Available online at www.elixirpublishers.com (Elixir International Journal)

Computer Engineering

Elixir Comp. Engg. 87 (2015) 35584-35590

ISSN: 2229-712X

Model-To-Model Transformation with approach by modeling: From UML to IoC Application model

Redouane Esbai and Mohammed Erramdani
MATSI Laboratory, ESTO, Mohammed First University, Oujda, Morocco.

ARTICLE INFO

Article history:
Received: 18 August 2015;
Received in revised form:
26 September 2015;
Accepted: 03 October 2015;

Keywords
MDA,
Model-To- Model,
Tranformation Rules,
Inversion of Control.

ABSTRACT

The continuing evolution of business needs and technology makes applications more demanding in terms of development, maintenance, usability and management. To cope with this complexity, various frameworks and patterns are integrated for producing stable, maintainable and testable code. Given this diversity of solutions, the generation of a code based on UML models has become important. This paper presents, after establishing the different meta-models, the application of the MDA (Model Driven Architecture) to generate, from the UML model, the Code following the Ioc (Inversion of Control) and Dao (Data Acces Object) patterns. The model-to-model transformations are also clearly and formally established by using the standard MOF 2.0 QVT (Meta-Object Facility 2.0 Query-View-Transformation) as transformation language. The transformation rules defined in this paper can generate, from the class diagram, an XML file containing the Business and the Data Access package. This file can be used to generate the necessary code of an architecture overview of IoC and DAO patterns.

Introduction

In recent years many organizations have begun to consider MDA as an approach to design and implement enterprise applications.

The central idea of MDA is to separate the platform independent design from the platform specific implementation of applications delaying as much as possible the dependence on specific technologies [1]. The MDA uses models as first class entities, enabling the definition and automatic execution of transformations between models and from models to code. The creation of meta-models for specifying modeling languages is a basic task in MDA. Models in MDA are the key artifacts in all phases of development and are mostly expressed with Unified Modeling Language (UML). Also the specification of transformations between models, are called model-to-model (M2M) transformations, and from model to code, are called model-to-text (M2T) transformations. The main advantage of this approach of software development is that MDA tools enable these transformations to be specified and executed automatically, using supporting languages and tools for MDA.

In software engineering, inversion of control (IoC) is a programming technique in which object coupling is bound at run time by an assembler object and is typically not known at compile time using static analysis.

Inversion of control as a design guideline serves the following purposes:

- There is a decoupling of the execution of a certain task from implementation.
- Every module can focus on what it is designed for.
- Modules make no assumptions about what other systems do but rely on their contracts.
- Replacing modules has no side effect on other modules.

In a recent work [2], the authors have developed a source and a target meta-model. The first was a PIM meta-model specific to class diagrams. The second was a PSM meta-model for MVP (Model-View-Presenter) web applications (particularly GWT), then they have elaborated transformation rules using the approach by modeling. The purpose of our contribution is to produce and generate automatically an IoC and DAO PSM model, from the class diagram. In this case, we elaborate a number of transformation rules using the approach by modeling and MOF 2.0 QVT, as transformation language, to permit the generation of an XML file that can be used to produce the required code of the target application. The advantage of this approach is the bidirectional execution of transformation rules.

This paper is organized as follows: related works are presented in the second section, the third section defines the MDA approach, and the fourth section presents IoC model and its implementation as a framework. The transformation language MOF 2.0 QVT is the subject of the fifth section. In the sixth section, we present the UML, DI and DAO meta-models. In the seventh section, we present the transformation rules using MOF 2.0 QVT from UML source model to the IoC target.

Relaed Works

Many researches on MDA and generation of code have been conducted in recent years. The most relevant are [3][4][5][6][7][8][9][10][11][12][13][14][15][20][36][38][39].

In [20] Bennitt, et al., allows code generation from models to aspects in AspectJ, a java implementation of aspect-oriented programming (AOP). The transformation among models is accomplished by means of Extensible Markup Language (XML) specifications and meta-models of XML and AspectJ. The code is generated from the XML specifications and the aspects are controlled in the system by throwing and handling exceptions.

The authors of the work [3] show how to generate JSPs and JavaBeans using the UWE [4], and the ATL transformation language [5][36]. Among future works cited, the authors considered the integration of AJAX into the engineering process of UWE.

Two other works followed the same logic and have been the subject of two works [6][7]. A meta-model for Ajax was defined using AndroMDA[37] tool. The generation of Ajax code has been illustrated by an application CRUD (Create, Read, Update, and Delete) that manages people.

Tele:
E-mail addresses: es.redouane@gmail.com

Meliá, Pérez and Díaz propose in [8] a new approach called OOH4RIA which proposes a model driven development process that extends OOH methodology. It introduces new structural and behavioral models in order to represent a complete RIA and to apply transformations that reduce the effort and accelerate its development. In another work [9] they present a tool called OIDE (OOH4RIA Integrated Development Environment) aimed at accelerating the RIAs development through the OOH4RIA approach which establishes a RIA-specific model-driven process.

The Web Modeling Language (WebML) [10] is a visual notation for specifying the structure and navigation of legacy web applications. The notation greatly resembles UML class and Entity-Relation diagrams. Presentation in WebML is mainly focused on look and feel and lacks the degree of notation needed for AJAX web user interfaces [11][12].

Nasir, Hamid and Hassan [13] have presented an approach to generate a code for the dotNet application Student Nomination Management System. The method used is WebML and the code was generated by applying the MDA approach, but the creation was not done according to the dotNet MVC2 logic.

This paper aims to rethink and complete the work presented in [2][14], by applying the standard MOF 2.0 QVT to develop the transformation rules aiming at generating the DI and DAO target model.

Model Driven Architecture (MDA)

In November 2000, OMG, a consortium of over 1 000 companies, initiated the MDA approach. The key principle of MDA is the use of models at different phases of application development. Specifically, MDA advocates the development of requirements models (CIM), analysis and design (PIM) and code (PSM).

The MDA architecture [16] is divided into four layers. In the first layer, we find the standard UML (Unified Modeling Language), MOF (Meta-Object Facility) and CWM (Common Warehouse Meta-model). In the second layer, we find a standard XMI (XML Metadata Interchange), which enables the dialogue between middlewares (Java, CORBA, .NET and web services). The third layer contains the services that manage events, security, directories and transactions. The last layer provides frameworks which are adaptable to different types of applications namely Finance, Telecommunications, Transport, medicine, E-commerce and Manufacture, etc.).

The major objective of MDA [1] is to develop sustainable models; those models are independent from the technical details of platforms implementation (JavaEE, .Net, PHP or other), in order to enable the automatic generation of all codes and applications leading to a significant gain in productivity. MDA includes the definition of several standards, including UML [17], MOF [18] and XMI [19].

The IoC pattern

In traditional programming, the flow of the business logic is determined by objects that are statically assigned to one another. With inversion of control, the flow depends on the object graph that is instantiated by the assembler and is made possible by object interactions being defined through abstractions. The binding process is achieved through dependency injection, although some argue that the use of a service locator also provides inversion of control.

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 [21].

In other words, Dependency Injection (DI) 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 [27], Symfony dependency injection [29], Spring.NET [28], EJB [31], PicoContainer [30]. (We have used some Spring classes in our source meta-model).

In object-oriented programming, there are several basic techniques to implement inversion of control. These are:

1. using a factory pattern
2. using a service locator pattern
3. using a dependency injection of any given below type:

- a constructor injection
- a setter injection
- an interface injection

IoC in Spring framework

In Spring framwork, the Inversion of Control (IoC) principle is implemented using the Dependency Injection (DI) design pattern.

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.

Figure 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.

Reference: Represents the reference of an element.

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

Approach by modeling

Currently, the models' transformations can be written according to three approaches: The approach by Programming, the approach by Template and the 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 [32]. 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 [33].

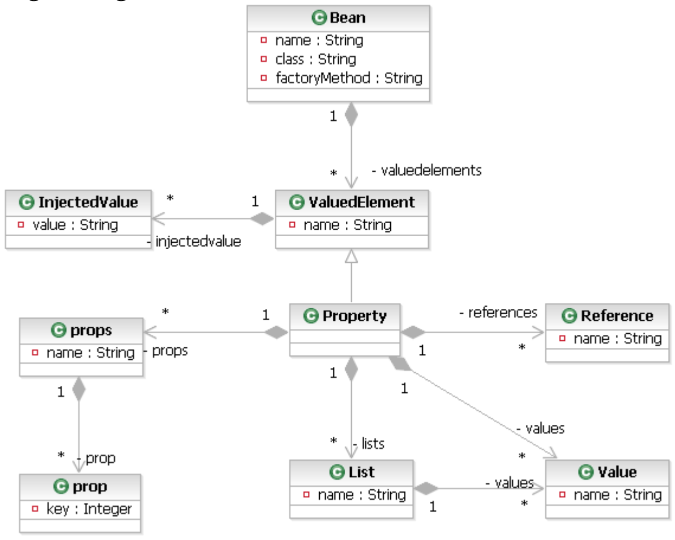


Figure 1. Spring IoC Meta-model

Figure 2 illustrates the approach by modeling. Models transformation is defined as a model structured according to MOF 2.0 QVT meta-model. The MOF 2.0 QVT meta-model expresses some structural correspondence rules between the source and target meta-model of a transformation [34]. This model is a perennial and productive model that is necessary to transform in order to execute the transformation on an execution platform.

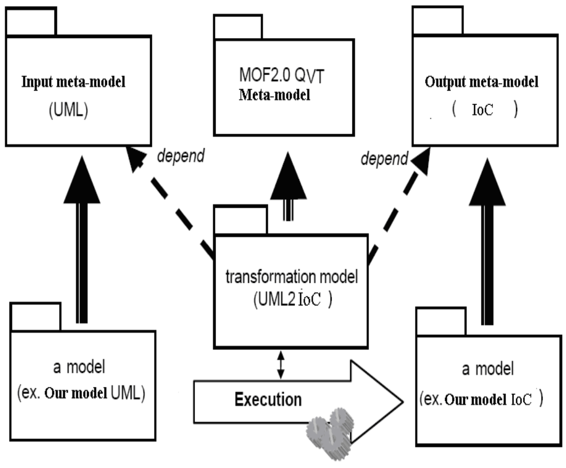


Figure 2. Approach by Modeling

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 (see Figure 3).

The declarative part of QVT is defined by Relations and Core languages, with different levels of abstraction. Relations are a user-oriented language for defining transformations in a high level of abstraction. It has a syntax text and graphics. Core language forms the basic infrastructure for the declaration part; this is a technical language of lower level determined by textual syntax. It is used to specify the semantics of Relations language in the form of a Relations2Core transformation. The declarative vision comes through a combination of patterns, source and target side to express the transformation.

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) and constructs in OCL edge effect.

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.

Finally, QVT suggests a second extension mechanism for specifying transformations invoking the functionality of transformations implemented in an external language Black Box.

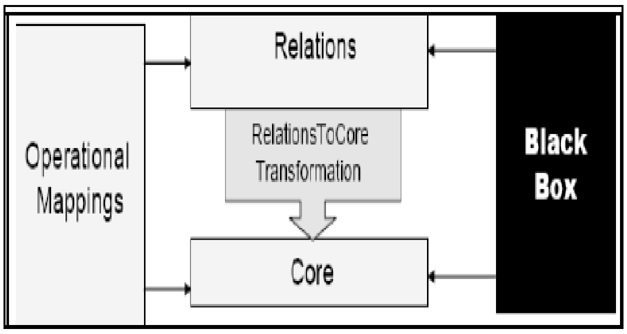


Figure 3. The QVT Structure

This work uses the QVT-Operational mappings language implemented by Eclipse modeling [35].

OCL (Object Constraint Language)

Object Constraint Language (OCL) is a formal language used to describe expressions on UML models.

These expressions typically specify invariant conditions that must hold for the system being modeled or queries over objects described in a model. Note that when the OCL expressions are evaluated, they do not have side effects. OCL expressions can be used to specify operations / actions that, when executed, do alter the state of the system. UML modelers can use OCL to specify application-specific constraints in their models.

In MOF 2.0 QVT, OCL is extended to Imperative OCL as part of QVT Operational Mappings.

Imperative OCL added services to manipulate the system states (for example, to create and edit objects, links and variables) and some constructions of imperative programming languages (for example, loops and conditional execution). It is used in QVT Operational Mappings to specify the transformations.

QVT defines two ways of expressing model transformations: declarative and operational approaches.

The declarative approach is the Relations language where transformations between models are specified as a set of relationships that must hold for successful transformation.

The operational approach allows either defining transformations using a complete imperative approach or complementing the relational transformations with imperative operations, by implementing relationships.

Imperative OCL adds imperative elements of OCL, which are commonly found in programming languages like Java. Its semantics are defined in [32] by a model of abstract syntax. The complete abstract syntax ImperativeOCL is shown in Figure 4.

The most important aspect of the abstract syntax is that all expression classes must inherit OclExpression.

OclExpression is the base class for all the conventional expressions of OCL. Therefore, Imperative Expressions can be used wherever there is OclExpressions.

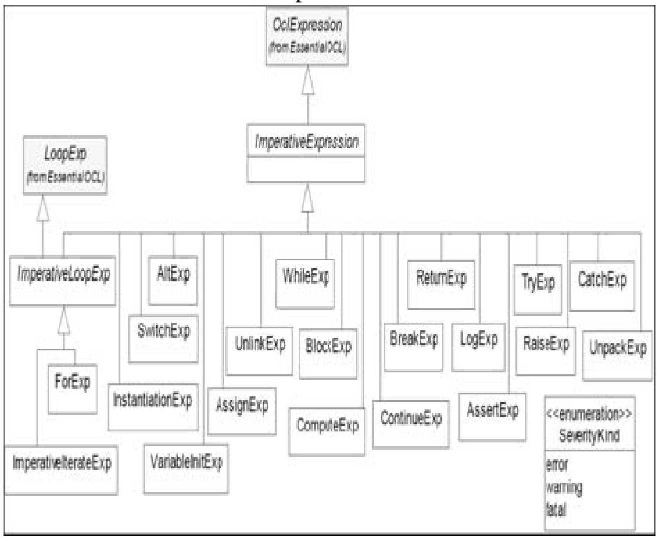


Figure 4. Imperative Expressions of ImperativeOCL

UML, DI and DAO meta-models

To develop the algorithm of transformation between the source and target model, we present in this section, the different meta-classes forming the UML source meta-model and the IoC target meta-model. The meta-model source structure simplified UML model based on a package containing the data types and classes. These classes contain properties typed and characterized by multiplicities (upper and lower). The classes contain operations with typed parameters. Figure 5 shows the source meta-model:

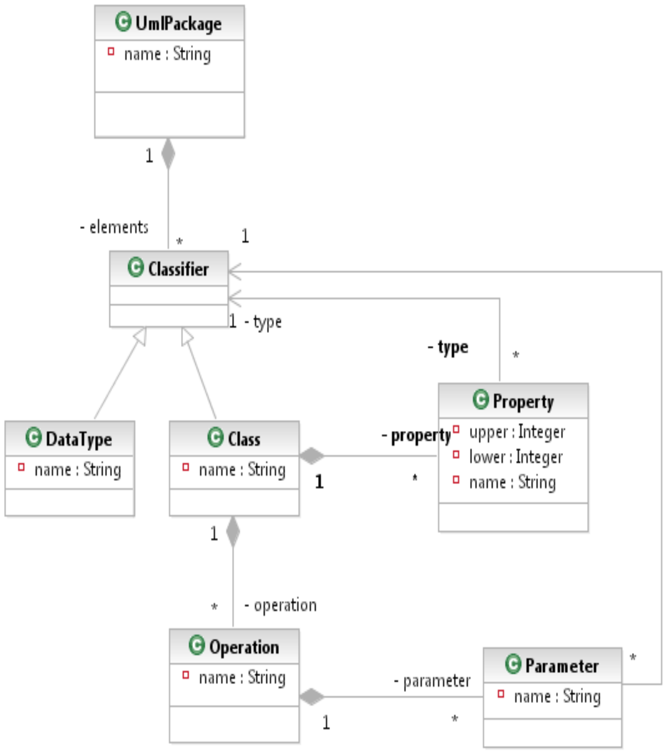


Figure 5. Simplified UML meta-model

DI and DAO target meta-models

Our target meta-model is composed of two essential part. Figure 6 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:

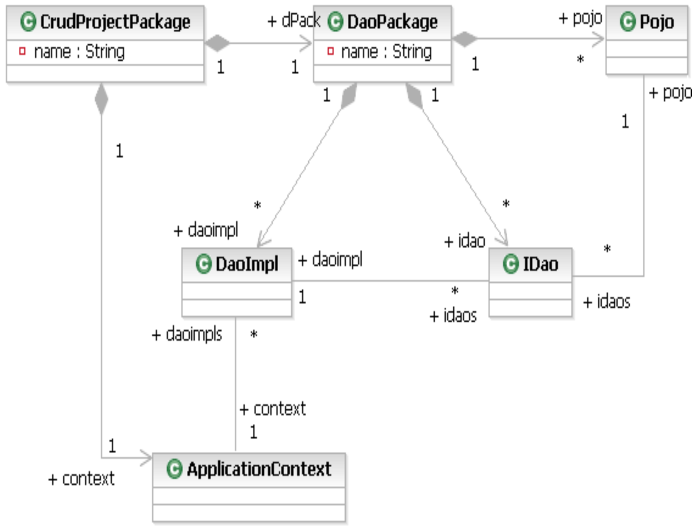


Figure 6. Simplified meta-model of DaoPackage

- **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.
- **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.

Figure 7 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 DI pattern. Here, we present the different meta-classes to express the concept of DI contained in the Business Package.

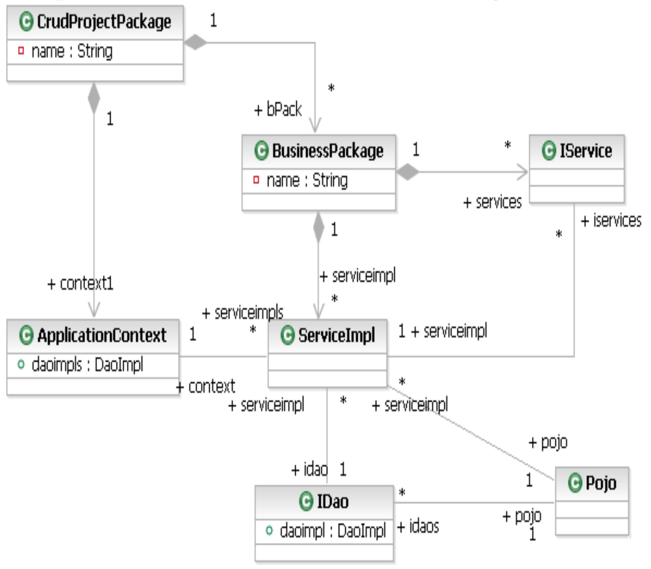


Figure 7. Simplified meta-model of a BusinessPackage

This meta-model structures the models representing the business logic of the target application.

- **BusinessPackage:** represents the package which contains the different meta-classes to express the concept of the business logic of target application.
- **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)
- **Pojo:** (already seen at the DaoPackage meta-model)

The Process of Transforming UML Source Model to IoC Target Model

CRUD operations (Create, Read, Update, and Delete) 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 ECORE 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 8). After applying the transformation on the UML model, composed by the class Employee, we generated the target model (see Figure 10).

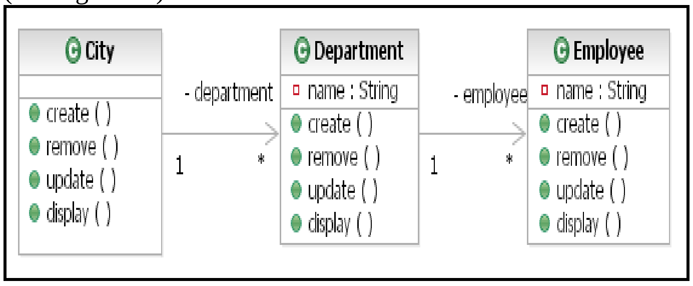


Figure 8. UML instance model

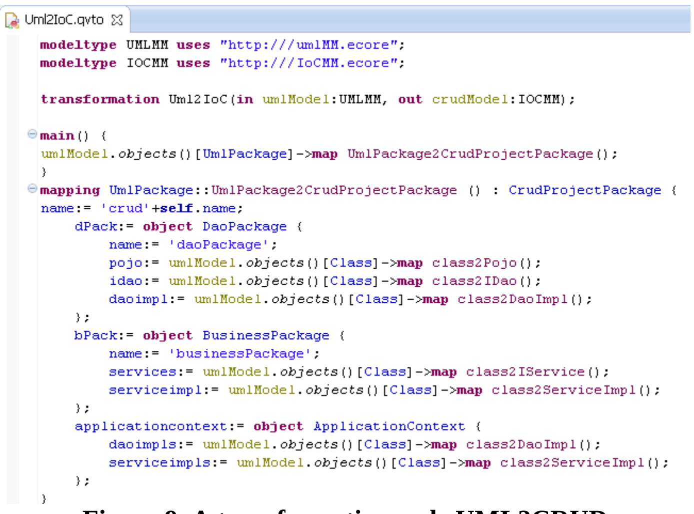
Transformation rules

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
 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 e ∈ source model
  x = transformationRuleTwoIService(e)
  link x to bp
  x= transformationRuleTwoSrviceImpl(e)
  link x to bp
 end for
 create ApplicationContext applicationContext;
 link applicationContext to crud
 link dp to crud
 link bp to crud
 create object ApplicationContext applicationContext;
 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
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):ServiceImpl
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
```

Figure 9 represents the first part of the code of the transformation of UML model source to IoC target model.



Uml2IoC.qvto

```
modeltype UMLMM uses "http:///umlMM.ecore";
modeltype IOCMM uses "http:///IoCMM.ecore";

transformation Uml2IoC(in umlModel:UMLMM, out crudModel:IOCMM);

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();
        idao:= umlModel.objects()[Class]->map class2IDao();
        daoimpl:= umlModel.objects()[Class]->map class2DaoImpl();
    };
    bPack:= object BusinessPackage {
        name:= 'businessPackage';
        services:= umlModel.objects()[Class]->map class2IService();
        serviceimpl:= umlModel.objects()[Class]->map class2ServiceImpl();
    };
    applicationcontext:= object ApplicationContext {
        daoimpls:= umlModel.objects()[Class]->map class2DaoImpl();
        serviceimpls:= umlModel.objects()[Class]->map class2ServiceImpl();
    };
}
```

Figure 9. A transformation code UML2CRUD

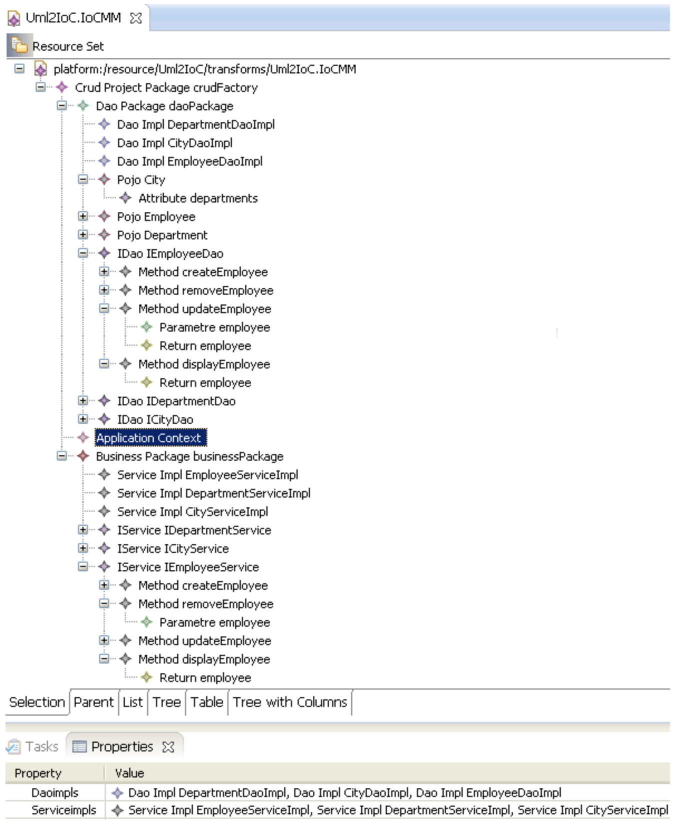
The transformation uses, in entry, a model of the UML type named umlModel, and in output a model of the IoC named crudModel.

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 CrudProjectPackage, by creating the elements of type package 'Dao' and 'Business'. It is a question of transforming each class of package UML 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 and creating the Spring ApplicationContext.

Result

Figure 10 shows the result after applying the transformation rules.



Uml2IoC.IoCMM

Resource Set

```
platform:/resource/Uml2IoC/transforms/Uml2IoC.IoCMM
  Crud Project Package crudFactory
    Dao Package daoPackage
      Dao Impl DepartmentDaoImpl
      Dao Impl CityDaoImpl
      Dao Impl EmployeeDaoImpl
      Pojo City
        Attribute departments
      Pojo Employee
      Pojo Department
      IDao IEmployeeDao
        Method createEmployee
        Method removeEmployee
        Method updateEmployee
          Parametre employee
          Return employee
        Method displayEmployee
          Return employee
      IDao IDepartmentDao
      IDao ICityDao
    Application Context
    Business Package businessPackage
      Service Impl EmployeeServiceImpl
      Service Impl DepartmentServiceImpl
      Service Impl CityServiceImpl
      IService IDepartmentService
      IService ICityService
      IService IEmployeeService
        Method createEmployee
        Method removeEmployee
          Parametre employee
        Method updateEmployee
        Method displayEmployee
          Return employee
```

Property	Value
Daoimpls	Dao Impl DepartmentDaoImpl, Dao Impl CityDaoImpl, Dao Impl EmployeeDaoImpl
Serviceimpls	Service Impl EmployeeServiceImpl, Service Impl DepartmentServiceImpl, Service Impl CityServiceImpl

Figure 10. Generated PSM IoC model

The first element in the generated PSM model is CrudFactory that contains ApplicationContext.xml file, DAO Package and Business Package.

The second element in the generated PSM model is DaoPackage which includes three DAOs' interfaces, three DAOs' implementations and three Pojos' objects that contains their attributes and correspond to the three objects 'city', 'employee' and 'department'. The last element in the generated PSM model is BusinessPackage which contains three services' interfaces that contains methods with their parameters and their implementations.

Conclusion and perspectives

In this paper, we applied the MDA to generate, from the UML class diagram, the code following the DI and DAO patterns for a SpringIoC Framework.

After modeling, we have defined the traceability links between the UML source meta-model and SpringIoC target meta-model already obtained. The algorithm execution of QVTo transformations allow browsing all transformation rules and generate Spring framework PSM model respecting the architecture of DI and DAO patterns. The generated SpringIoC PSM model is an EMF model. This file can be used to produce automatically the necessary target application code. Finally, 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, we plan to generate a code from the generated SpringIoC model by applying the model-to-Text (M2T) 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 JPA, JBossSeam and DotNet.

References

[1] Pastor, O.,Molina J.C, Model-Driven Architecture in Practice: A Software Production Environment Based on Conceptual Modeling (New York: Springer-Verlag, 2007).

[2] Esbai, R., Erramdani, M., Mbarki, S., Model-Driven Transformation for GWT with approach by Modeling:From UML model to MVP web applications, International Review on Computers and Software (I.RE.CO.S.), Vol. 9. n. 9, pp. 1612-1620, September 2014.

[3] 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).

[4] 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

[5] 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.

[6] 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).

[7] 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)

[8] Meliá S., Gómez J., Pérez P., Díaz O., A Model-Driven Development for GWT-Based Rich Internet Applications with

OOH4RIA, Proceedings of ICWE '08. Eighth International Conference on, Yorktown Heights, NJ, (Page: 13, Year of publication: 2008, ISBN: 978-0-7695-3261-5).

[9] Meliá S., Gómez J., Pérez S., Diaz O. Facing Architectural and Technological Variability of Rich Internet Applications. IEEE Internet Computing, vol. 99, pp.30-38, 2010.

[10] S. Ceri, P. Fraternali, and A. Bongio. Web modeling language (WebML): a modeling language for designing web sites. Computer Networks, vol. 33(1-6) pp137–157, 2000.

[11] Preciado J. Carlos, M. Linaje, S. Comai, and F. Sanchez-Figueroa. Designing Rich Internet Applications with Web engineering methodologies. Proceedings of the 9th IEEE International Symposium on Web Site Evolution (WSE'07)(Page: 23 Year of publication: 2007).

[12] Trigueros M. L., J. C. Preciado, and F. S´anchez-Figueroa. A method for model based design of Rich Internet Application interactive user interfaces. In ICWE'07: Proceedings of the 7th International Conference Web Engineering (page: 226 Year of publication: 2007).

[13] 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

[14] Esbai. R, Erramdani, M., Mbarki, S., Arrassen. I, Meziane. A. and Moussaoui. M., Model-Driven transformation with approach by modeling: From UML to N-tiers Web Model, International Journal of Computer Science Issues (IJCSI) , Vol. 8, Issue 3, May 2011, ISSN (Online): 1694-0814

[15] Esbai. R, Erramdani, M., Mbarki, S., Arrassen. I, Meziane. A. and Moussaoui. M., Transformation by Modeling MOF 2.0 QVT: From UML to MVC2 Web model, InfoComp - Journal of Computer Science, vol. 10, no. 3, p. 01-11, September of 2011, ISSN 1807-4545.

[16] Miller, J., Mukerji, J., al. MDA Guide Version 1.0.1 (OMG, 2003).

[17] UML Infrastructure Final Adopted Specification, version 2.0, September 2003, http://www.omg.org/cgi-bin/doc?ptc/03-09-15.pdf

[18] Meta Object Facility (MOF), version 2.0 (OMG, 2006)

[19] XML Metadata Interchange (XMI), version 2.1.1 (OMG, 2007),

[20] J. Bennett, K. Cooper and L. Dai, "Aspect-Oriented Model-Driven Skeleton Code Generation: A Graph-Based Transformation Approach, Science of Computer Programming-Elseiver. vol. 75, no. 8, (2010), pp. 689-725.

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

[22] Echo2 source web site (http://echopoint.sourceforge.net/)

[23] Harris, Robert; Warner, Rob, The Definitive Guide to SWT and JFACE (1st ed.), (Apress, 2004).

[24] Vaadin Framework web site (https://vaadin.com/home)

[25] ZK framework web site (http://www.zkoss.org)

[26] Nucleo .NET framework web site (http://nucleo.codeplex.com/)

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

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

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

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

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

[32] Meta Object Facility (MOF) 2.0 Query/View/Transformation (QVT), Version 1.1 (OMG, 2009).

[33] Czarnecki, K., Helsen, S., Classification of Model Transformation Approaches, Proceedings of the 2nd OOPSLA'03 Workshop on Generative Techniques in the Context of MDA. Anaheim (Year of publication: 2003).

[34] Philip Langer, Manuel Wimmer, Gerti Kappel, Model-to-Model Transformations By Demonstration, Theory and Practice of Model Transformations, LNCS Volume 6142, 2010, pp 153-167 (Springer Berlin Heidelberg, 2010)

[35] Eclipse modeling, http://www.eclipse.org/modeling/.

[36] Jouault, F. Kurtev, I.: Transforming models with ATL. In: Bruel, J.-M. (ed.) MoDELS 2005, LNCS, vol. 3844, pp. 128-138. Springer, Heidelberg (2006)

[37] AndroMDA web site (http://www.andromda.org/).

[38] Dennis Wagelaar,Viviane Jonckers, Explicit Platform Models for MDA, Model Driven Engineering Languages and Systems, LNCS Volume 3713, 2005, pp 367-381 (Springer Berlin Heidelberg, 2005).

[39] Kevin Lano,Shekoufeh Kolahdouz-Rahimi, Model-Driven Development of Model Transformations, Theory and Practice of Model Transformations, LNCS Volume 6707, 2011, pp 47-61, (Springer Berlin Heidelberg 2011).