MDA-Based Modeling and Transformation to Generate N-Tiers Web Models

M'hamed Rahmouni*, Samir Mbarki
Department of Computer Science, Faculty of Science, Ibn Tofail University, Kenitra, BP 133, Morocco.

Corresponding author. Tel.: +212 664 79 94 44; email: md.rahmouni@yahoo.fr
Manuscript submitted: August 8, 2014; accepted March 20, 2015.
doi: 10.17706/jsw.10.3.222-238

Abstract: Data interchanges between companies are increasing. To improve this interchange and meet the increasing user needs, various frameworks and patterns are integrated for producing stable, maintainable and testable code. Some of the design patterns that will be used in the applications design and development are the MVC model, the DAO and DI (Dependency Injection) patterns. In this paper, we integrate these patterns to generate the N-tiers web model and thereafter generate the N-tiers application web code from this model. To obtain this, we start by modeling the Spring IoC, Struts2 and Hibernate frameworks for establishing their meta-models. Each framework from these is based on a pattern from the cited above. After establishing the different meta-models, we lead a model transformation process to generate N-tiers web model from the integrated meta-models. The model-to-model transformations are also clearly and formally established by using ATL transformation language. The model-to-code transformation will be the subject of the future work. Finally, a case study is provided to exemplify the generated PSM model respecting the architecture overview of MVC 2, DI and DAO patterns.

Key words: MDA, hibernate DAO, spring IoC, struts, ATL transformation, n-tiers architecture.

1. Introduction

Data interchanges between companies are increasing. To improve this interchange and meet the increasing user demands, various frameworks and patterns are integrated for producing stable, maintainable and testable code. Some of the design patterns that will be used in our application design and development are the MVC (Model-View-Controller) model, the DAO (Data Access Object) and DI (Dependency Injection) patterns. To have the different patterns cited above, we will integrate many frameworks based on these patterns like: Struts2 [1], Hibernate [2] and Spring IoC [3], [4]. At the same time, the traditional mode has been changed greatly by the frameworks integration.

Struts 2, Spring and Hibernate can be integrated to develop web sites. The goal is to reuse existing application services and minimize data duplication in different information systems. In technical terms, Struts 2 is a framework based on MVC 2 architecture. This framework is built for designing web applications. Using such architecture provides a separation code between views that display, the data store and pure logic part. This separation provides better maintainability and scalability of the application. Spring provides an abstraction layer to access data, including JPA, Hibernate, iBATIS, or JDBC directly. Working with a Spring service will allow the developer to ignore the technical properties of the data store underlying. The Hibernate framework is the most powerful and popular Object Relational Mapping (ORM) that can be used

with J2EE. All access to data or schema of the system is done using hibernate.

The application development architecture precisely the Web systems development [5], is as well focusing on the model driven engineering approach. The approach used in this work is the Model-Driven Architecture (MDA) [6]-[7].Most of the current Web Engineering approaches (WebML [8], OO-H [9], OOWS [10], UWE [11], WebSA [12]) already propose to build different views of Web systems following a horizontal separation of concerns.

The objective of this work is to generate automatically an N-tiers web model that is a PSM model. This model respects the architecture of MVC2, DI and DAO patterns. To meet this objective, we elaborate the PIM and PSM meta-models then we establish the traceability links between the components of these meta-models and thereafter we define the different rules by ATL transformation language. Finally, we demonstrate and exemplify these rules by a case study.

The remainder of this paper is organized in eight sections as follows: Section II describes the process and methodology. Section III presents the MDA approach. Section IV explains N-tiers system architecture. Section V is dedicated to the UML and N-tiers architecture meta-models. The transformation rules implementation is the subject of Section VI. Section VII presents the transformation rules execution and the result of the execution process. The related work is the objective of Section VIII. Finally, Section VIX concludes the work and gives hints about future work.

2. Process and Methodology

In this work, the process starts by meta-modeling Spring IoC, Struts 2 and Hibernate DAO frameworks allow implementing the different CIM, PIM, and PSM meta-models. The CIM model of this work is an UML class diagram of a case study of an Employee management. The functional model (PIM) is a simplified UML meta-model. The PSM meta-model is composed of the meta-models of Spring IoC, Struts 2 and Hibernate DAO frameworks. The PSM meta-models are presented in the Fig. 3,-Fig. 5. The next step is to define the ATL transformation rules. This step starts by the implementation of KM3 models corresponding to each meta-model and the Ecore models corresponding to each KM3. The last step is to establish the traceability links between the components of source and target meta-models then, we define the different transformation rules in ATL transformation language. The result of this work is the n-tiers web model represented in EMF model. This is the configuration file of the proposed application. From the generated PSM model, we can generate the application code of the case study by applied an M2C transformation. The M2C transformation is neglected in this work. It will be the subject of future work.

The tools support of this work is the UML, ATL transformation language, MOF, XMI, KM3, OCL and EMF Project.

3. MDA (Model-Driven Architecture)

One of the main aims of MDA [13] (Model-Driven Architecture) approach is to separate design from architecture. The basic principle of this approach is the development of different models. Those models are the heart of MDA. This latter defines the requirement model (Computation Independent Model, CIM), then given the analysis and design model (Platform Independent Model, PIM) corresponding to its and finally the transformation of the PIM model into a PSM (Platform Specific Model) model specific to the target platform for the concrete implementation of the system. The techniques used in the MDA context are mainly the techniques of modeling and model transformations. MDA is related to multiple standards including UML [14], MOF [15], XMI [16], OCL [17] and many others.

This work is depicted to develop N-tiers application architecture. In the following section, we present the n-tiers system architecture of our application.

4. N-Tiers System Architecture

In this paper, we will integrate the Spring IoC, Struts 2 and Hibernate DAO frameworks in order to implement n-tiers web application. This application is composed from three layers. Each layer is managed by a framework through the frameworks cited above.

4.1. Web Presentation Layer / Ui Tier with Struts 2

The presentation layer allows display the data and the interaction of the application with the user. The separation of this layer permits in particular proposing several presentations for the same application: the same treatment layer can then be used for heavy duty application and a lightweight application. In this work, this layer will be managed by Struts 2 framework.

4.2. Business Layer with Spring IoC

The Business layer contains all business whose module components. These business components are responsible for managing the lifecycle of business objects managed by the module. This layer is now provided by Spring IoC.

4.3. Persistence Layer with Hibernate DAO

This layer is responsible for data access and their handling, regardless of the DBMS. All CRUD operations are implemented in this layer component. This layer can be handled by Hibernate Framework.

Fig. 1 shows the different layers which will be developed in this work.

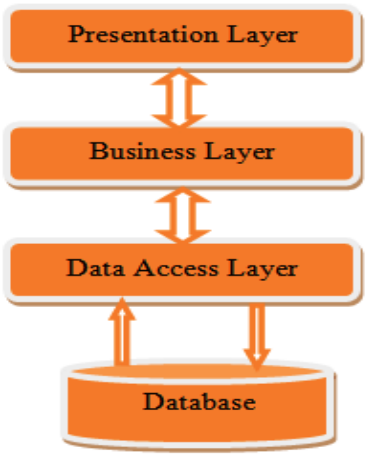

Fig. 1. N-tiers layers.

To achieve each implementation layer, we conduct a development method based on MDA approach. In this method, we begin by meta-modeling the frameworks and thereafter given the meta-model corresponding of each framework. The meta-modeling is the subject of the following section.

5. UML and N-Tiers Architecture Meta-Models

In this section, we present the various meta-classes forming the UML source and N-tiers target meta-models.

5.1. UML Source Meta-Model

Fig. 2 shows the source meta-model which is a simplified UML class diagram. *UMLPackage* represents the notion of UML package. The *UMLPackage* meta-class is linked to the Classifier meta-class. This latter represents as well the concept of UML class and the concept of data type. The Property meta-class

expresses the notion of an UML class properties or references to other classes (uni and bidirectional associations). The Operation meta-class presents the concept of the method in the UML context. This meta-class is composed of a set of Parameter. Also the Parameter meta-class represents the attributes and method parameters.

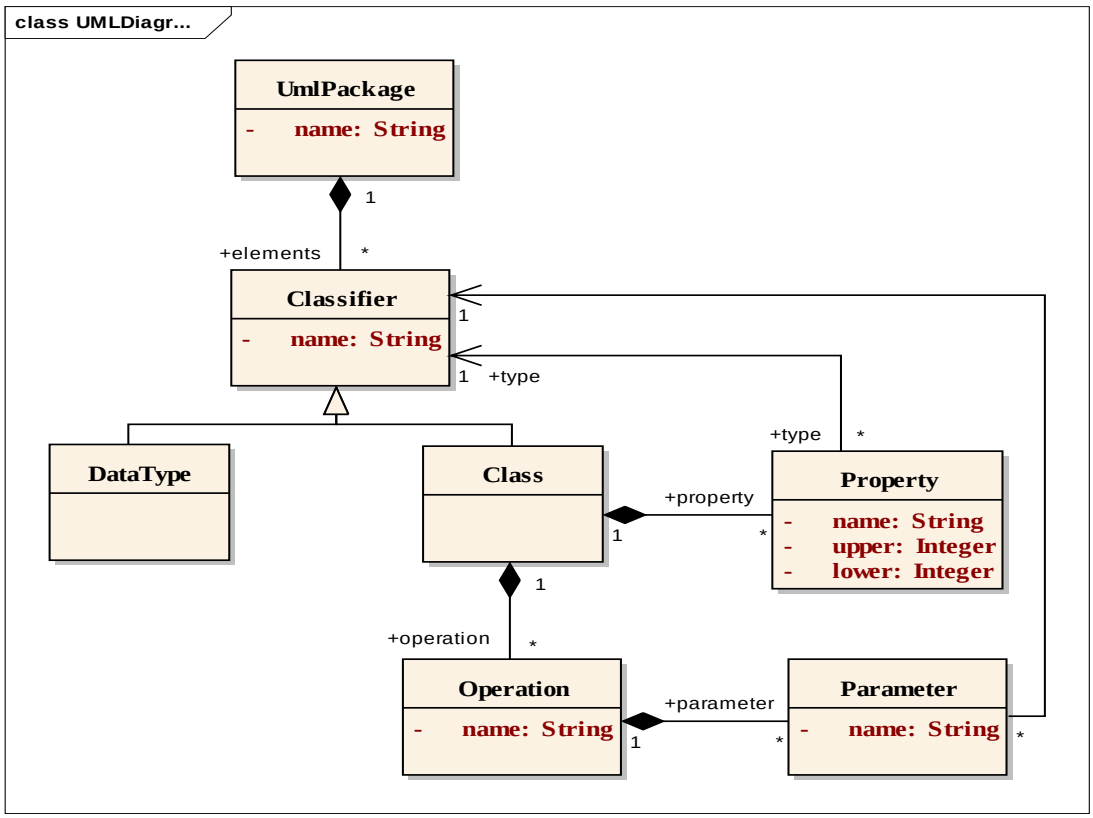

Fig. 2. Simplified UML meta-model.

5.2. N-Tiers Target Meta-Model

This meta-model is composed of a various meta-models. The first meta-model is the Spring IoC meta-model, the second is the Struts 2 meta-model and the third is the Hibernate DAO meta-model. In this section, we begin by modeling the Spring IoC.

5.2.1. Spring IoC

The Spring IoC container allows defining, mostly in XML but also through Java annotations, so-called beans, which are (usually named) instances of a Java type that is managed and configured by the Spring framework.

Fig. 3 presents the Spring IoC meta-model. Spring IoC manages the different Beans. In this meta-model, a bean instantiation can take three forms different depending on the given task:

1) An instantiated bean constructor using a java constructor;
2) A bean factory-class uses a class with a static factory method to retrieve an instance.
3) A factory-bean class uses a non-static factory method in another bean.

A bean specification may include the constructor arguments can uses are a constructor values or a factory method parameters. Otherwise, each bean specification may include properties containing values to be set by using the setters' method or directly on fields. Values can be referenced to another beans or be either a basis or again beans.

The Spring IoC defines four supported collection types: list, set, map, and props. Each of them allows both

setting static values and references to other beans within the context... except for props. There doesn't seem to be a way to set a reference when using props. Each property may take form of one or several lists, props, or reference. These property forms are:

Property: Represents a data type.

Props: This can be used to inject a collection of name-value pairs where the name and value are both Strings.

Prop*:* Represents a property from the properties of the properties file.

Value: Represents the value of property or many properties.

Ref: Represents the reference of an element.

List*:* This helps in wiring ie injecting a list of values, allowing duplicates.

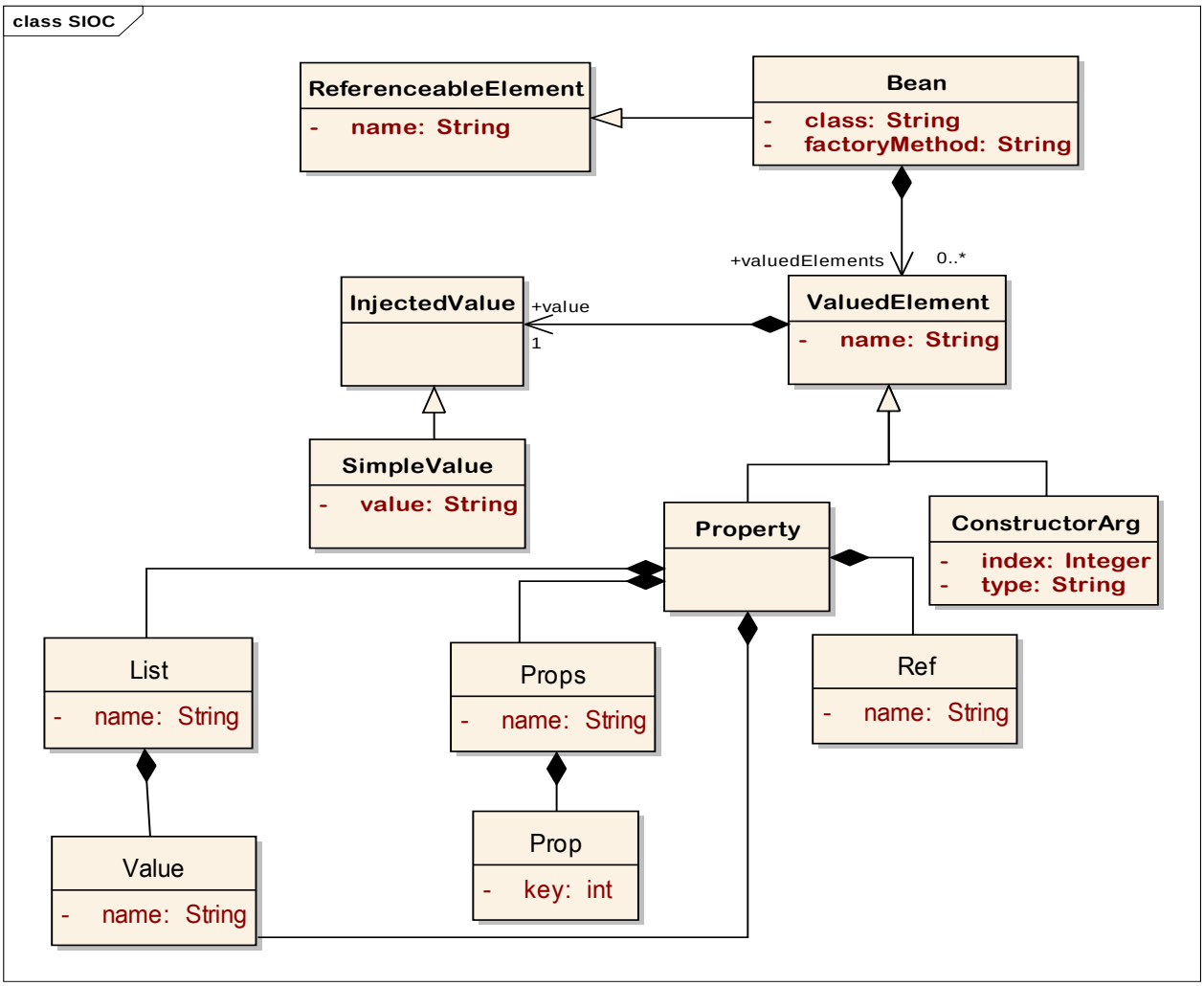

Fig. 3. Spring IoC meta-model.

5.2.2. Struts 2 framework

Struts 2 [1] is a second-generation web application framework that implements the Model-View-Controller (MVC) design pattern. All Struts 2 components will be detailed in the following section.

5.2.2.1. Struts 2 core components

The Struts 2 framework is used for easier web application development cycle, from building to maintaining the applications. The various Struts 2 components can be categorized into the following groups,

according to the Struts2 architecture. The principle components are Action, Interceptors and Value Stack/OGNL. The different components are grouped in the following meta-model.

5.2.2.2. Struts 2 meta-model

Fig. 4 illustrates the second part of the target meta-model. This meta-model represents the concept of MVC2 model. The meta-classes of Struts 2 meta-model are as follows:

1) **Model Package:** Represents the concept of UML package and designed the notion of Model in the MVC2 architecture. This package is composed of a set of actions.
2) **Controller Package:** Represents the concept of controller. The Controller is dedicated to map incoming HTTP requests to actions.
3) **View Package:** This is the package of Views. This package groups the different Jsp pages.
4) **Action Mapper:** Represents the *Action Mapper* class concept. The *ActionMapper* interface provides a mapping between HTTP requests and action invocation request and vice-versa.
5) **Action Proxy:** Represents the concept of *ActionProxy class*. According to the configuration file it creates an instance of an ActionInvocation class and delegates the control. The *ActionProxy* specifies the appropriate method for each Action class called.
6) **ActionInvocation:** Represents the *ActionInvocation* class concept. It is responsible for command pattern implementation. An *ActionInvocation* encapsulates how the Action is executed when a request is invoked. It is responsible for invoking the Interceptors one by one and after that invokes the Action. In the Other hand, it is responsible to associate the proper result to the proper Action according to the configuration file.
7) **Action:** This is the concept of action. The Action serves to transfer the user requests and renders the response through the result view.
8) **Jsp Pages:** Represents a Jsp package. The Jsp page serves to call an Action class through a hyperlink. The link between Result and **Jsp** page is trivial. The Jsp page uses the information contained in an *HttpRequest* object.
9) **Result***:* Contains the result generated by an Action class.
10) **The Interceptors**: Represents the concept of *Interceptor* package. It composed from many *Interceptor* classes. The Interceptors are used to perform pre / post processing of the request.
11) **Interceptor***:* Represents the concept of interceptor classes.
12) **Http Request:** is the concept of *HttpServletRequest* classes.
13) **Http Response:** represents the concept of *HttpServletResponse* classes.
14) **Result:** Represents the concept of Result classes. It contains the result of the Action output and then returned this result to the user

5.2.3. Hibernate data access

Hibernate [2] is an open source solution like ORM (Object Relational Mapping) which facilitates the development of an application persistence layer. Hibernate can be represented in a database in Java objects and vice versa. The Hibernate framework facilitates the persistence and data retrieval in a database by realizing himself the objects creation and filler treatments thereof by accessing the database. Hibernate is mainly associated with databases [20]-[21].

5.2.3.1. Hibernate architecture

Hibernate uses the database specification from Hibernate Properties file. Automatic mapping is performed on the basis of the properties defined in hbm XML file defined for particular Java object. Hibernate requires several elements to operate:

1) A JavaBean class type that encapsulates the occurrence data of a table.
2) A configuration file that provides the correspondence between the class and the table (mapping).
3) The configuration properties especially information regarding the connection to the database.

A Hibernate mapping file defines a JavaBeans property name, a database column name, and defines a JavaBeans property name, a database column name, and a Hibernate type name. It maps a JavaBean class to a database table.

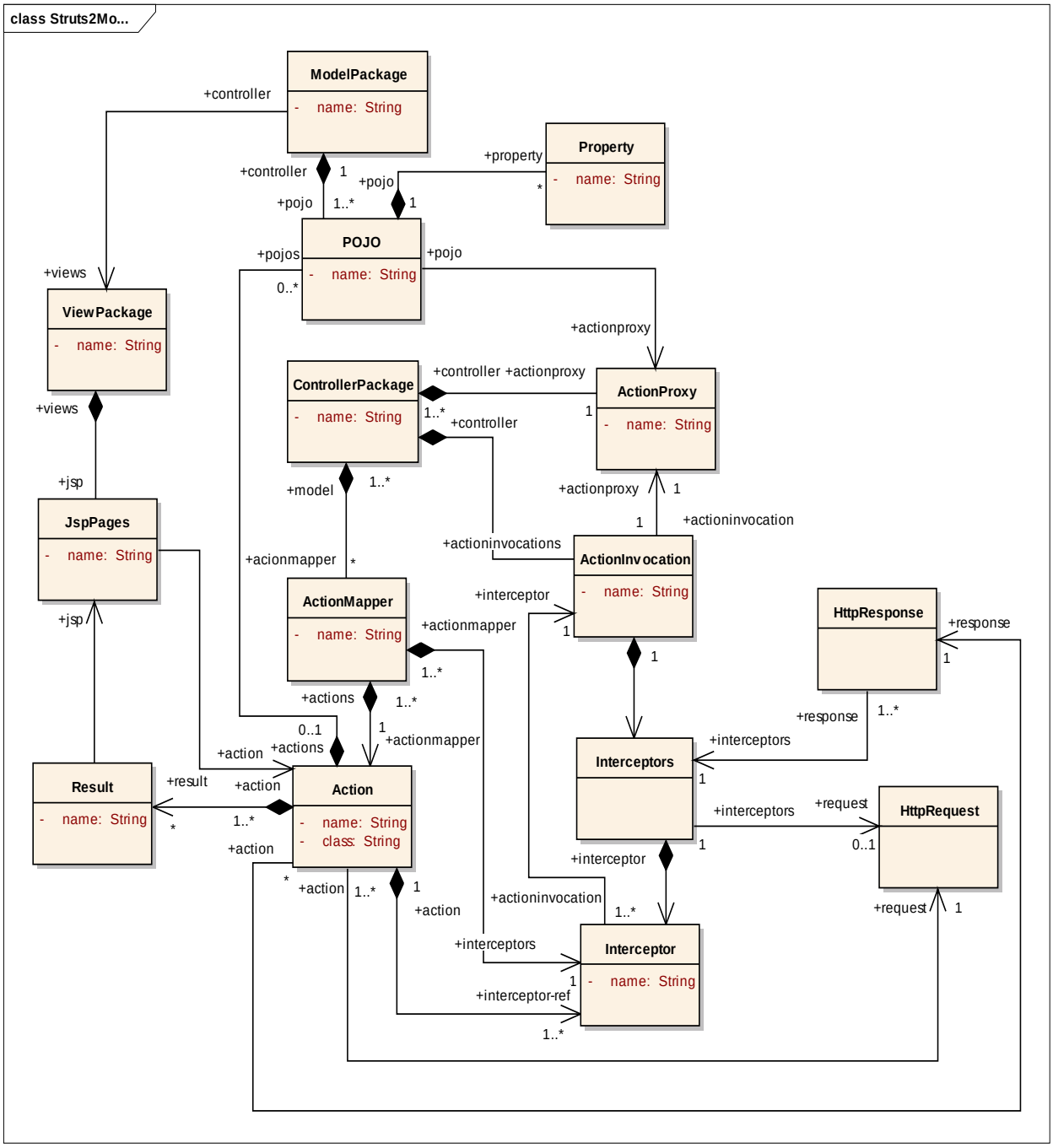

Fig. 4. Struts2 meta-model.

5.2.3.2. DAO (data access object) pattern

Objects in memory are often related to persistent data (stored in the database, in files, in directories, etc..). DAO pattern proposes to group the data persistent access in separate classes, rather than disperse

them. It comes especially don't write those access in the business classes, which will only be changed if the business management rules are changed.

Finally, the Hibernate DAO meta-model is shown in the Fig. 5.

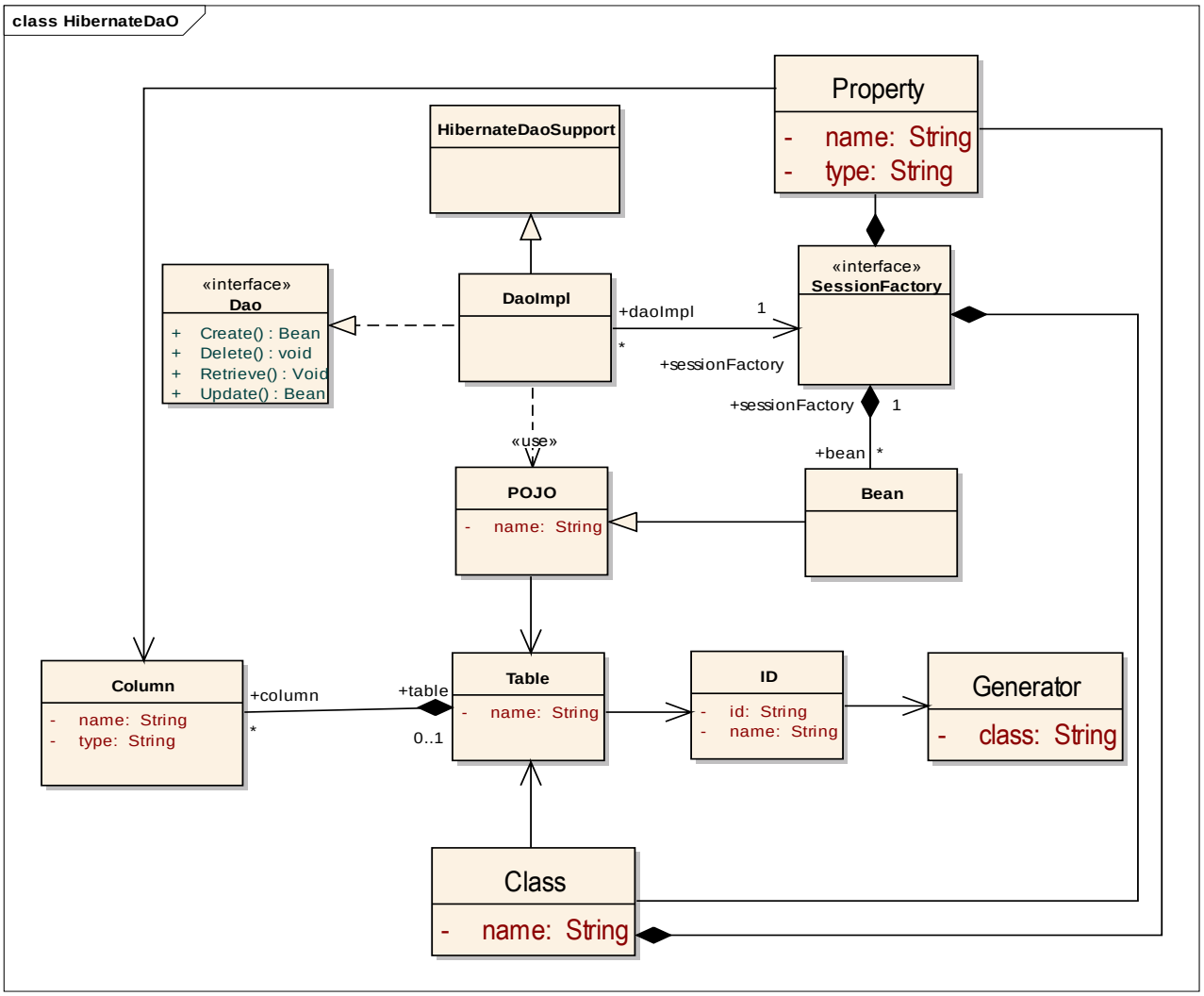

Fig . 5. Hibernate DAO Meta-model.

The different meta-model components are as follows:

1) Dao Package: Represents the package that contains the meta-classes to express the DAO concept.
2) HibernateDaoSupport: Can create own DAO. It defines the template methods for DAO initialization.
3) Interface: Represents the UML interface concept.
4) Dao: This is the Dao interface containing the CRUD (Create, Retrieve, Update and Delete) operations to manipulate data in the database.
5) DaoImpl: Represents the Dao Implementation class. This class contains several methods to create, retrieve, update and delete data in database.
6) Pojo: expresses the concept of pojo class. It extends the meta-class Class. Pojo is an ordinary java object. It communicates with the database tables.
7) Table: Represents the table in the area of relational databases. Each table is characterized by an id, a name and a column.
8) SessionFactory: This is a singleton instance which implements Factory design pattern.
9) Class: Expresses the concept of persistent java class.

10) ID: Represents the concept of ID class. This ID can specify which field acts as the primary key of database table.
11) Generator: This class represents the different id that will are generated automatically.
12) Property: This class represents the different properties of a given class.

6. Transformation Rules Implementation

This section presents the different steps from implementation to execution of different transformation rules. The first step is to implement the following meta-models: *N-tiers.km3*, *N-tiers.ecore*, *UML.km3*, *UML.ecore*. The second step is to establish the rules specification. After that, we define the different rules based on the specification rules. These rules are written in ATL language in a file named UML2N-tiers.atl. Finally, we prepare the source model. This model is an UML class diagram of Employee management. We translate the source model in XMI language.

To achieve this work, we have used different tools like: ATL plugin integrated in Eclipse, OCL, XMI, UML, EMF project, KM3 and MOF.

In the following sections, we present the ATL transformation language then the rules specification and finally the implementation and execution of ATL transformation rules. The *km3* and *ecore* meta-models cited above are not presented in this paper for letting it quite understandable and clear.

6.1. ATL: Atlas Transformation Language

ATLAS Transformation Language (ATL) is a model transformation language inspired by the OMG standard QVT. It developed in the framework of ATLAS project at LINA in Nantes [22]-[24]. ATL is part of Eclipse M2M (Model-to-Model) project [25]. Fig. 6 shows the ATL operational framework.

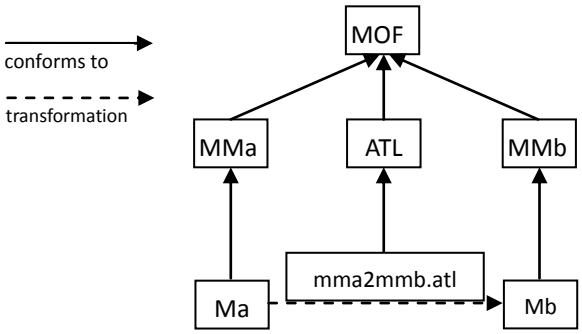

Fig. 6. ATL Operational framework.

6.2. Rules Specification

In this section, we present the main rules to transform an UML Class Diagram into an N-tiers Web model. The specification rules are as follow:

1) The View package is composed of a set of JSP pages.
2) Each UML package can generate a Struts2 package, a Spring package (named SIOC package) and a Hibernate package.
3) The SIOC package is composed of a set of Beans package.
4) The Beans is composed of a set of Bean classes.
5) The Struts 2 package is composed of a Struts package.
6) The Struts package is composed of a Controller package and a View package.
7) The Controller Package is composed of a set of Action classes.
8) The Hibernate mapping is composed of a set class.

9) The Bean class is composed of a set of DAO classes, BO (Business Object) classes and an Action classes.
10) Each Bean class is composed of a set of Property classes.
11) Each class is composed of a set of ID class and a Property classes.
12) The ID class is composed of a set of Column classes.
13) The Retrieve operation of root class does not generate a Form class.
14) Each Action class is composed of a set of Result classes.
15) Each Result is composed of a set of JSP pages.

6.3. Rules-Based Transformation Written in ATL

In this section we present the different rules which transform the UML model into N-tiers web model. These rules are as follow:

6.3.1. Rule 1: from operation to struts 2 action

This rule permits to generate the different action classes and jsp result of each Action. In this rule the name of jsp page is the name of the operation concatenated with the name of the class and followed by the extension ".jsp". Fig. 7 shows this rule.

6.3.2. Rule 2: from UML package to hibernate package

This rule generates the different XMI packages of the Hibernate configuration file. This rule is composed of a three rules. The different rules are as follows:

6.3.3. From UML class to hibernate class

This rule provides the elements representing the class in the Hibernate context. The name is the UML class name, the property table has the class name, the property receives the name of properties and the attribute id receives the value of "id" class. Fig. 8 presents this rule.

```
rule Operation2Action{

      from
            c : UML!Operation
      to
          js : NTiers!JspPage (
                    name<- if c.name<>'Delete'then
                              c.name+c.class.name+'.jsp'
                           else 'Retrieve'+c.class.name+'.jsp'
                         endif
                    ),
          frm : NTiers!Action(
                name<- c.name+c.class.name+'Action',
                method <- c.name+c.class.name,
                class <- 'com.web.struts2.'+c.name+c.class.name+'Action',
                result <- Sequence{fr}
               ),
           fr : NTiers!Result(
                name <- 'Success',
                type <- 'redirect',
                jsp <- js
             )
    }
```

Fig. 7. Rule that generates action classes and Jsp pages.

```
rule Class2Class{
  from
      d : UML!Class
  to
         cls : NTiers!Class(
          name <- d.name,
          table <- d.name,
          property <- d.properties,
          id <- d.id
        )
   }
```

Fig. 8. Rule that generates the different Hibernate DAO classes.

6.3.4. From property to hibernate property

This rule can generate the different properties of each class. The different properties generated are the name, the type and the column. Each column is characterized by the name, the length and the etat (null or not-null). This rule is shown in Fig. 9.

```
rule property2prperty{
  from
      a : UML!Property
  to
     pr : NTiers!Property(
          name<- a.name,
          type <- a.type,
          column <- Sequence{clm1}
          ),
   clm1 : NTiers!Column(
             name <- a.name,
             length <- '90',
           etat <- 'not-null'
           )
       }
```

Fig. 9. Rule that generates the hibernate class properties.

6.3.5. From UML ID to hibernate ID

This rule can generate the different id of each class. Each id is characterized by a name, a type and a column. Fig. 10 shows this rule.

```
rule id2id{

  from
      t : UML!ID
  to
      ids : NTiers!ID(
           name <- t.name,
           type <- t.type,
           column <- Sequence{clm}
       ),
      clm : NTiers!Column(
           name <- t.name+'_ID'
        )
    }
```

Fig. 10. Rule that generates the class ID and its column.

6.3.6. Rule 3: from UML package to spring package

This package is composed from various beans. The Spring's beans generated are: Action, Bo (Business Object) and DAO classes. This rule is as follow:

6.3.7. From Operation to Spring's Bean

This rule generates the different Beans and the bean properties. The Spring's beans generated are: Action, Bo (Business Object) and DAO classes. Fig. 11 shows this rule.

```
rule Operation2Bean{

  from
      c : UML!Operation
  to
      bean1 : NTiers!Bean (
            id <- c.name+c.class.name+'Action',
            class <- c.name+c.class.name+'Action',
            property <- Sequence{pr1}
                 ),
      bean2 : NTiers!Bean (
            id <- c.name+c.class.name+'DAO',
            class <- c.name+c.class.name+'DAOImpl',
            property <- Sequence{pr2}
                 ),
      bean3 : NTiers!Bean (
            id <- c.name+c.class.name+'Bo',
            class <- c.name+c.class.name+'BoImpl',
            property <- Sequence{pr3}
                 ),
      pr1 : NTiers!Property(
          name <-  c.name+c.class.name+'Action',
          ref <-  c.name+c.class.name+'Action'
              ),
      pr2 : NTiers!Property(
          name <- 'sessionFactory',
          ref <-  'sessionFactory'
              ),
      pr3 : NTiers!Property(
          name <-  c.name+c.class.name+'DAO',
          ref <-  c.name+c.class.name+'DAO'
              )
   }
```

Fig. 11. Rule that generates Action, Bo and DAO classes.

7. Transformation Rules Execution

In this section, we present a case study to demonstrate and exemplify our proposition. The UML class diagram of this case study represents the source model of our ATL transformation. The execution algorithm of ATL transformation allows browsing all transformation rules and thereafter generates the N-tiers web model. This latter is represented in Fig. 13.

7.1. Case Study

In this case study, we consider a system of a three classes. This system can manage the employee of a given department. The system classes are: the City class, the Department class and the Employee class. In this system, we use only the CRUD (Create, Retrieve, Update, and Delete) operations. These are most often implemented in all systems. In this case we can generate the model which can manage the requested layer from the layers already defined. Fig. 12 shows the UML class diagram of this system.

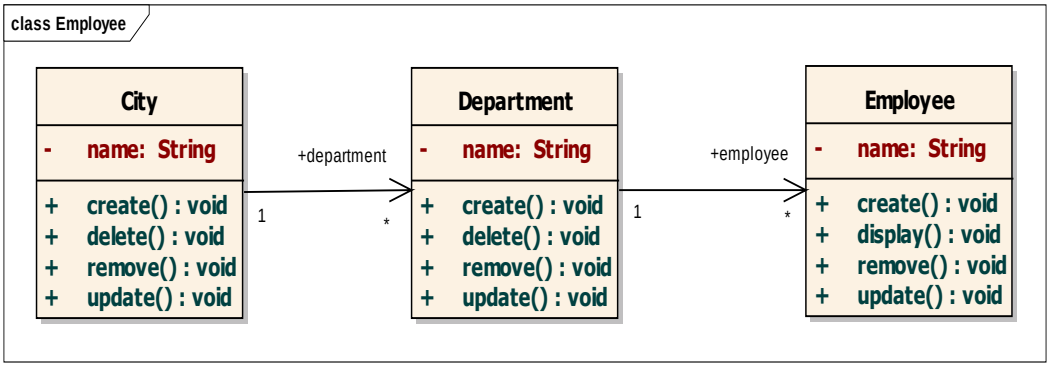

Fig. 12. UML class diagram of the employee management system.

7.2. ATL Transformation Result

The generated PSM model respects the n-tiers system architecture based on the patterns integration. Indeed, this model is composed of a set of Beans package (DAO, Action and Bo): Controller package, View package and hibernate mapping package. The beans package is composed of a set of Action, DAO and Bo classes. The controller package is composed of a set of Action classes. The view package represents the different jsp pages. The Hibernate mapping package represents the City, Department and Employee classes.

Fig. 13 shows the generated N-tiers Web model. This model contains the different ingredients for implementing an N-tiers system architecture respecting the architecture of MVC2, DI and DAO patterns.

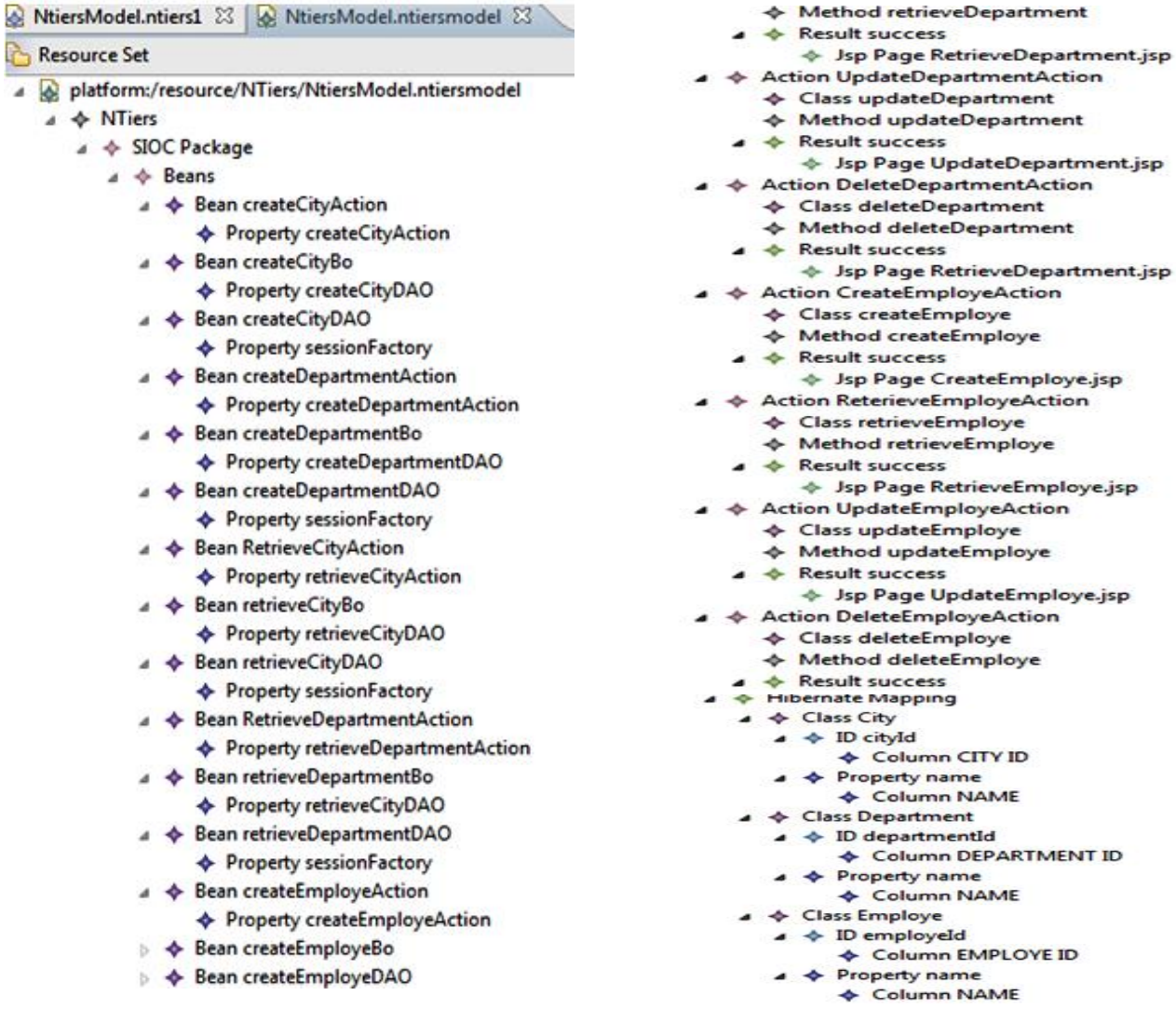

Fig. 13. The generated N-tiers web model.

8. Related Work

In this decade, several studies in model transformation and code generation have been conducted. The most relevant are: [11], [26]-[34].

The objective of the work [26] is to generate JSP pages and JavaBeans by applying the UWE [11], combined with the ATL transformation language [24]. The integration of AJAX into the UWE engineering process is considered as a future work of the authors.

In [27] Bézivin et al. can transform the PIMs models defined by Enterprise Distributed Object Computing into generated PSMs for different services platforms. The transformation rules corresponding PIMs to PSMs are written in ATL transformation language.

The work of Billing et al. [28] is dedicated to define the transformation rules permits to translate the PIM into PSM in the context of EJB by applying the approach by modeling based on QVT.

The objective of the work [29] is considered the MDA as a software industrialization pattern (or a software factory). The authors illustrate this work by a real case study in an IT services company. It is a proposal to create MDA tools based on XMI, XSLT and the Visitor pattern.

In other work, a model-driven development approach for E-Learning platform is proposed [30]. In this case the authors established the CIM model through the analysis of business logic. And after, they establish the robustness diagram of the system then the robustness analysis. Finally they proposed a transformation method from PIM to PSM layer by layer.

In [31], the objective is to introduce a new framework for the design of secure Data Warehouses based on MDA and QVT.

The AndroMDA approach has gained attention in the community of web-based MDA [32].The objective of

this work is to transform a PIM schemes to model by integrating a wide variety of scenarios and comes with a set of plugins, called cartridge.

The work [33] was conducted to generate the MVC 2 web model. In this work, the author appropriates the transformation rules by ATL transformation language in order to generate only the CRUD operations from three classes C_i, C_j and C_k.

Recently, the work [34] is arrived to generate the MVC 2 web model from the combination of the UML class diagram and the UML activity diagram. The principle idea of this combination is to stabilize the UML class diagram and to determine the input jsp page of each Action class. In [35], the authors generate the MVC2 web applications code through the model already generated in [34] by using JET2 template integrated in Eclipse project.

In this paper we have generalized the work presented in the articles [33]-[34]. This paper aims to integrate many frameworks and thereafter, by applying the ATL transformation language, we have generated an N-tiers web model instead of MVC2 web model in the case of [33]-[34].

9. Conclusion and Future Work

In this paper, we have applied the approach by modeling based on ATL transformation language to generate N-tiers web model from UML class diagram. In this case, we have began by modeling Spring IoC, Struts 2 and Hibernate DAO frameworks to obtain a N-tiers target meta-model through these frameworks integration. After modeling, we have defined the traceability links between the UML source meta-model and N-tiers target meta-model already obtained. The algorithm execution of ATL transformations allow browsing all transformation rules and generate N-tiers PSM model respecting the architecture of MVC2, DI and DAO patterns. The generated N-tiers PSM model is an EMF model. This file can be used to produce automatically the necessary target application code. Finally, the transformation result was demonstrated and exemplified by a case study.

Furthermore, we plan to generate an e-business web code from the generated n-tiers web model by applying the model-to-code (M2C) transformations. Our objective is to facilitate more and more the applications development. In other hand, we can extend this method for considering other frameworks like: PHP, Zend and DotNet.

References

[1] Strsus2 Homepage. (2013). Retrieved June 4, 2013, from http://www.struts.apache.org/2.x/.

[2] Hibernate Homepage. Retrieved June 7, 2013, from http://hibernate.org/.

[3] Spring Homepage. (2014). Retrieved June 4, 2013, from http://projects.spring.io/spring-framework/.

[4] Arnaud, C., Thierry, T., Julien, D., & Jean, P. R. (2009). *Spring Par La Pratique* (2nd Ed.). Eyrolles.

[5] Gerti, K., Birgit, P., Siegfried, R., & Werner, R. (2006). *Web Engineering*. John Wiley.

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

[7] Kleppe, A., Warmer, J., & Bast, W. (2003). *MDA Explained: The Model Driven Architecture: Practice and Promise*. Addison-Wesley Professional.

[8] Stefano, C., Piero F., Aldo, B., Marco, B., Sara, C., & Maristella, M. (2003). *Designing Data-Intensive Web Applications*. Morgan Kaufman.

[9] Jaime, G., & Cristina, C. (2002). *OO-H: Extending UML to Model Web Interfaces*. Information Modeling for Internet Applications. IGI Publishing.

[10] Pedro, V., Joan F., & Vicente, P. (2005). From web requirements to navigational design–A transformational approach. *Proceedings of the 5th Int. Conference of Web Engineering* (ICWE'05).

[11] Koch, N. (2006). Transformations techniques in the model-driven development process of UWE. *Proceedings of the 2nd Workshopsh of Model-Driven Web Engineering (MDWE'06)*. Palo Alto.

[12] Santiago, M., & Jaime, G. (2006). The Websa approach: Applying model driven engineering to web applications. *Journal of Web Engineering*, *5(2)*.

[13] Object Management Group (OMG). *Model Driven Architecture*. Retrieved December 12, 2013, from http://www.omg.org/mda/.

[14] OMG. (2013, October). *Unified Modeling Language*. http://www.uml.org/

[15] Meta Object Facility (MOF). Version 2.0, Retrieved January 20, 2006, from http://www.omg.org/spec/MOF/2.0/PDF/.

[16] XML Metadata Interchange (XMI). Version 2.1.1. Retrieved December 20, 2013, from http://www.omg.org/spec/XMI/.

[17] Object Management Group (OMG). (2014). *UML 2 Object Constraint Language (OCL)*. Retrieved December 10, 2012, from http://www.omg.org/cgibin/doc?ptc/2005-06-06.

[18] Pan, Y., & Wang, J. l. (2010). Design and implementation of accounting e-business platform for source documents. *Proceedings of the International Conference on Computer Application and System Modeling (ICCASM 2010).*

[19] Spring AOP homepage. Retrieved November 8, 2013, from http://static.springsource.org/spring/docs/2.5.x/reference /aop.html.

[20] Praveen G., & Govil, M. C. (2010). Spring web MVC Framework for rapid open source J2EE application development: A case study. *International Journal of Engineering Science and Technology*, *2(6), 1684-1689.*

[21] Praveen, G., & Govil, P. M. C. (2010). MVC design pattern for the multi framework distributed applications using XML, spring and struts framework. *International Journal on Computer Science and Engineering, 2(4), 1047-1051.*

[22] MENET, L. (2010). Formalisation d'une approche d'Ingénierie Dirigée par les modèles appliquée au domaine de la gestion des données de reférence. PhD thesis, Université de Paris VIII, Laboratoire d'Informatique Avancée de Saint-Denis (LIASD), école doctorale : Cognition Langage Interaction (CLI).

[23] Frédéric J., & Ivan, K. (2006). Transforming models with ATL. *Proceedings of MoDELS 2005 Workshops, LNCS 3844, (pp. 128 – 138),* Springer-Verlag Berlin Heidelberg.

[24] Jouault, F., Allilaire, F., Bézivin, J., & Kurtev, I. (2008). *A Model Transformation Tool*. Sciences of Computer Programming-Elseiver, *72(1-2), 31–39.*

[25] AtlanMod. (2013). Atl transformation language home page. Retrieved December 20, 2013, from http://www.eclipse.org/m2m/atl/.

[26] Kraus, A. K., & A., Koch, N. (2007). Model-driven generation of web applications in UWE. *Proceedings in the 3rd International Workshop on Model-Driven Web Engineering, CEUR-WS*, *Vol. 261.*

[27] Bezivin, J., Hammoudi, S., Lopes, D., & Jouault, F. (2004). Applying MDA approach for web service platform. *Proceedings of the 8th IEEE International Entreprise Distributed Object Computing Conference (EDOC'04)* (pp. 58-70).

[28] Bezivin, J., Busse, S., Leicher, A., & Suss, J. G. (2004). Platform independent model transformation Based on triple. *Proceedings of the 5th ACM/IFIP/USENIX International Conference on Middleware (pp. 493-51).*

[29] Ndie, T. D., Tangha, C., & Ekwoge, F. E. (2010). MDA (model- driven architecture) as a software industrialization pattern: An approach for a pragmatic software factories. *Journal of Software Engineering & Applications*, *3 (6), 561-571.*

[30] Cong, X., Zhang, H., Zhou, D., Lu, P., & Qin, L. (2010). A model-driven architecture approach for developing E-learning platform, entertainment for education. *Proceeding of Digital Techniques and Systems Lecture Notes in Computer Science* (pp. 111-122).

[31] Soler, E., Trujillo, J., Blanco, C., & Fernandez M. E. (2009). Designing secure data warehouses by using MDA and QVT. *Journal of Universal Computer Science, 15(8), 1607-1641.*

[32] AndroMDA. (2013). Retrieved October 10, 2013 from, http: //www.andromda.org/.

[33] Rahmouni, M., & Mbarki, S. (2011). MDA-based ATL transformation to generate MVC 2 web models. *International Journal of Computer Science and Information Technology, 3(4), 57-70.*

[34] Rahmouni, M., & Mbarki, S. (2013). Combining uml class and activity diagrams for mda generation of mvc 2 web applications. *International Review in Computers and Software*, *8(4), 949-957.*

[35] Rahmouni, M., & Mbarki, S. (2013). An end-to-end code generation from uml diagrams to mvc2 web applications. *International Review in Computers and Software (IRECOS), 8(9), 2123-2135.*

M'hamed Rahmouni received his diploma of higher approfondie studies in computer science and telecommunication from the Faculty of Science, Ibn Tofail University, Morocco, in 2007, and doctorat of high graduate studies degrees in computer sciences from Ibn Tofail University, Morocco, in 2015 . He participated in various international congresses in MDA (Model Driven Architecture) integrating new technologies XML, EJB, MVC, Web Services, etc. and he published many papers in the MDA domain.

Samir Mbarki received his B.S. degree in applied mathematics from Mohammed V University, Morocco, in 1992, and doctorat of high graduate studies degrees in computer sciences from Mohammed V University, Morocco in 1997. In 1995 he joined the faculty of science Ibn Tofail University, Morocco where he is currently a professor in Department of Computer Science. His research interests include software engineering, model driven architecture, software metrics and software tests. He obtained an HDR in computer science from Ibn Tofail University in 2010.