Vers une architecture MVSOA pour la mise en œuvre des composants multivue

Bouchra El Asri, Adil Kenzi, Mahmoud Nassar, Abdelaziz Kriouile

Laboratoire SI2M, ENSIAS, Rabat, Maroc
elasri@ensias.ma, adil.kenzi@gmail.com, {nassar, krouile}@ensias.ma

Résumé. L'objectif de cet article est de proposer un processus de mise œuvre des systèmes à base de composants multivue selon une architecture orientée services multivue (MVSOA : MultiView SOA). Cette architecture repose sur une extension du standard WSDL appelée MVWSDL (MultiView Web Service Description Language) et sur un ensemble d'adaptateurs pour la publication et la sélection des services multivue. Dans cet article, nous présentons en premier lieu un modèle métier à base de composants multivue. Ensuite, et via des règles de transformations, ce dernier sera transformé vers une plateforme *services web* selon l'architecture MVSOA. Pour ce faire, nous avons défini deux transformations : La première consiste en la définition des règles de transformation pour générer la description d'un composant multivue selon le format MVWSDL. La deuxième transformation cible la plateforme J2EE en se basant sur le méta-modèle J2EE pour générer l'implémentation du composant correspondant.
Mots clés : Architecture orientée service multivue, composant multivue, MDA/MDE, Web Service, MVWSDL.

1 Introduction

Dans l'objectif de répondre à des problèmes récurrents en informatique tels que la réutilisation, la maintenance, l'évolution et l'intégration des systèmes, l'ingénierie logicielle n'a cessé de produire de nouvelles approches et de nouveaux paradigmes. Ainsi, l'ingénierie logicielle à base de composants se positionne récemment comme une approche prometteuse pour faire face à ces types de problèmes. Dans ce contexte, plusieurs approches à base de composants ont vu le jour. En effet, des approches CUP (Component Unified Process) (Renaux 2004), Catalysis (D'Souzaet al. 1999), Fractal (Bruneton et al. 2003) et d'autres pour le développement des systèmes à base de composants ont été proposées. Ces différentes approches visent une meilleure réutilisation des modèles à base de composants. En revanche la complexité inhérente aux besoins croissants des utilisateurs et à la maintenabilité des systèmes reste un champ d'investigation pour les chercheurs. En effet, les composants identifiés lors du cycle de vie sont, en général, des entités qui regroupent les différents rôles joués par ces composants dans les différentes collaborations. Aucune trace des besoins fonctionnels identifiés à la phase d'analyse n'est gardée le long du cycle de développement de logiciel.

Par conséquent, des approches permettant la séparation des préoccupations fonctionnelles ont été proposées dans le but de capitaliser les besoins fonctionnels dans des entités modulaires. Dans cette perspective, nous proposons une approche à base de composant

multivue pour le développement des systèmes à base de composant. Dans cette approche, nous définissons le concept du composant multivue comme une extension du composant UML ayant la spécificité d'avoir des interfaces appelées « interfaces multivue » dont l'accessibilité et le comportement changent dynamiquement selon le point de vue de l'utilisateur courant. Pour un déploiement à échelle des systèmes à base de composants multivue, nous avons proposé une architecture orientée service multivue (MVSOA : MultiView SOA). Cette architecture repose sur une extension du standard WSDL appelée MVWSDL (Multiview Web Service Description Language) et sur un ensemble d'adaptateurs pour la publication et la sélection de services multivue. Dans cet article, nous proposons de décrire le modèle de composant multivue, l'architecture MVSOA, le langage MVWSDL et le processus de mise en œuvre des systèmes à base de composant multivue selon l'architecture MVSOA. La plate-frome choisie pour une telle mise en œuvre est la technologie des services web.

Etant donné que le processus de mise en œuvre des systèmes à base de composant multivue est guidé par l'approche MDA (OMG, 2003), nous élaborons en premier lieu le modèle métier à base de composants multivue (le PIM selon le vocabulaire MDA). Ensuite, et via des règles de transformations, le modèle métier sera transformé vers la plateforme *services web* via deux transformations complémentaires. La première transformation consiste à définir les règles de transformation pour générer la description de composant multivue selon le format MVWSDL. La deuxième transformation cible la plateforme J2EE en se basant sur le méta-modèle J2EE pour générer l'implémentation du service correspondant.

Cet article est structuré comme suit. Nous présentons tout d'abord le modèle de composants multivue de VUML dans la section 2 en le définissant et en présentant son méta-modèle. La section 3 est dédiée à la présentation de l'architecture MVSOA et du langage MVWSDL. La section 4 décrit notre processus de mise en œuvre des systèmes à base de composants multivue selon l'architecture MVSOA. Situé dans le contexte de l'approche MDA, nous définissons, dans cette section, les différentes correspondances entre le méta-modèle source (méta-modèle du composant multivue) et les méta-modèles cibles (MVWSDL, J2EE), l'implémentation en ATL (Jouault et al., 2005) des transformations correspondantes et quelques exemples de code généré. La section 5 présente des travaux relatifs. Dans la conclusion, nous faisons le point sur le travail effectué, et nous présentons nos principales perspectives de recherche.

2 Le modèle de composants multivue dans VUML

2.1 Principes de VUML

VUML (View based Unified Modeling Language) (Nassar 2005) est fondée sur une approche centrée utilisateur, dont l'objectif est de représenter les besoins et les droits d'accès des utilisateurs de l'analyse jusqu'à l'implémentation. VUML se fonde principalement sur les concepts que nous définissons informellement comme suit : Un **acteur** est une entité logique ou physique qui interagit avec le système. Un **point de vue** est la « vision » d'un acteur sur le système (ou sur une partie de ce système). Une **vue** est un élément de modélisation (statique) ; elle correspond à l'application d'un point de vue sur une entité donnée. Par simplification de langage, nous dirons dans la suite qu'une vue est associée à un acteur, en considérant comme implicite l'entité sur laquelle le point de vue s'applique. Le

concept de **classe multivue** est l'élément clé de l'approche VUML. Par rapport à une classe "habituelle", une classe multivue est dotée d'une ou plusieurs vues représentant les besoins et les droits spécifiques des acteurs. Pour prendre en compte le déploiement à échelle, l'évolution d'une architecture et la réutilisation d'éléments d'une application multivue, nous avons introduit la notion de **composant multivue**.

2.2 Le modèle de composants multivue de VUML

Un composant multivue est un module autonome réutilisable et adaptable aux différents acteurs d'un système. Le méta-modèle définissant le concept de composant multivue est présenté dans la figure 1. En plus des caractéristiques d'un composant UML 2.0 standard, un composant multivue est doté d'un ensemble *d'interfaces multivue* fournies et/ou requises qui décrivent le comportement du composant et son interaction avec l'environnement extérieur (El Asri et al. 2005).

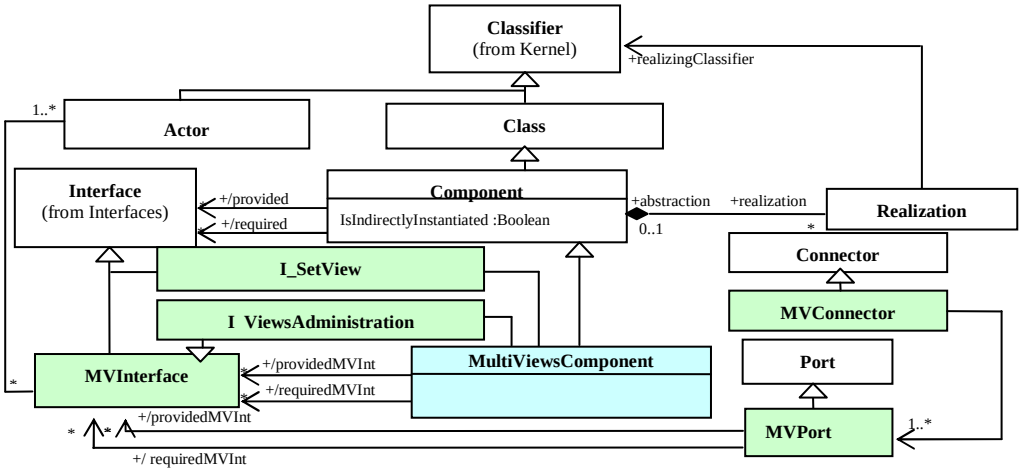

Figure 1 : le méta-modèle du composant multivue

Une *interface multivue* (*MVInterface*) est une interface qui définit, elle aussi, une relation de dépendance "utiliser/fournir" entre le composant et son environnement, mais le comportement des services offerts via cette interface change dynamiquement selon le point de vue actif. Un composant externe doit d'abord activer la vue adéquate avant de pouvoir interagir avec le composant multivue. Cette activation se fait à travers l'interface fournie (*I_SetView*) donnant accès à l'opération *setView().* Chaque composant multivue est doté d'une interface multivue fournie (*I_ViewsAdministration*) qui permet la gestion des vues (ajout/suppression, verrouillage/déverrouillage). Les interfaces multivue peuvent être organisées sous forme de *ports multivue* (*MVPort).* La communication entre le composant et son environnement via un tel port est filtrée et routée vers les constituants internes (*parts*) correspondants selon la vue active. L'interaction via un port mutlivue est réalisée moyennant des *connecteurs multivue*. Ces derniers sont des extensions du connecteur UML classique. Ils permettent les assemblages de composants selon un point de vue déterminé, ce qui permet d'avoir une interaction conditionnée par la vue active.

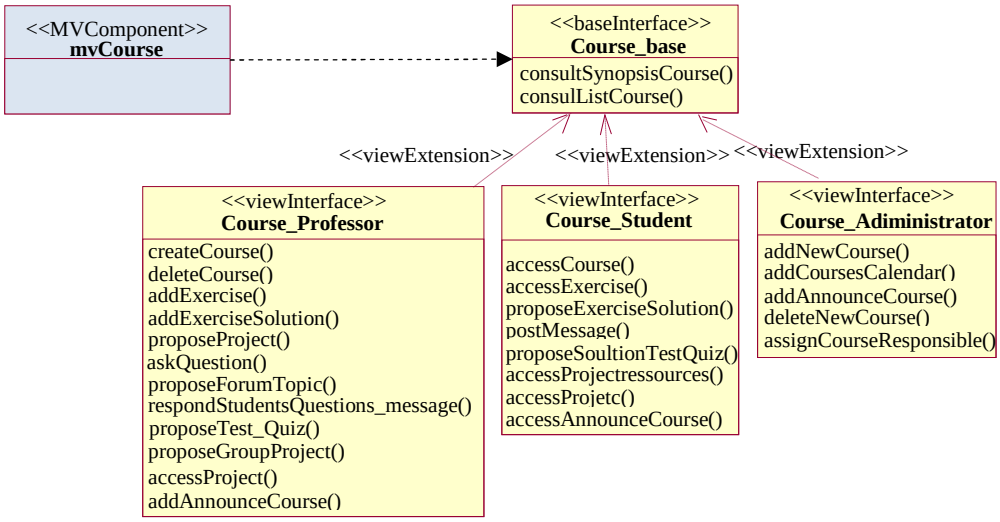

Figure 2 : Le composant multivue mvCourse

La figure 2 ci-dessus montre un exemple de composants multivue *mvCourse* interagissant via des interfaces multivue pour un système d'enseignement à distance. Le composant multivue *mvCourse* offre une interface multivue *Course* pour pouvoir traiter les différents cours demandés par les différents étudiants. Cette interface propose 2 méthodes : *consultSynopsisCourse() et consulListCourse()* qui sont accessibles par tous les types d'acteurs. Les différentes méthodes déclarées dans les "viewInterface" sont des méthodes spécifiques dont l'accès dépend de la vue active. A titre d'exemple, la méthode *createCourse()* ne peut être exécutée que par un enseignant alors que la méthode *addNewCourse*() ne peut être exécutée que par un administrateur. En revanche, le comportement de la méthode *addAnnounceCourse()* change dynamiquement selon la vue active du moment où elle est déclarée dans les viewInterfaces correspondantes à l'enseignant et à l'administrateur.

3 MVSOA : une architecture pour la mise en oeuvre des composants multivue

L'architecture orientée service se base sur une architecture standard définissant trois acteurs et trois opérations standards. Plusieurs extensions ont été proposées pour la prise en compte de différentes préoccupations que l'architecture standard ne prend pas en compte telles UCSOA (User Centric SOA) (Chang, 2006). UCSOA est proposée pour la prise en compte de l'utilisateur final. Dans notre approche, nous proposons une extension à cette architecture appelée MVSOA (MultiView SOA) pour la prise en compte du profil de l'utilisateur final. Cette architecture est implémentée en utilisant les standards définis avec la technologie des services web et les standards inhérents (SOAP, WSDL, UDDI) tout en

établissant une extension légère du standard WSDL appelée MVWSDL (MultiView Web Service Description Language). Cette extension permet la description des interfaces d'un composant multivue tout en prenant en compte le profil de l'utilisateur interagissant avec ce composant multivue. Dans cette section, nous décrivons l'architecture MVSOA et le langage MVWSDL.

3.1 L'architecture MVSOA

L'architecture orientée service (SOA- Service Oriented Architecture) est un modèle d'architecture qui permet la réorganisation des applications isolées en un ensemble des services accessibles à travers des messages et des interfaces standards. SOA définit trois acteurs (le fournisseur de services, l'annuaire de services, le client de service) et trois opérations standards (*find*, *bind*, *publish*). Le fournisseur de services qui possède l'implémentation de services procède à la publication de sa description dans un annuaire de services. Ce dernier contient l'ensemble des descriptions de services publiés par les différents fournisseurs de services. Il fournit les moyens nécessaires pour la publication et la découverte des services. Le client de service pour un besoin donné, recherche dans l'annuaire le service qui le convient.

Pour réaliser une architecture SOA, plusieurs technologies ont été proposées, la plus importante est la technologie des services web qui définit un ensemble de protocoles et de standards se basant sur la norme XML. Ces standards et protocoles fournissent une infrastructure pour décrire (WSDL) (W3C 2006), découvrir (UDDI) (Clement et al. 2004), invoquer (SOAP) (Mitra et al. 2003), et composer (BPEL4WS) des services.

L'architecture SOA ne prend pas en compte le profil de l'utilisateur interagissant avec le service. Pour répondre à cette limite, nous avons étendu l'architecture orientée service en élaborant une architecture appelée MVSOA (MultiView SOA) (cf. figure 3) par l'ajout d'un module de correspondance. Un tel module joue un rôle d'intermédiaire entre le client de service et l'annuaire et entre le fournisseur du service et l'annuaire de service. En relation avec le fournisseur de service ***(l'opération Publish)***, le module de correspondance récupère la description du composant service multivue qui décrit en plus des interfaces simples, les interfaces multivue, stocke les informations concernant les profils d'acteurs et leurs interfaces correspondant, ensuite il normalise la description du service et la publie dans l'annuaire de service. En relation avec le client de service, le module de correspondance récupère la requête du client (***l'opération Find (Service, Profil actor)***) qui contient en plus des besoins du client de service en terme de services, les informations concernant le profil du client de service. Ensuite, il cherche dans l'annuaire de service les services correspondant aux besoins du client de service et il les adapte au profil du client de service avant de les envoyer au client de service.

La figure 3, ci-dessous, présente l'architecture MVSOA :

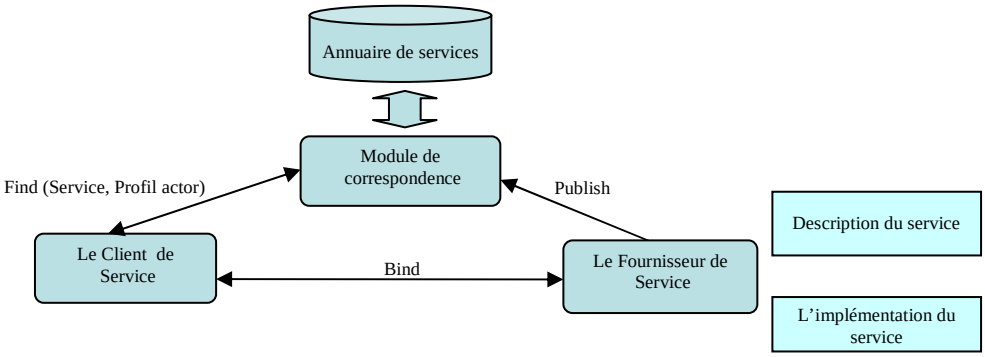

Figure 3 : l'architecture MVSOA

3.2 MVWSDL : une extension de WSDL pour la description des composants services multivue

Dans notre approche, nous avons défini un composant multivue comme une extension d'un composant fournissant, en plus des interfaces simples, des interfaces multivue permettant la prise en compte du profil de l'acteur en interaction avec le service. Pour la description de tels composant services, nous nous basons sur une légère extension du protocole WSDL. Ce protocole qui définit un schéma XML standard pour la description des services. Le schéma XML se base sur six éléments principaux (types, messages, portType, Binding, Port, Service) permettant de définir les interfaces de services, leurs opérations, les paramètres d'entrée/sortie de chaque opération, le type de ces paramètres ainsi que les points d'entrée (URL) de ces opérations. Cependant, ce standard ne permet pas la prise en compte du profil de l'acteur interagissant avec le service. Ainsi, nous définissons une extension du protocole WSDL pour la description d'un composant service multivue. Une telle extension est appelée MVWSDL (MultiView WSDL). Cette extension permet de faire adapter chaque élément du WSDL à l'acteur interagissant avec le service. L'objectif de cette extension est d'une part, de décrire dans un seul document XML l'ensemble des interfaces fournies du composant service qu'elles soient simples ou multivue. D'autre part, elle permet d'adapter cette description à l'exécution suivant le profil de l'utilisateur interagissant avec le service en fournissant à chaque utilisateur la description WSDL standard correspondant à son profil.

Dans l'objectif de génération de code et d'inscription dans les principes et les standards de l'approche MDA, nous établissons le méta-modèle de MVWSDL comme une extension du méta-modèle WSDL via l'ajout de la méta-classe « actor ». Une telle méta-classe permet de définir le type d'acteur interagissant avec le service, et elle s'associe aux différentes méta-classes du méta-modèle WSDL (voir figure 4) qui doivent s'adapter aux profils des utilisateurs. Par exemple, l'élément « PortType » du méta-modèle WSDL qui permet la description d'une interface du service en spécifiant ses opérations fournies, doit être associé à un type d'acteur. Aussi, l'élément «message » qui décrit le format des messages reçus ou envoyés par le service doit être associé à un acteur donné car un message contient des données qu'une opération d'un service peut avoir en entrée ou en sortie. Ainsi, il est nécessaire d'associer à chaque message un acteur donné. Le même principe s'applique au méta-classe « Binding ».

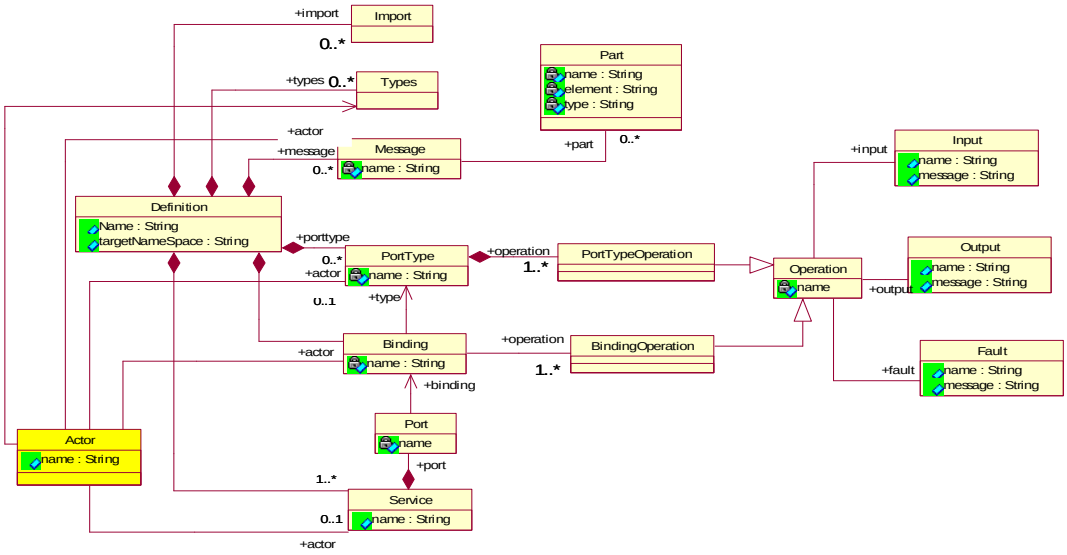

Figure 4 : le méta-modèle du MVWSDL

4 Mise en œuvre des composants multivue sous l'architecture MVSOA

Pour le développement des composants multivue, nous nous sommes basés sur l'approche MDA. Ainsi, nous procédons en premier lieu à l'élaboration d'un modèle métier PIM en se basant les composants multivue. Ce PIM reflète la structure et les fonctionnalités des systèmes indépendamment des plateformes technologiques. Pour sa mise en œuvre, nous avons choisi la plateforme des services web comme modèle spécifique aux plateformes. Pour ce faire, nous avons défini deux transformations. La première consiste à transformer le modèle source à base de composant multivue en un modèle cible conformément au méta-modèle MVWSDL pour ensuite générer un fichier XML permettant la description des interfaces du composant multivue. La deuxième transformation consiste en la transformation de notre modèle source vers le modèle J2EE dans un but de génération du code java qui constitue l'implémentation du composant multivue.

La figure 5 ci dessous illustre globalement notre processus de transformation :

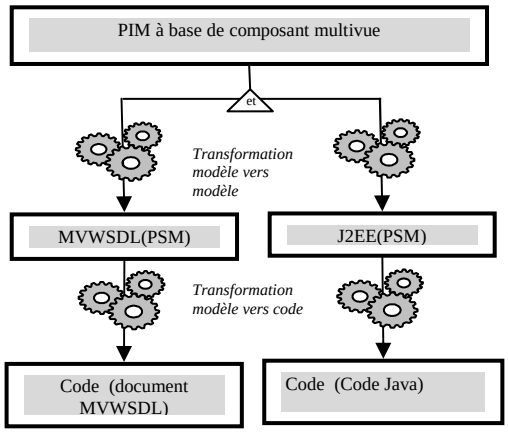

Figure 5: Le processus de transformation de composant multivue vers une plateforme services Web.

4.1 Du PIM à base de composant multivue vers MVWSDL

Comme nous l'avons présenté précédemment, MVWSDL est une extension du standard WSDL pour la description du composant multivue. Pour la transformation des modèles à base de composants multivue vers des modèles MVWSDL dans le cadre de l'approche MDA, nous avons défini en premier lieu les correspondances entre les éléments du méta-modèle source (MVComponent) et les éléments du méta-modèle cible (MVWSDL). En deuxième lieu, nous avons utilisé le langage de transformation de modèle ATL (Atlas Transformation Language) (Jouault et al., 2005) pour implémenter les différentes correspondances identifiées.

La table ci dessous présente les différentes correspondances entre les principaux éléments du méta-modèle du composant multivue et les éléments du méta-modèle MVWSDL.

Elément du méta-modèle composant multivue	Elément du méta-modèle MVWSDL
MVComponent	Definition
<<BaseInterface>>	portType, Binding, Service avec un attribut actor= «allactors »
<<viewInetrface>>	Types, portType, Binding, Service avec un attribut actor = « acteurconcerné> »
Operation	PortypeOperation, BindingOperation, Input, Ouput
Parameter	Part
DataType	Types WSDL

TAB. 1 – *Table de correspondances entre méta-modèle du composant Multivue et MVWSDL*.

Après la spécification des correspondances, vient l'étape de la définition des règles de transformation. Pour définir de telles règles, nous avons utilisé le langage ATL (Atlas

Transformation Language) comme un langage de transformation de modèle. Ces règles permettent l'automatisation de la génération de code correspondant au méta-modèle MVWSDL à partir d'un modèle se conformant au méta-modèle du composant multivue. Plus concrètement, et dans le cadre d'un système d'enseignement à distance, nous identifions les différents composants multivue définissant la structure et les fonctionnalités de ce système tels que le composant multivue *mvCourse* (voir figure 2). Pour chaque composant multivue constituant le système, et en exécutant les règles de transformation spécifiées, nous aurons en sortie un modèle se conformant au méta-modèle MVWSDL. Ce modèle possède assez d'informations pour la génération de code MVWSDL d'un composant multivue tel que le composant mvCourse.

La figure 6 présente une implémentation en ATL de la règle ViewInterface2WSDL. Cette règle permet la création de cinq instances des éléments du méta-modèle MVWSDL : PortType, Types, Binding, Port et Service. Chaque instance créée est initialisée avec les caractéristiques de l'élément viewInterface du méta-modèle du composant multivue.

```
rule ViewInterface2WSDL{
from v: MVComponent!viewInterface
to out : WSDL!PortType(
name <-v.name,
actor <-v.actor.name,
operations<- v.operations_v-> collect ( x |
thisModule.resolveTemp (x,'wsdlop')),
types : WSDL!Types(
actor <- v.actor.name,
xmlschema <- xschema
),
xschema : WSDL!XMLSchema (
namespace<- v.name,
complexType <- v.operations_v ->
collect (
x|thisModule.resolveTemp(x,'complextype_IN'))->
union(v.operations_v -> collect ( x
|thisModule.resolveTemp(x,'complextype_OUT')))),
```

```
bd: WSDL!Binding(
name <-v.name + 'Binding',
actor<-v.actor.name,
porttype <-out,
 boperations<- v.operations_v -> collect ( x |
thisModule.resolveTemp (x,'wsdlob')),
pport : WSDL!Port(
 name <- v.name +'Port',
 binding <- bd
                ),

sv: WSDL!Service (
                        name <-'service' +
v.name,
                        actor <- v.actor.name,
                         ports <- pport
                          )}
```

Figure 6 : Implémentation en ATL de la règle ViewInterface2WSDL

Le code ci-dessous présente un document MVWSDL du service multivue *Course*. Ce service multivue fournit des interfaces multivue qui se composent des interfaces de base et des interfaces vues (*viewinterface)*. Pour illustrer notre approche, nous avons choisi seulement deux opérations des interfaces associées à deux acteurs qui sont l'étudiant et l'enseignant (*Student* et *Professor*) et deux opérations constituant l'interface de base. Cette dernière contient les opérations : *String consultSynopsisCourse (Course : String)* et *String consultCourseList (level :String)*. L'interface associée à l'enseignant contient les opérations suivantes : *String addExercice (exerciseID : integer, Exercise : String)* et *String addExerciseSolution (ExerciseId : integer, ExerciseSolution : String).* L'interface associée à l'étudiant contient les opérations : *String postMessageForum (Course : String, message : String)* et *String ProposeExerciseSolution (ExerciseId : integer, Solution : String).*

Le code ci-dessous illustre un extrait du code généré suivant le méta-modèle de MVWSDL. Dans cet extrait, nous nous focalisons seulement sur l'élément PortType parce que les autres éléments se génèrent de la même façon.

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions name="Course"
targetNamespace="urn://Course.wsdl">
<Types  actor = "allactors">
...
</Types>
<Types actor = "Teacher">
...
</Types>
<Types actor = "Student">
</Types>
<message name="addExerciceRequest"
actor="Teacher">
...
</message>
<portType name="Course_base"
actor="allactors">
<operation
name="consultSynopsisCourse">
  <input
name="consultSynopsisCourseRequest"/>
  <output
name="consultSynopsisCourseResponse"/>
</operation>
```

```
<operation name="consultCourseList">
 <input
name="consultCourseListRequest"/>
 <output
name="consultCourseListResponse"/>
</operation>
</portType>

<portType name="Course_Teacher"
actor="Teacher">

<operation name="addExercice">
   <input name="addExerciceRequest"/>
   <output
name="addExerciceResponse"/>
</operation>

<operation name="addExerciseSolution">
       <input
name="addExerciseSolutionRequest"/>
       <output
name="addExerciseSolutionResponse"/>
 </operation>
 </portType>
...
</definitions>
```

Figure 7: Un extrait du document MVWSDL du service multivue Course

4.2 Du PIM à base de composant multivue vers un PSM (J2EE)

Un service web est un élément logiciel possédant une description du service et une implémentation du service. Pour implémenter un composant multivue en tant que service web, nous avons identifié les correspondances suivantes entre les méta-classes constituant le méta-modèle du composant multivue vers le méta-modèle correspondant à la plateforme technologique J2EE. Le but de ces règles de transformation est la génération du code Java qui constitue la réalisation du composant multivue. La table ci-dessous illustre les correspondances existantes entre le méta-modèle du composant multivue et le méta-modèle JAX-RPC. En effet, JAX-RPC est un ensemble d'APIs du Pack JWSDP pour l'implémentation des services web dans le cadre de la plateforme J2EE.

L'objectif de la spécification de ces correspondances est la définition des règles de transformation pour la génération du code java d'un composant multivue. Pour chaque composant multivue, des règles de transformation s'exécutent pour donner en sortie un modèle se conformant au méta-modèle J2EE. Ce modèle possède assez d'informations pour la génération de code d'un composant multivue.

Pour automatiser la génération de code, nous avons utilisé JAX-RPC comme plateforme d'implémentation. En effet, JAX-RPC est un constituant essentiel du pack JWSDP (Java Web Services Developer Pack) de la plate-forme J2EE dédié pour l'implémentation de services.

Elément du méta-modèle de Composant multivue	Elément du méta-modèle JaxRPC	Règle de transformation
Package	JaxRpcPackage	Package2jaxrpcPackage
MVComponent	JaxrpcClass	MVComponent 2jaxrpcClass
BaseInterface	Interface	binterface2Interface
ViewInterface	Interface	viewinterface2interface
Parameter	JavaParameter	Param2part
Operation	Method	Operation2Method
DataType	Types	Data2Primitive

TAB2 - *Table de Correspondance VUML/J2EE*

La figure 8 présente la règle de transformation *MVComponent2jaxrpcClass* (exprimée en ATL). Elle permet la création d'une instance de classe JAX-RPC du méta-modèle JAX-RPC à partir de l'élément MVComponent du méta-modèle du composant multivue. Pour chaque instance classe générée, on crée des interfaces qui sont initialisées avec les caractéristiques des interfaces du composant multivue (*baseInteface*, *viewInterface)*. Le package et le nom de la classe créé prennent les valeurs des caractéristiques du composant multivue (MVComponent).

```
rule MVComponent2jaxrpcClass {
from d: MVComponent!MVComponent
to class : jaxrpc1!JaxrpcClass(
packagerpc <- d.namespace,
name <-d.name,
interfaces <-
d.provided_mvinterfaces ->collect (x|x.view_interfaces) ->
union(d.provided_mvinterfaces
->collect(x|x.base_interface_M ) ))
}
```

Figure 8: règle de transformation en ATL MVComponent2jaxrpcClass

Les règles de transformations définies permettent la transformation d'un modèle vers d'autres modèles. Elles permettent la génération des modèles dépendants de plateforme (PSM) à partir des modèles métiers de haut niveau (PIM). Pour générer le code, nous avons implémenté de nouvelles transformations en se basant sur les helpers (fonctions définies en ATL) définies dans le contexte de chaque élément du méta-modèle cible. Le rôle de ces helpers est donc la génération de code ciblant une plateforme donnée. Dans notre cas de figure, nous avons généré le code java qui constitue l'implémentation du composant multivue en définissant des helpers dans le contexte de chaque élément du méta-modèle JAX-RPC. Ainsi, nous avons généré pour chaque composant multivue une classe implémentant l'ensemble des interfaces (baseInterface, viewInterface) du composant multivue. Nous avons

généré aussi pour type d'interface et pour chaque opération d'un composant multivue des interfaces Java étendant l'interface Remote et des méthodes qui lèvent l'exception RemoteException.

La figure 9 illustre le code généré de la classe implémentant le composant multivue mvCourse, le code généré qui correspond à l'interface de base, l'interface associé à l'acteur *Student* ainsi que l'interface associée à l'acteur *Teacher*.

```
package DLS;

import java.rmi.Remote;
import java.rmi.RemoteException;

public interface Course_base extends Remote {
public String consultCourseList(String level)throws RemoteException;
public String consultSynopsisCourse(String Course)throws RemoteException;
}

package DLS;

import java.rmi.Remote;
import java.rmi.RemoteException;

public interface Course_Teacher extends Remote {
public String  addExerciseSolution(Integer ExerciseId, String ExerciseSolution)throws RemoteException;
public String  addExercice(Integer exerciseID, String Exercise)throws RemoteException;
}
package DLS;

import java.rmi.Remote;
import java.rmi.RemoteException;

public interface Course_Student extends Remote {
public String postMessageForum(String Course, String message)throws RemoteException;
public String ProposeExerciseSolution(Integer ExerciseId, String Solution) throws RemoteException;
}
package DLS;

public class Course implements Course_Teacher,Course_Student,Course_base,{

public String addExerciseSolution(Integer ExerciseId, String ExerciseSolution) {
        //[Implementation Code to beCompleted]
}
public String addExercice(Integer exerciseID, String Exercise) {
        //[Implementation Code to beCompleted]
}
…
}}
```

Figure 9 : Extrait du code généré de la classe implémentant le composant mvCourse

5 Travaux relatifs

Pour le développement d'applications à base de composants à travers l'architecture SOA, (Stojanovic et al. 2004) proposent une approche pour la modélisation et la conception des solutions SOA. Cette approche se base sur le concept «service component» et sur le langage UML. Deux types de «service component» de différente granularité ont été définis : le BSC

(Business Service Component) et le ASC (Application Service Component) pour la description de systèmes SOA. Dans le même sens, (Zhang et al. 2006) introduisent un framework de modélisation pour l'architecture SOA en se basant sur le concept de «service component». Quant à (Johnston et al. 2006), ils ont définit un profil UML pour SOA se composant d'un ensemble de stéréotypes : service, message, specification, etc.

Pour le développement des applications orientées services en se basant sur l'approche MDA, plusieurs approches ont été élaborées, à savoir (Gronmo et al. 2004), (Bezevin et al., 2004), (Yu et al., 2007) ainsi que (Patrasciou, 2004). Ces approches permettent l'élaboration d'un PIM de haut niveau en se basant sur un langage de modélisation (UML ou EDOC). Ce PIM est transformé vers des plateformes services web (WSDL, J2EE, etc.) via des règles de transformation ou en se basant sur un outil logiciel de transformation de modèles (Gronmo et al., 2004). Cependant, ces approches ne prennent pas en compte des utilisateurs finals au niveau PIM.

Dans une perspective d'adaptabilité de service, plusieurs travaux ont intégré le concept de vue, d'aspect, ou de contexte à celui de composant service. Ainsi, pour l'adaptation des services composites, (Maamar et al. 2005) définissent une vue comme étant une image dynamique de la spécification des services web composites selon un contexte donné. Le concept de vue est utilisé pour faire le suivi de l'exécution des services web composites et pour prendre les mesures correctives dans le cas où l'exécution de services ne se conformant pas aux préférences des utilisateurs. (Chauvel et al., 2008), quant à eux proposent une extension au modèle Fractal pour supporter la notion de vue. Elle consiste à diviser un composant en un composant de base et un ensemble de services visuels. Par contre, (Fink et al. 2003) définissent une approche à base de vue et un langage VPL (View Policy Language) pour la gestion des droits d'accès aux opérations d'un service. Enfin, pour le concept d'aspect, (Kim et al. 2005) proposent une méthode de conception orientée service avec aspects appelée SODA (Service-Oriented Design with Aspects). Cette méthode définit un modèle de services représenté par le langage UML et ses mécanismes d'extensions ainsi que les réseaux de Petri pour la description du comportement du service.

Certes, ces différentes propositions mettent en avant des techniques pour l'adaptation des services en se basant sur différents mécanismes (vues, aspects, etc.) mais elles ne permettent pas la modélisation des besoins des utilisateurs dans un haut niveau d'abstraction. De plus, ces approches ne permettent pas la génération automatique de code. Dans ce sens, (Chang et al., 2006) proposent une extension de l'architecture SOA centrée utilisateur final appelée UCSOA (User Centric Service Oriented Architecture). UCSOA permet non seulement à l'utilisateur final de découvrir les descriptions de services qui correspondent à ses besoins mais elle lui permet d'exprimer ses besoins et ses préférences pour que des fournisseurs de services puissent y répondre.

La différence principale entre UCSOA et MVSOA, bien que les deux architectures sont centrées utilisateur final, est que MVSOA met à la disposition des fournisseurs de services et aux clients de service un framework permettant l'adaptation de chaque service aux profils des utilisateurs interagissant avec le service en fournissant à chaque client de service la description qui correspond à son profil.

6 Conclusion

Dans un objectif de personnalisation, d'adaptabilité et de séparation des préoccupations fonctionnelles, nous avons proposé dans cet article une approche dans le cadre de l'approche MDA pour le développement des systèmes à base de composants multivue sur une plateforme services web. Premièrement, nous avons défini le concept de composant multivue qui permet de décrire pertinemment les besoins fonctionnels d'utilisateurs et leurs droits d'accès. Nous avons ensuite défini son méta-modèle ainsi qu'une extension du méta-modèle WSDL pour la description du composant multivue. Enfin, nous avons défini les correspondances entre les différents éléments du méta-modele source (le méta-modèle du composant multivue) et les méta-modèles cibles (le méta-modèle MVWSDL et le méta-modèle J2EE).

Nos travaux de recherche en cours consistent à finaliser le développement des règles de transformation de modèles pour l'automatisation de la génération de code en utilisant le langage ATL.

Par ailleurs, et pour accompagner l'évolution de l'ingénierie logicielle vers les approches orientées service d'une part, et pour la prise en compte des évolutions considérables dans le domaine de la technologie mobile notamment avec l'amélioration des débits des réseaux et les performances des terminaux mobiles, nous travaillons actuellement à l'élaboration d'une Méthode pour le Développement des Systèmes d'Information Orienté Services Adaptables. Une telle méthode combinant entre trois grandes disciplines en informatiques : l'Informatique Orientée Service, l'Informatique sensible au Contexte et l'ingénierie dirigée par les modèles. Les objectifs de telle méthode est la prise en compte de l'utilisateur final au sens large du terme en prenant en compte, en plus de la gestion de ses droits, son contexte qui inclut entre autre, la localisation, le temps, les performances des dispositifs d'accès ainsi que les préférences de l'utilisateur (sa langue, sa religion, …).

REFERENCES

Bézivin, J., Hammoudi, S., Lopes, D. Jouault F., (2004). *Applying MDA approach for web service platform*. Proc of the 8th IEEE international enterprise distributed object computing conference (EDOC 2004).

Bruneton E., Coupaye T., Stefani .J.-B (2002). *The Fractal commponent model. Specification*, Technical Report v1, v2, The ObjectWeb Consortium. http://fractal.objectweb.org

Chang M.; Jackson He; W.T. Tsai; Bingnan Xiao; Yinong Chen (2006). *UCSOA: User-Centric Service-Oriented Architecture*. e-Business Engineering, 2006. ICEBE apos;06. IEEE International Conference on Volume , Issue , Oct. 2006 Page(s):248 - 255

Chauvel F., Olivier Barais, Noël Plouzeau, Isabelle Borne, Jean-Marc Jézéquel (2008): Expression qualitative de politiques d'adaptation pour Fractal. CAL 2008: 119

Clement L., Hately A., Riegen C., Rogers T. (2004), *UDDI version 3.0.2 spécifications, Technical Commitee Draft*, Octobre, , http://uddi.org/pubs/uddi_v3.htm

D. D'Souza and A. Wills (1999). Objects, *Components, and Frameworks with UML: The Catalysis Approach.* Addison-Wesley Longman,.

El Asri B., Nassar M., Coulette B., Kriouile A. (2005): *Assemblage de composants multivue par contrats*, Actes du XXIIIème Congrès INFORSID (INFORSID'2005), Grenoble, France, 24-27 mai,. p. 29-44.

Fink T, Manuel Koch, Cristian Oancea (2003): *Specification and Enforcement of Access Control in Heterogeneous Distributed Applications*. ICWS-Europe, 88-100.

Fuchs M, (2004) *Adapting Web Services in a Heterogeneous Environment*, Proceedings of the IEEE International Conference on Web Services (ICWS'04), 656-664

Gronmo R., Stogan D., Solheim I., Oldevik J., (2004). *Model Driven web services Development*. Int Journal Web services Res 1(4) : 1-13(2004).

Johnston S , K, Alan W. Brown (2006). *A Model-Driven Development Approach to Creating Service-Oriented Solutions*. ICSOC 624-636

Jouault, F., Kurtev, I., 2005. Transforming Models with ATL. In Proceedings of *the Model Transformations in Practice, Workshop at Models*. Montego Bay, Jamaica 2005.

Kim T., Chang C .K. (2005). *Service-Oriented Design with Aspects (SODA)*, Proceedings of the 2005 IEEE International Conference on Services Computing (SCC'05).

Maamar Z., Benslimane D., Chirine Ghedira (2005). *A View-based Approach for Tracking Composite Web Services*. ECOWS 2005: 170-181

Mitra N., SOAP version 1.2 part 0: primer , W3C Recommendation, juin, 2003.

Nassar M. (2005). *Analyse/conception par points de vue : le profil VUM*. Thèse de l'Institut National Polytechnique de Toulouse, Septembre.

Nassar M., Coulette B., Guiochet J., Ebersold S., El Asri B., Crégut X., Kriouile A. (2005) *Vers un profil UML pour la conception de* composants multivue. Revue RSTI-L'Objet, vol.11 –n°4/2005.

Octavian PATRASCOIU (2004). *Mapping EDOC to Web Services using YATL*. 8th IEEE International Enterprise Distributed Object Computing Conference (EDOC 2004), pages 286–297, September, 2004.

OMG (2003). Object Management Group, MDA Guide, version 1.0.1, June 2003.

Papazoglou M. P. (2003). *Service Oriented Computing : Concepts, characteristics and Directions », actes de la 4° conférence WISE`03, Rome, 10-12décembre*, Italie, IEEE Computer Society, p. 3-12.

Renaux E. (2004). *Définition d'une démarche de conception de systèmes à base de composants*. Thèse de l'Université des sciences et technologies de Lille, décembre 2004.

Stojanovic Z., A. Dahanayake, H. Sol (2004). *Modeling and design of Service Oriented Architecture.* , IEEE International Conference on Systems, Man and Cybernytics.

W3C, 2006. *Web Services Description Language (WSDL) version 2.0 Part 1* : Core language, W3C 2006

Yu X., Zhang Y., Zhang T., Wang L., Hu J., Zhao J., Li X., (2007). *A model-driven development framework for enterprise Web services*. Information Systems Frontiers 9(4): 391-409.

Zhang T., S.Ying, S. Cao, and X.Jia, (2006) *A Modeling Framework for Service-Oriented Architecture*, Proceedings of the Sixth International Conference on Quality Software (QSIC'06).

Summary

The aim of this paper is to propose an approach for the development of Multiview Component Based Systems according to a mutliview service oriented architecture. This architecture relies on a lightweight WSDL extension called MVWSDL and a set of adapters for publishing and selecting multiview services. To situate our approach in the MDA framework, we establish firstly a Platform Independent Model which is based on the Multiview component regardless of technology and standards. Then, we define the transformation rules in order to targeting a web service platform according to the MVSOA architecture. To this end, we have defined two complementary transformations. The first one, consists in the definition of the transformation rules generating the description of the multiview component according to MVWSDL (Multiview Web Service Language Description) format. The second transformation targets a J2EE platform on the basis of the meta-model of the J2EE in order to generate the implementation of the multiview component. Mapping to Platform Specific Model and code generation is done by specifying transformations as a collection of rules implemented in ATL.

Key words : Multiview Component, MVSOA, MDA/MDE, Web Service, MVWSDL.