Transformation ATL pour la génération de modèles Web MVC 2

Samir MBARKI *, M'hamed RAHMOUNI *, Mohammed ERRAMDANI **

* Laboratoire CIROS
Faculté des Sciences
Université Ibn Tofaïl
B.P. 133, KENITRA
MAROC
mbarkisamir@hotmail.com
md.rahmouni@yahoo.fr

** Département de Management
Ecole Supérieure de Technologie
Université Mohammed Premier
OUJDA
MAROC
mramdani69@yahoo.co.uk

RÉSUMÉ. De nos jours, les applications Web sont devenues de plus en plus complexes. Une complexité due à la richesse du contenu et de la présentation, mais également à la structure des entités définies. Pour assurer l'évolutivité de telles applications, plusieurs frameworks (supportant le pattern MVC 2) ont été élaborés. Face à cette diversité de frameworks et l'amélioration continue de la technologie Web, nous souhaitons développer un outil de génération de code à partir de diagrammes UML. L'objectif de ce papier est d'appliquer une approche MDA assurant le passage d'un modèle UML vers un modèle Web MVC 2. Pour cela, nous avons élaboré deux méta-modèles PIM et PSM. Afin d'assurer la traçabilité entre ces deux méta-modèles, nous avons défini des règles de transformation en langage ATL.

ABSTRACT. Web applications have the complexity of designing, developing, maintaining and managing. These systems have increased significantly, as well. To cope with this complexity, several frameworks (supporting the MVC 2 pattern) have been elaborated. Facing this diversity and incessant improvement of Web technology, we are in need of developing a tool which is able to produce the code from the UML models. In this paper, we apply MDA approach for generating PSM from UML design to MVC 2 Web implementation. That is why we have developed two metamodels handling UML class diagrams and MVC 2 Web applications, then we have to set up transformation rules. These last are expressed in ATL language.

MOTS-CLÉS : MDA, Méta-modèles, Règles de transformation, ATL, Développement d'applications Web.

KEYWORDS: MDA, Meta-models, Transformation rules, ATL, Web Applications Development.

1. Introduction

L'industrie de développement de logiciels doit faire face à des enjeux économiques, stratégiques et techniques. Récemment les applications Web sont devenues de plus en plus complexes et les besoins de l'utilisateur ne cessent d'évoluer. Pour rester compétitives, les entreprises doivent réduire de façon significative leurs coûts de développement et de maintenance. Le MDA (Model-Driven Architecture) est une nouvelle discipline du génie logiciel qui a émergé pour assurer le développement de modèles productifs [2] [12]. L'approche MDA est de plus en plus appliquée. Beaucoup de praticiens s'y sont adhérés et un bon nombre d'outils supportant le MDA a été développé. D'autre part, beaucoup d'implémentations du pattern MVC 2, dans le domaine des applications Web, ont été effectuées. Citons à ce niveau les frameworks suivants : Struts [4], Spring MVC [3], PHP.MVC [7], Zend [9], PureMVC [8]. Parmi ces frameworks, Struts a atteint une maturité et a ainsi gagné la confiance des développeurs.

Nous présentons dans ce papier un outil qui transforme un diagramme de classes muni des opérations CRUD vers un modèle cible (particulièrement struts). Le résultat de cette transformation est un fichier XMI, qui, par la suite, peut être utilisé comme modèle source afin de produire le code d'une application Web MVC 2. Cet outil permet d'afficher, supprimer, modifier et ajouter les différents objets du système d'information. L'accent est mis sur les associations entre les classes. La transformation est exprimée en langage ATL (Atlas Transformation Language) [1] [5] [6].

Le reste de ce papier est organisé de la manière suivante : dans la section 2, nous allons présenter l'architecture MDA, les langages de méta-modélisation et de transformation. Dans la section 3, nous allons élaborer nos deux méta-modèles UML et Web MVC 2. Dans la section suivante, nous allons développer les règles de transformation. La section 5 fait l'objet de l'implémentation et l'exécution. Nous terminons par une conclusion.

2. Architecture dirigée par les modèles (MDA)

L'architecture du MDA se découpe en quatre couches. Dans la première couche, se trouvent le standard UML (Unified Modeling Language), MOF (Meta-Object Facility) et CWM (Common Warehouse Metamodel). Dans la couche suivante, se trouve le standard XMI (XML Metadata Interchange) qui facilite les échanges de modèles et leur stockage. La troisième couche contient les services qui permettent de gérer les événements, la sécurité et les transactions. Enfin, la dernière couche propose les

frameworks adaptables à différentes types d'applications (Finance, Télécom, Transport, Médecine, etc.). [2] [12]

Le principe clé de MDA consiste en l'utilisation de modèles aux différentes phases du cycle de développement d'une application. Plus précisément, MDA préconise l'élaboration de modèles d'exigence (CIM), d'analyse et de conception (PIM) et de code (PSM). L'objectif majeur du MDA est l'élaboration de modèles pérennes, indépendants des détails techniques d'implémentation, afin de permettre la génération automatique de la totalité du code des applications et d'obtenir un gain significatif de productivité. Dans la suite de cette section, nous allons présenter les principaux artefacts de l'ingénierie des modèles, les langages exprimant les méta-modèles et les transformations de modèles :

MOF (Meta Object Facility) a été adopté par l'OMG en 1997. La spécification de MOF définit un langage abstrait et un framework pour la spécification, la construction et la gestion des méta-modèles génériques. De plus, MOF définit une plateforme pour l'implémentation de modèles décrits par les méta-modèles [13].

ECORE est un langage de méta-modélisation qui fait partie d'EMF (Eclipse Modeling Framework) et qui est le résultat des efforts du projet ETP (Eclipse Tools Project). EMF est un framework de modélisation et génération de code pour supporter la création d'outils et d'applications dirigées par les modèles. ECORE définit des éléments principaux comme : *EnamedElement*, *EClassifier*, *ETypedElement*, *EPackage*, *EClass*, *EDataType*, *EAttribute*, *EReference*, *EOperation*.

La transformation de modèles joue un rôle prépondérant dans l'ingénierie dirigée par les modèles. A cet effet, plusieurs travaux ont été menés dans le but de définir des langages de transformation assurant efficacement la traçabilité entre les différents types de modèles MDA. ATL (Atlas Transformation Language) est le langage de transformation développé dans le cadre du projet ATLAS [5]. ATL est développé au LINA à Nantes par l'équipe de Jean Bézivin. Il fait partie du projet Eclipse M2M (Model-to-Model). La figure 1 résume la place d'ATL dans l'architecture MDA.

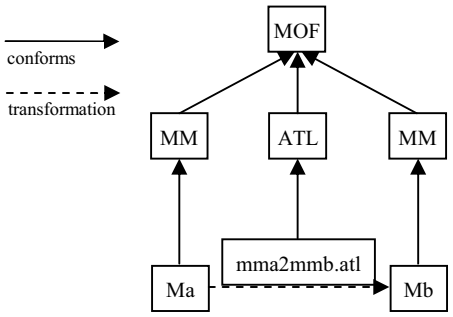

Figure 1. *Cadre opérationnel d'ATL*

3. Méta-modèles UML et Web MVC 2

Conformément au schéma général de la figure 1, on présente dans cette section les différentes méta-classes constituant les méta-modèles PIM et PSM. La figure 2 illustre le méta-modèle source qui est une représentation simplifiée d'un diagramme de classes UML. *UMLPackage* correspond au concept de package UML, cette méta-classe est reliée à la méta-classe *Classifier*. Cette dernière représente aussi bien le concept de classe UML ainsi que le concept de type de données. La méta-classe *Property* exprime la notion de propriétés d'une classe UML ou des références vers les autres classes (Associations uni et bidirectionnelles). La figure 3 correspond au méta-modèle PSM. Dans ce méta-modèle, un grand intérêt est porté à la partie contrôleur. La méta-classe *ActionMapping* contient les informations de déploiement d'une classe *Action* particulière. La méta-classe *ActionForm* est un Bean encapsulant les paramètres d'un formulaire en provenance de la partie vue. La méthode *execute( )* de la classe *Action* effectue son traitement puis appelle la méthode *findforward( )* sur l'objet mapping. La valeur de retour est un objet de type *ActionForward*. La méta-classe *Action* représente le concept de contrôleur secondaire. Les classes *Action* contiennent le traitement propre de l'application. Par voie de conséquence, elles doivent être liées aux classes métier. Le méta-modèle cible complet est détaillée dans [10].

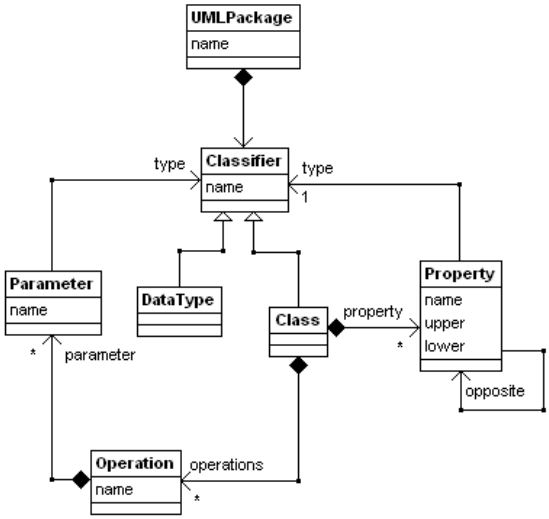

Figure 2. *Méta-modèle source UML*

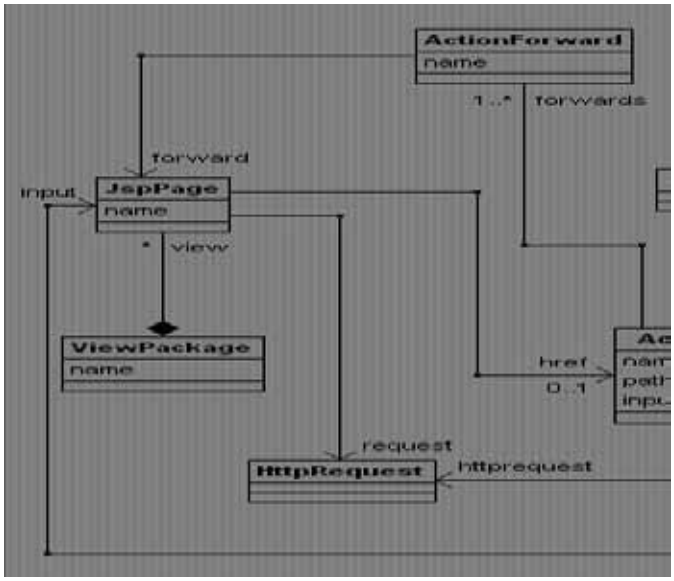

Figure 3. *Méta-modèle cible Struts*

4. Les règles de transformation

Dans cette section, nous allons présenter les règles de transformation permettant le passage d'un diagramme de classes UML vers un modèle Web MVC 2. La règle *UML2ControllerPackageandViewPackage* génère les éléments de ViewPackage à partir du package UML. Le nom du package ViewPackage est le nom de UmlPackage. De même pour le ControllerPackge. L'actionmapping est composée d'un ensemble d'actions qui sont générées à partir d'une opération. Formbean est composée d'un ensemble d'actionForm qui sont générées à leur tour à partir d'une opération. Dans la règle *Operation2ActionetActionForm*, chaque Opération donne lieu à une Action. Seules les opérations CRUD sont considérées Create, Retrieve, Update et Delete, génèrent chacune un élément de type ActionForm. L'operation Retrieve relative à la classe racine ne génère pas d'ActionForm.

```
rule UML2COntrollerPackageAndViewPackage{
    from   a : UML!UML
    to
      vout : STRUTS!ViewPackage(name <- a.name,
             view <- Sequence{thisModule.allMethodDefs
            ->collect (e |thisModule.resolveTemp(e,'jsp'))}),
           out  : STRUTS!ControllerPackage(name <- a.name,
            actionmappings <- Sequence{act},
            formbeans <- Sequence{frmb}),
```

```
    act  : STRUTS!ActionMapping(name <- 'action'+'-'+'mappings',
         actions <-Sequence{thisModule.allMethodDefs->collect(e |
                           thisModule.resolveTemp(e, 'frm'))}),
    frmb : STRUTS!FormBean(name <- 'form'+'-'+'beans',
         form <- Sequence{thisModule.allMethodDefs
          ->collect(e| thisModule.resolveTemp(e, 'actf1'))
                ,thisModule.allMethodDefs
          ->collect(e | thisModule.resolveTemp(e, 'actf'))})}

rule Operation2ActionAndActionForm{
   from c   : UML!Operation
   to
         jsp : STRUTS!JspPage (name <- if c.name='Delete'
                           then OclUndefined
               else  c.name+c.class.name+'.jsp'
               endif),
    frm : STRUTS!Action(path <- '/'+c.name+c.class.name+'Action',
       name<- if c.class.opposite.type.name='Void'
          then OclUndefined
               else  c.name+c.class.name+'Form'
               endif,
           input <- if c.class.opposite.type.name='Void'
          then OclUndefined
       else if c.class.opposite.type.name<>c.class.name
           then '/'+c.name+c.class.opposite.type.name+'.jsp'
              else  OclUndefined
              endif  endif, forwards <- Sequence{fr}),
        fr : STRUTS!ActionForward(name <- 'Success',
       forward <- jsp),
        actf : STRUTS!ActionForm (name <- if c.class.opposite.type.name='Void'
           then if c.name='retrieve'
                then OclUndefined
             else c.name+c.class.name+'Form'
                endif
                      else c.name+c.class.name+'Form'
            endif ),
         actf1 : STRUTS!ActionForm (name <- if c.name='create'
                then
        c.name+c.class.name+'End'+'Form'
                    else if c.name='update'
          then c.name+c.class.name+'End'+'Form'
                   else OclUndefined  endif  endif)}
```

5. Implémentation et exécution de la transformation

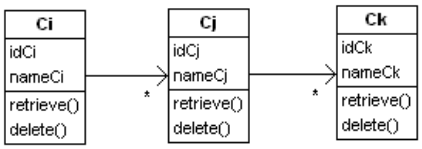

Figure 4. *Modèle source UML*

Nous avons d'abord créé les modèles ECORE correspondants à nos deux méta-modèles source et cible, après on a implémenté les règles de transformation en langage ATL. Pour valider nos règles de transformation, nous avons mené plusieurs tests. A titre d'illustration, nous considérons le diagramme UML composé par les classes C_i, C_j et C_k.

Nous obtenons le modèle XMI ci-après. Afin d'alléger le modèle cible, nous n'avons inclus que les opérations Retrieve et Delete. Pour l'opération "Retrieve", la classe racine C_i n'a pas d'actionform. Pour l'opération "Delete", L'attribut input de la balise actions de DeleteCiAction est /RetrieveCi.jsp car on fait la suppression d'une ligne à partir de la liste de ses propriétés.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xmi:XMI xmi:version="2.0"
    xmlns:xmi="http://www.omg.org/XMI" xmlns="STRUTS">
  <ViewPackage name="ProjetStruts">
    <view name="RetrieveCj.jsp"/>
    <view name="RetrieveCk.jsp"/>
    <view name="RetrieveCi.jsp"/>
  </ViewPackage>
  <ControllerPackage name="ProjetStruts">
    <actionmappings name="action-mappings">
      <actions path="/DeleteCjAction" name="DeleteCjForm" input="/RetrieveCj.jsp">
        <forwards name="Success" forward="/0/@view.0"/>
      </actions>
      <actions path="/DeleteCkAction" name="DeleteCkForm" input="/RetrieveCk.jsp">
        <forwards name="Success" forward="/0/@view.1"/>
      </actions>
      <actions path="/DeleteCiAction" name="DeleteCiForm" input="/RetrieveCi.jsp">
        <forwards name="Success" forward="/0/@view.2"/>
      </actions>
      <actions path="/RetrieveCjAction" name="RetrieveCjForm"
input="/RetrieveCi.jsp">
        <forwards name="Success" forward="/0/@view.0"/>
      </actions>
      <actions path="/RetrieveCkAction" name="RetrieveCkForm"
input="/RetrieveCj.jsp">
        <forwards name="Success" forward="/0/@view.1"/>
      </actions>
      <actions path="/RetrieveCiAction">
        <forwards name="Success" forward="/0/@view.2"/>
      </actions>
    </actionmappings>
    <formbeans name="form-beans">
      <form name="DeleteCjForm"/>
      <form name="DeleteCkForm"/>
      <form name="DeleteCiForm"/>
      <form name="RetrieveCjForm"/>
      <form name="RetrieveCkForm"/>
    </formbeans>
  </ControllerPackage>
</xmi:XMI>
```

6. Conclusion

Nous avons appliqué l'approche MDA à l'ingénierie des applications web. Il s'agit particulièrement de générer les ingrédients d'une application web en se basant sur un diagramme de classes UML. Ce dernier est construit sur la base de différents attributs du système d'information. Le processus de génération donnera la possibilité à l'utilisateur d'ajouter, de modifier, de supprimer et surtout d'afficher des différents objets dont il a besoin. Il doit pouvoir afficher les objets d'une classe donnée en se basant sur les informations d'un autre objet d'une autre classe pour peu que les deux classes soient

liées via des associations au niveau du diagramme de classes. Pour y arriver, nous avons en premier lieu développé le méta-modèle source régissant les diagrammes de classes UML. Au niveau du méta-modèle cible, nous avons conçu toutes les méta-classes nécessaires pour pouvoir générer un modèle PSM respectant une architecture MVC 2. Des règles de transformations ont été élaborées en langage ATL. Le processus de transformation permet de parcourir le diagramme de classe source et de générer, via ces règles, un fichier XMI contenant toutes les actions, les formulaires, les forwards et les pages JSP qui peuvent être utilisés pour générer le code nécessaire de l'application cible. Comme perspective, nous envisageons d'étendre le travail publié dans [11], en proposant une automatisation de notre algorithme de transformation PIM vers PIM.

Bibliographie

[1] J. Bézivin, G. Dupé, F. Jouault, "First Experiments with the ATL Model Transformation Language: Transforming XSLT into Xquery", *2nd OOPSLA Workhop on Generative Techniques in the context of Model Driven Architecture*, October 2003.

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

[3] J. Dubois, J-P. Retaillé, T. Templier, *Spring par la pratique*, Eyrolles, 2006.

[4] J. Goodwill, *Mastering Jakarta Struts*, Wiley, 2002.

[5] F. Jouault, Contribution à l'étude des langages de transformation de modèles, thèse de doctorat, Université de Nantes, 2006.

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

[7] The Model View Controller Framework for PHP Web Applications (http://www.phpmvc.net/index.php).

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

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

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

[11] S. Mbarki, M. Erramdani, "Model-Driven Transformations: From Analysis to MVC 2 Web Model", International Review on Computers and Software (I.RE.CO.S.), Vol. 4, n. 5, September 2009. ISSN: 1828-6003.

[12] S.J. Mellor, K. Scott, A. Uhl, D. Weise, *MDA Distilled, Principles of Model-Driven Architecture*, Addison-Wesley, Pearson Education, 2004.

[13] OMG/MOF meta object facility (MOF) specification, OMG Document AD/97-08-14, September 1997 (www.omg.org).