS);

main() {
    umlModel.objects()[UmlPackage]->map UmlPackage2CrudProjectPackage();
}
mapping UmlPackage::UmlPackage2CrudProjectPackage() : CrudProjectPackage {
    name:= 'crud'+self.name;
    dPack:= object DaoPackage {
        name:= 'daoPackage';
        pojo:= umlModel.objects()[Class]->map class2Pojo();
        dao:= umlModel.objects()[Class]->map class2IDao();
        daoimpl:= umlModel.objects()[Class]->map class2DaoImpl();
        };

    bPack:= object BusinessPackage {
        name:= 'businessPackage';
        dto:= crudModel.objects()[Pojo]->map pojo2DTO();
        services:= umlModel.objects()[Class]->map class2IService();
        serviceimpl:= umlModel.objects()[Class]->map class2ServiceImpl();
        };

    uPack:= object UIPackage {
        name:= 'presentationPackage';
        vPack := object ViewPackage {
            name := 'viewPackage';
            view := umlModel.objects()[Class]->map class2view();
        };
        cPack := object ControllerPackage {
              name := 'controllerPackage';
              actionmappings:= object ActionMapping {
                name:='actionMappings';
                actions := umlModel.objects()[Class]->map class2action();
                };
            }
        };
}
```

Fig. 8 A transformation code UML2CRUD

The transformation uses, in entry, a model of the UML type named umlModel, and in output a model of the N-tiers named crudModel.

The entry point of the transformation is the method 'main'. This method makes the correspondence between all the elements of the UMLPackage type of the input model and the element of the CrudProjectPackage type of the output model. The objective of the second part of this code is to transform a UML package into N-tiers package, by creating the elements of type package 'Dao', 'Business' and 'Presentation. It is a question of transforming each class of package UML to Jsp page and Action in the View package, to DTO, IService and ServiceImpl in the Business package, and to Pojo, IDao and DaoImpl in the Dao package, without forgetting to give names to the different packages.

7.2 Result:

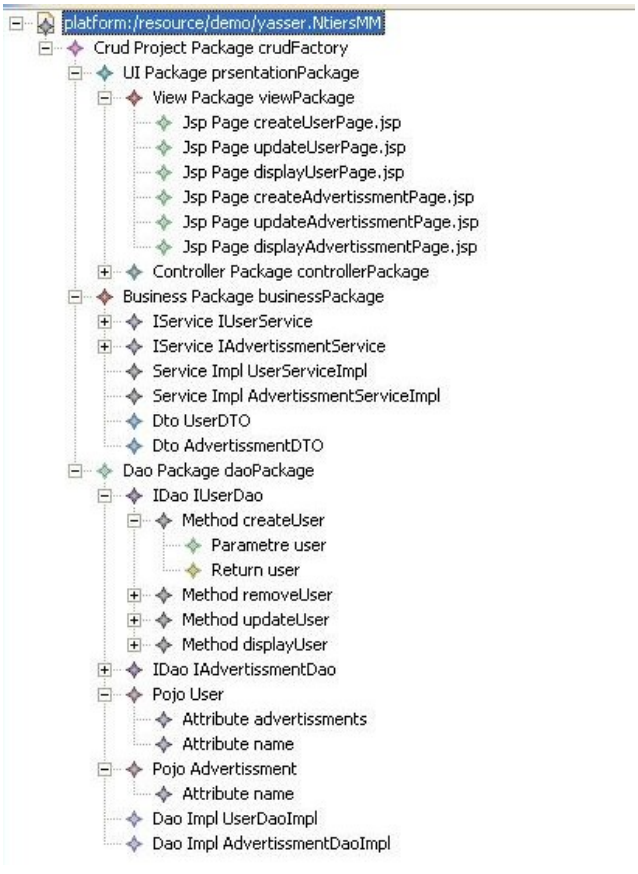


Fig. 9 Generated PSM N-tiers Web model

The first element in the generated PSM model is UIPackage which includes viewPackage that contains the JSPs, namely DisplayUserPage.jsp, Display-AdvertissmentPage.jsp,CreateUserPage.jsp, CreateAdvertissmentPage.jsp, UpdateUserPage.jsp, and UpdateAdvertissmentPage.jsp. Since the operation of the removal requires any form, we'll go to the controllerPackage element, which contains a single element ActionMapping. The latter contains eighteen delegating action proxy whose names are respectively DisplayXAction, CreateXAction, UpdateXAction, RemoveXAction, CreateXEndAction, UpdateXEnd- Action, where X should be replaced by User, and Advertissment. Operations for creation and update, add forms to enter new values. For this reason, we add CreateXEndAction and UpdateXEndAction.

The second element in the generated PSM model is businessPackage which includes three services' interfaces, three services' implementations and three Dtos' objects correspond to the two objects 'User' and 'Advertissment. The last element in

IJCSI International Journal of Computer Science Issues, Vol. 10, Issue 4, July 2013 ISSN (Online): 1694-0814 www.IJCSI.org

the generated PSM model is DaoPackage which contains three Pojos' objects that contains their attributes, three Daos' interfaces that contains methods with their parameters and their implementations.

8. Conclusion

In this paper, we applied the MDA approach to generate the N-tiers web application based on UML class diagram to generate a skeleton of a social network and create appropriate advertisements to the users in function of them profiles.

This involves developing all meta-classes needed to be able to generate an N-tiers application respecting a MVC2, DI and DAO patterns, then we applied the approach by modeling and used the MOF 2.0 QVT standard as a transformation language. The transformation rules defined allow browsing the source model instance class diagram, and generating, through these rules, an XML file containing layers of N-tiers architecture according to our target model. This file can be used to produce the necessary code of the target application. The algorithm of transformation manages all CRUD operations. Moreover, it can be re-used with any kind of methods represented in the UML class diagram. In the future, this work should be extended to allow the generation of other components of Web application besides the configuration files. For instance, we will be able to provide part of user interface. Afterward we can consider integrating other execution platforms like PHP and DotNET.

References

[1] Apache Software Foundation: The Apache Struts Web Application Software Framework (http://struts.apache.org).

[2] Apache Software Foundation: The Apache iBatis Framework (http://ibatis.apache.org/).

[3] Alur, D., Crupi, J., Malks, D., Core J2EE Patterns: Best Practices and Design Strategies (Prentice Hall, 2003).

[4] AndroMDA. http://www.andromda.org/.

[5] ASP.NET MVC site http://www.asp.net/mvc/

[6] Blanc, X., MDA en action : Ingénierie logicielle guidée par les modèles (Eyrolles, 2005).

[7] Bezivin, J., Busse, S., Leicher, A., Suss, J.G, Platform Independent Model Transformation Based on TRIPLE. In Middleware'04: Proceedings of the 5th ACM/IFIP/USENIX International Conference on Middleware, pages 493- 511,2004.

[8] Bezivin, J., Hammoudi, S., Lopes, D., Jouault, F., Applying MDA approach for web service platform. In EDOC'04 preceedings of the 8th IEEE International Entreprise Distributed Object Computing Conference, pages 58-70, 2004.

[9] Czarnecki, K., Helsen, S., Classification of Model Transformation Approaches, in online proceedings of the 2nd OOPSLA'03 Workshop on Generative Techniques in the Context of MDA. Anaheim, October, 2003.

[10] Cong, X., Zhang, H., Zhou, D., Lu, P., Qin, L., A Model- Driven Architecture Approach for Developing E-Learning Platform , Entertainment for Education. Digital Techniques and Systems Lecture Notes in Computer Science, Volume 6249/2010, 111-122, DOI: 10.1007/978-3-642-14533-9_12, 2010.

[11] Distante, D., Rossi, G., Canfora, G., Modeling Business Processes in Web Applications: An Analysis Framework. In Proceedings of the The 22nd Annual ACM Symposium on Applied Computing (Page: 1677, Year of publication: 2007, ISBN: 1-59593-480-4).

[12] Eclipse.org. ATLAS Transformation Language (ATL). http://www.eclipse.org/m2m/atl/.

[13] Gharavi, V., Mesbah, A., Deursen, A. V., Modelling and Generating AJAX Applications: A Model-Driven Approach. Proceeding of the7th International Workshop on Web- Oriented Software Technologies, New York, USA (Page: 38, Year of publication: 2008, ISBN: 978-80-227-2899-7)

[14] Gwittir Source Web Site http://code.google.com/p/gwittir/

[15] Hibernate Framework (http://www.hibernate.org/)

[16] Jouault, F., Allilaire, F., Bézivin, J., Kurtev, I., ATL: A model transformation tool. Science of Computer Programming-Elsevier Vol. 72, n. 1-2: pp. 31-39, 2008.

[17] Koch, N., Transformations Techniques in the Model-Driven Development Process of UWE, Proceeding of the 2nd International Workshop Model-Driven Web Engineering, Palo Alto (Page: 3 Year of publication: 2006 ISBN: 1-59593- 435-9).

[18] Kraus, A., Knapp, A., Koch N., Model-Driven Generation of Web Applications in UWE. Proceeding of the 3rd International Workshop on Model-Driven Web Engineering, CEUR-WS, Vol. 261, 2007

[19] Fowler, M., Inversion of Control Containers and the Dependency Injection pattern (http://martinfowler. com/articles/injection.html)

[20] Mbarki, S., Erramdani, M., Toward automatic generation of mvc2 web applications, InfoComp - Journal of Computer Science, Vol.7 n.4, pp. 84-91, December 2008, ISSN: 1807- 4545.

[21] Mbarki, S., Erramdani, M., Model-Driven Transformations: From Analysis to MVC 2 Web Model, International Review on Computers and Software (I.RE.CO.S.), Vol. 4. n. 5, pp. 612-620, September 2009.

[22] Mbarki, S., Rahmouni, M., Erramdani, M., Transformation ATL pour la génération de modèles Web MVC 2, 10e Colloque Africain sur la Recherche en Informatique et en Mathématiques Appliquées, Theme5:Information Systems, CARI 2010.

[23] Meta Object Facility (MOF), version 2.0, January 2006, http://www.omg.org/spec/MOF/2.0/PDF/

[24] Meta Object Facility (MOF) 2.0 Query/View/Transform-ation (QVT), Version 1.1, December 2009. http://www.omg.org/spec/QVT/1.1/Beta2/PDF/

[25] Miller, J., Mukerji, J., al. MDA Guide Version 1.0.1, 2003. http://www.omg.org/docs/omg/03-06-01.pdf.

[26] Nasir, M.H.N.M., Hamid, S.H., Hassan, H., WebML and .NET Architecture for Developing Students Appointment Management System, Journal of applied science, Vol. 9, n. 8, pp. 1432-1440, 2009.

[27] Ndie, T. D., Tangha1, C., Ekwoge, F. E., MDA (Model- Driven Architecture) as a Software Industrialization Pattern: An Approach for a Pragmatic Software Factories. J. Software Engineering & Applications, pages 561-571, 2010

[28] NHibernate Framework home site (http://nhforge.org/)

[29] Puremvc framework (http://puremvc.org/).

[30] Panda, D., Rahman, R., Lane, D., EJB3 in action (Manning co., 2007).

[31] PicoContainer. http://www.picocontainer.org/

[32] Schincariol, M., Keith, M., Pro JPA 2: Mastering the Java Persistence API (Apress, 2009)

[33] SmartQVT documentation Copyright © 2007, Copyright(c) France Telecom. http://smartqvt.elibel.tm.fr/doc/index.html [34] Soler, E., Trujillo, J., Blanco, C., Fernandez-Medina, E., Designing Secure Data Warehouses by Using MDA and QVT. Journal of Universal Computer Science, vol. 15, no. 8 pages 1607-1641, 2009.

[35] Spring Source Web Site (http://www.springsource.org/).

[36] SpringNet Web Site(http://www.springframework.net/).

[37] Symfony open-Source PHP Web Framework Site (http://www.symfony-project.org/

[38] Zend Framework (http://framework.zend.com/).

[39] UML Infrastructure Final Adopted Specifcation, version 2.0, September 2003, http://www.omg.org/cgi-bin/doc?ptc/03-09- 15.pdf
[40] XML Metadata Interchange (XMI), version 2.1.1, December 2007, http://www.omg.org/spec/XMI/

Lamlili el Mazoui Nadori Yasser is pursuing his Ph.D at Mohammed First University in the Faculty of Sciences. He got a degree of an engineer in Computer Sciences from the National School of Applied Sciences at Oujda. He received his M.Sc. degree in New Information and Communication Technologies from the faculty of sciences and Techniques at Sidi Mohamed Ben Abdellah University. His research activities at the MATSI Laboratory (Applied Mathematics, Signal Processing and Computer Science) have focused on WebMarketing in social networks using MDA (Model Driven Architecture) approach.

Mohammed Erramdani teaches the concept of Information System at Mohammed First University. He got his thesis of national doctorate in 2001. His activities of research in the MATSI Laboratory (Applied Mathematics, Signal Processing and Computer Science) focusing on MDA (Model Driven Architecture) integrating new technologies XML, EJB, MVC, Web Services, etc.

Ibtissam Arrassen Graduate as Computer Science Enginner from the INPT(National Institut of Poste and Telecommunication) and Ph-D-Student at Faculty of Sciences, Laboratory for Computer Science Research, Mohammed First University, Oujda, Morocco.

Redouane Esbai Ph.D at Mohammed First University in the Faculty of Sciences. He got a degree of an engineer in Computer Sciences from the National School of Applied Sciences at Oujda. He received his M.Sc. degree in New Information and Communication Technologies from the faculty of sciences and Techniques at Sidi Mohamed Ben Abdellah University. His research activities at the MATSI Laboratory (Applied Mathematics, Signal Processing and Computer Science) have focused on MDA (Model Driven Architecture).

Mimoun Moussaoui is a Professor, Vice-Director of High School of Technology and Responsible of the MATSI Laboratory (Applied Mathematics, Signal Processing and Computer Science) at Mohammed First University, Oujda, Morocco.