VERS UN MODELE DE COMPOSANTS LOGICIELS MULTI-VUES

HAIR Abdellatif
Département de Mathématiques et Informatique
Faculté des Sciences et Techniques B.P. 523
23000 BENI-MELLAL MAROC
a.hair@fstbm.ac.ma

1. Introduction

La construction d'applications présente de multiples avantages notamment en terme de réutilisation de code, de qualité de développement, de modularité, etc. L*'ingénierie logicielle basée composant* a pour objectif d'améliorer le développement d'applications par assemblage de *composants logiciel*s [ACCORD 2003][Belloir et al. 2000]. Ces dernières années ont vu émerger différents modèles à composants tels que les EJBs (Enterprise Java Beans) [EJB 2.0] ou CCM (CORBA Component Model) [CCM 3.0]. Ces modèles facilitent la construction d'applications par assemblage de composants logiciels définis par leurs interfaces requis et fournis. Malgré l'engouement autour de ces modèles, la communauté académique propose de nombreux nouveaux modèles pour faciliter la spécification, la construction, le déploiement ou la reconfiguration d'applications à base de composants logiciels .

Par ailleurs, chacun s'accorde à reconnaître l'intérêt de prendre en compte l'utilisateur — au sens large du terme — dans le développement des systèmes complexes. Le concept de point de vue est un moyen pertinent pour mettre en oeuvre cette préoccupation [Carré et al. 1991] [Finkelstein et al. 1990]. Les concepts de vue/point de vue ont été étudiés dans plusieurs domaines de l'informatique, parfois sous d'autres termes tels que sujet [Harrison et al. 1993], séparation multidimensionnelle des préoccupations [Tarr et al. 1999], rôle [Kristensen 1996], aspect [Kiczales et al. 1997], etc.

Dans le cadre d'intégration du concept de vue dans la modélisation orientée objet, le langage de programmation VBOOL [Marcaillou et al. 1995], et la méthode associée VBOOM [Kriouile 1995] ont été définis. Et, plus récemment, nous avons défini la méthode unifiée d'analyse et de conception U_VBOOM fondée sur le concept de vue et basée sur la méthode VBOOM [Hair 2004a][Hair 2004b][Hair 2005]. Et, nous avons proposé encore une implémentation de la relation de visibilité introduite par la méthode VBOOM basée sur le patron d'analyse Rôle [Hair 2004c].

En nous basant sur le concept de vue/point de vue et le composant logiciel, nous allons proposé un modèle de CLM permettant de faciliter le développement d'applications à base de composants logiciels. Ce modèle que nous présentons permet la construction d'un composant logiciel appelé "composant logiciel multi-vues " par assemblage d'un composant logiciel (composant base) et des composants logiciels (composants vues). La spécificité d'un composant logiciel multi-vues est la possibilité d'avoir des interfaces dont l'accessibilité et le comportement changent dynamiquement selon le point de vue de l'utilisateur courant. C'est à dire le client peut activer/désactiver des *composant vues* durant l'exécution. Nous introduisons ainsi la notion de visibilité

dans les modèles de composants logiciels. La réification de cette relation est réalisée par le nouveau type de connecteur que nous proposons, *connecteur de visibilité.*

Ce travail propose donc un modèle de CLM pour les systèmes multi-vues à base de composants. L'assemblage des composants dans ce modèle est réalisée par la relation de visibilité pour pouvoir concevoir des composants dits composants logiciels multi-vues. Nous présentons dans la deuxième section la définition et les propriétés de la relation de visibilité. Dans la troisième section, nous décrivons le modèle de CLM et les concepts *composant base*, *composant vue* et *connecteur de visibilité*. Ensuite, nous présentons dans la section 4, le processus de transformation et la mise en oeuvre des CLM en CCM [CCM 3.0]. La version enrichie du langage IDL3 (Interface Definition Language) [ACCORD 2002] associée à la déclaration des CLM est alors proposée et est appelée IDL3-VIEW. Et, nous finissons par une conclusion.

2. Relation de visibilité : Définition et propriétés

2.1. Définition de la relation de visibilité

La relation de visibilité s'applique quand on veut exprimer le fait qu'une "entité" peut être considérée sous des "angles" multiples [Coulette et al. 1995]. C'est bien le cas d'une Médiathèque, par exemple, qui est appréhendée sous l'angle des prêts, des exemplaires, des comptes adhérents, etc. Dans le langage de programmation VBOOL [Marcaillou et al. 94] l'entité Médiathèque est représentée par une classe multi-vues qui déclare des liens de visibilité avec les classes Prêts, Exemplaires, Comptes_Adhérents appelées ses "vues".

La relation de visibilité est donc une relation qui lie une entité (*entité base*) avec ses vues (*entités vues*) par un ensemble de liens de visibilité. Elle assure la sémantique "est vue comme" qui caractérise l'assemblage d'une *entité base* avec ses *entités vues*. Elle est représentée par un graphe (Figure 1), dénommé **graphe de visibilité**, qui peut dériver à un multi-graphe si au moins un de ses vues est une *entité base* (c'est la propriété de transitivité, qui sera définie par la suite).

Les entités multi-vues sont donc des entités complexes créées par l'assemblage d'une *entité base* et d'un ensemble d'*entités vues* à l'aide de la relation de visibilité. En d'autres termes, une entité multi-vues est formée d'une *entité base* et de plusieurs *entités vues* qui décrivent chacune un type d'utilisation de l'entité multi-vues [Hair 2004c].

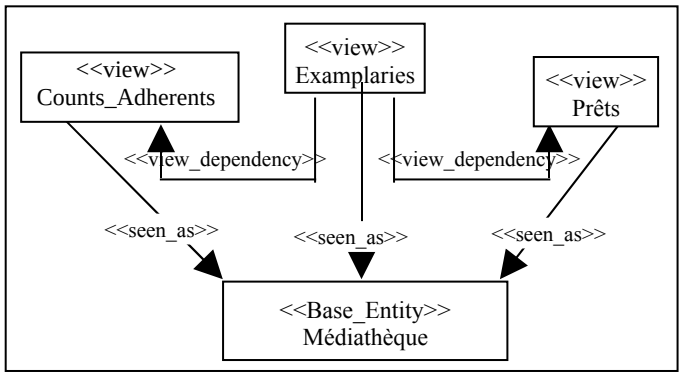

Figure 1. *Graphe de visibilité*

2.2. Propriétés de la relation de visibilité

Parmi les propriétés les plus essentielles de cette relation sont les suivantes :

La transitivité : La relation de visibilité est transitive lorsque les mêmes types de relation de visibilité sont impliqués dans les prémisses.

L'antisymétrique : Une *entité base* est vue comme une *entité vue*, mais l'inverse n'est pas possible. Par exemple : l'*entité base* "Médiathèque" est vue comme une *entité vue* "Prêts" alors que l'entité "Prêts" ne peut pas être vue comme une entité vue "Médiathèque" ;

La dépendance : C'est une propriété importante qui permet la propagation des modifications entre les *entités vues* dépendantes. Le sens de la propagation peut être celui de l'*entité vue* source vers l'*entité vue* destinataire (par exemple : le prêt d'un livre effectué par un adhérent modifie le nombre d'exemplaires disponible de ce livre, ainsi les deux *entités vues* "Prêts" et "Exemplaires" sont dépendantes) ;

L'exclusion mutuelle : Cette propriété consiste à réaliser le droit d'accès sur un modèle multi-vues. L'activation d'une *entité vue* V1 qui présente l'exclusion mutuelle avec une autre *entité vue* active V2 ne peut se produire que si l'*entité vue* V2 est désactivée et inversement.

3. Modèle de composants logiciels multi-vues

Nous consacrons cette partie à la présentation de notre proposition concernant le modèle de composants logiciels multi-vues. Nous commençons premièrement par la présentation des notions de base d'un composant logiciel. Et deuxièmement par l'introduction de la relation de visibilité pour les composants logiciels en proposant de nouveaux concepts tels que le composant base, le composant vue et le connecteur de visibilité. Et nous terminons cette partie par la présentation du modèle de composants logiciels multi-vues.

3.1. Notions de base d'un composant logiciel

Un composant logiciel est une unité de composition spécifiant, par contrats, ses interfaces (fournies et requises) et ses dépendances explicites aux contextes. Un composant logiciel peut être déployé indépendamment et peut être sujet d'assemblage par un tiers pour la conception d'applications logicielles [ACCORD 2003].

L'architecture d'un composant logiciel (Figure 2) spécifie ses entrées et ses sorties afin de faciliter la description de son comportement (services offerts) quels que soient les langages de programmation utilisés.

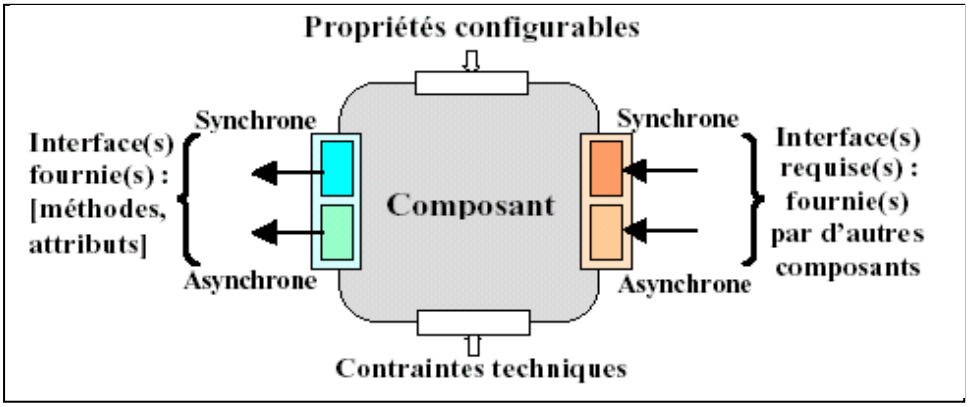

Figure 2. *Architecture d'un composant logiciel*

Comme le présente la figure ci-dessus, un composant logiciel possède, principalement, les trois éléments suivants :

- Les interfaces fournies *(sortie*s) et les interfaces requises *(entrée*s), en mode synchrone ou asynchrone : ce qui définit ses moyens mis en oeuvre pour coopérer. Ces moyens peuvent être des opérations (des fonctions promises aux clients) ou des propriétés ;

- Les propriétés configurables : généralement, ce sont des attributs. Elles permettent d'adapter et de personnaliser le composant dans des contextes d'exécutions spécifiques;

- Les contraintes techniques (QoS : *Quality of Service*s), qui peuvent être : la sécurité, la persistance, les transactions, etc.

3.2. Relation de visibilité pour les composants logiciels

Pour pouvoir introduire la relation de visibilité dans les composants logiciels, nous devons commencer à donner de manière précise la notion de composant logiciel multi-vues ainsi que l'ensemble de propriétés de la relation de visibilité pour les composants logiciels.

Un composant logiciel multi-vues (CLM) est un composant logiciel qui peut être considéré et appréhendé sous des "angles" multiples, dénommés ses ***composants vues****. Un composant logiciel multi-vues a un composant de logiciel appelé* ***composant base*** *lié avec des* ***liens de visibilité*** *avec* ***les composants vues****.*

A partir de cette définition nous pouvons établir les règles à respecter pour pouvoir introduire la notion de composant logiciel multi-vues dans les modèles de composants logiciels muti-vues :

Règle 1 : un CLM est avant tout un composant logiciel qui publie des interfaces fournies et requises.

Règle 2 : un CLM est constitué par un *composant base* et les *composants vues*. Un *composant base* est lié à ses *composants vues* par des liens de visibilité.

Règle 3 : les liens de visibilité doivent refléter les propriétés de la relation de visibilité telles que par exemple la dépendance, l'exclusion mutuelle, etc..

Règle 4 : un *composant vue* ne peut s'appliquer que sur un unique *composant base*.

Règle 5 : les *composants vues* sont accessibles directement tant qu'ils ne font pas partie d'un CLM. A partir du moment où ils deviennent des *composants vues* d'un CLM, ils seront inaccessibles qu'à travers leur CLM.

Règle 6 : un CLM doit offrir la possibilité de changer dynamiquement son comportement et son accessibilité selon le type d'utilisation (besoin d'un client).

3.3. présentation du modèle de composants logiciels multi-vues.

Dans l'approche "composant", une application logicielle est construite par une collection de composants logiciels interconnectés à l'aide de connecteurs [Traverson et al. 2002] [Nikunj et al. 2000]. Dans notre modèle, la liaison de visibilité est traduite en un connecteur appelé *connecteur de visibilité* qui est un composant logiciel. Il possède ses interfaces et ses propriétés et s'interpose entre le *composant base* et le composant vue pour réifier la sémantique de la relation de visibilité (Figure 3). Par conséquent, le connecteur de visibilité doit assurer les propriétés de la relation de visibilité est plus particulièrement les deux propriétés : la dépendance et l'exclusion mutuelle.

La dépendance : Pour assurer la relation de dépendance entre deux *composants vues*, une variable entière sera stockée dans le connecteur de visibilité correspondant. Cette variable doit être testée pour toutes les opérations de modifications concernant les informations liées entre les deux composants vues.

L'exclusion mutuelle : Pour assurer le droit d'accès entre deux *composants vues* en exclusion mutuelle, une variable entière sera stockée dans les connecteurs de visibilité correspondants. Cette variable doit être testée pour toutes opérations d'activation d'un composant vue en exclusion mutuelle.

L'utilisation de CLMnécessite la définition de l'ensemble des opérations permettant d'assurer les services associés à la relation de visibilité. Les opérations de manipulation de CLM sont activer et désactiver un *composant vue* faisant un lien de visibilité avec un *composant base* et l'invocation de l'un des interfaces du CLM. Il est nécessaire de vérifier la cohérence de la sémantique de la relation de visibilité lors de l'utilisation de chaque opération. Nous soulignons que les interfaces fournies par le CLM incluent les interfaces fournies définies par ses *composants vues*. L'utilisation de ces derniers est spécifiée lors de la définition des liens de visibilité.

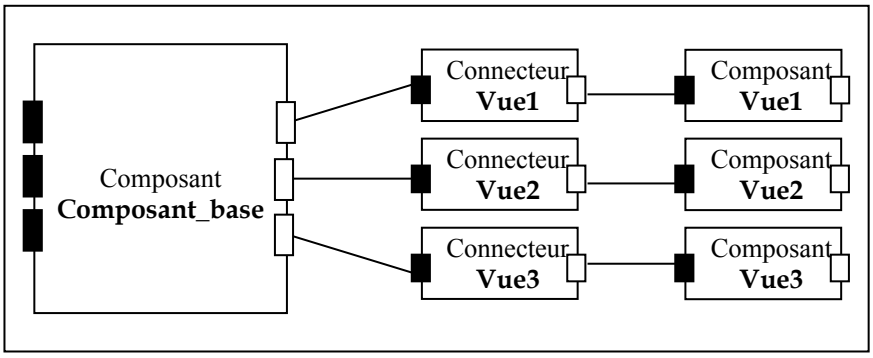

Figure 3. *Modèle de composant du CLM*

4. Processus de transformation et mise en oeuvre des CLM

4.1. Présentation du processus de transformation

Conformément à la démarche MDA (Model driven architecture) [MDA], notre modèle de CLM est un modèle indépendant de toute plate-forme (PIM). A partir de ce modèle, un processus de transformation a été défini et utilisé pour les projections CCM [CCM 3.0] et EJB [Greenfield 01] [Edwards et al. 2002]. La figure 4 illustre le processus de transformation utilisé.

Dans un premier temps, un nouveau modèle est automatiquement généré à partir du modèle indépendant fourni. Ce nouveau modèle correspond à la copie conforme du modèle d'origine à laquelle on ajoute un ensemble de décisions spécifiques à la plate-forme cible qui ne peuvent être spécifiés dans le modèle indépendant.

Une fois les décisions spécifiques à la plate-forme cible sont ajoutées, une autre transformation délivre le modèle UML spécifique à la plate-forme choisie. Ce modèle est un PSM (Platform Specific Model) dans le contexte MDA. Les règles de transformation sont automatiquement appliquées en fonction des décisions spécifiques à la plate-forme cible.

Finalement, le concepteur peut adapter le modèle spécifique (PSM) obtenu, en particulier en définissant ses choix d'implantation. Ces modèles spécifiques à une plate-forme particulière sont exprimés à l'aide de profils UML. Les éléments spécifiques à la plate-forme CCM sont exprimés à l'aide du langage UML [UML 2.0] et du profil CCM (fondé sur le profil CORBA [UML CORBA V3.0.3]). Dans la plate-forme CCM, l'assemblage est exprimé à l'aide de fichiers XML, appelés "descripteurs d'assemblage", ceux-ci sont également générés par l'outil sous la forme de notes UML.

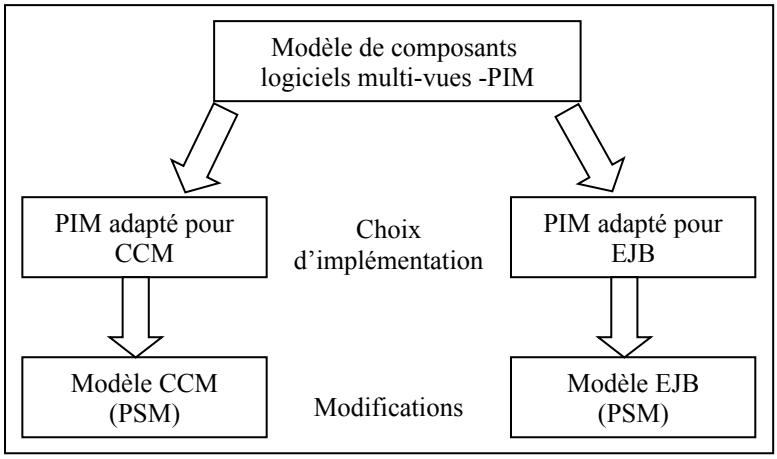

Figure 4. *Processus de transformation*

4.2. Mise en oeuvre des CLM en CCM

4.2.1 Processus de conception de CLM en CCM

Dans la partie précédente, nous allons introduire la relation de visibilité pour les modèles de CLM. Dans cette partie, nous allons mettre en oeuvre ce modèle pour les composants CCM. Pour pouvoir définir et utiliser des CLM dans le modèle CCM, nous allons procédé comme suite :

- La description du CLM ou de son utilisation en VIEW-IDL3 ;

- La projection de cette description ou de cette utilisation vers IDL3 par le compilateur **VIEW-IDL3 ToIDL3** ;

- L'interconnexion entre le composant base, les connecteurs de visibilité et les composants vues.

3.2.2. Description des CLM : le langage VIEW-IDL3

La description d'un type de CLM nécessite l'introduction de nouveaux concepts. Ces derniers sont ajoutés à ceux définis par le langage IDL3 [ACCORD 2002]. La description d'un CLM, à l'aide de VIEW-IDL3 définit principalement les éléments suivants: le module de la classe composant logiciel multi-vues, son *composant base*, ses *composants vues* et ses *connecteurs de visibilité* (liens de visibilité) et leurs propriétés.

L'exemple suivant montre la description d'un CLM Médiathèque, son *composant base* et les *composants vues*, ainsi les propriétés des *connecteurs de visibilité*. La déclaration en VIEW-IDL3 des CLM se fait tout simplement comme celle en IDL3 (Tableau 1).

```
module demoMediatheque {
//déclaration des interfaces
interface LoansInterface {
liste opérations….. ;};
interface Counts_AdherentsInterface {
liste opérations….. ;};
interface ExamplariesInterface {
liste opérations….. ; };
//déclaration du composant composant_base multi-vues Mediathèque
composant_base Mediathèque {
attribute string the_name;
type : asynchrone ;
port_out LoansInterface ;
port_out ExamplariesInterface ;
port_out Counts_AdherentsInterface ;
port_in LoansInterface ;
```

```
port_in ExamplariesInterface ;
port_in Counts_AdherentsInterface  ;
composant_vue Loans_View {
attribute string the_name;
//déclaration des variables de la relation de visibilité
actif : false ;
indice : 1 ;
dependance : 0 ; // composant vue indépendant des autres composants vues
exclusion_mutuelle : 0 ; // composant vue ne présentant aucune exclusion mutuelle
};
composant_vue Examplaries_View {
attribute string the_name;
//déclaration des propriétés de la relation de visibilité
actif : false ;
indice : 2 ;
dependance : 1 ; // indice du composant vue dépendant : Loans_view
exclusion_mutuelle : 3 ;  // indice du composant vue qui présente l'exclusion  mutuelle avec ce composant vue };
composant_vue Counts_Adherents_View {
attribute string the_name;
//déclaration des propriétés de la relation de visibilité
actif : false;
indice :3;
dependance : 1 ; // indice du composant vue dépendant : Loans_view
exclusion_mutuelle : 3 ; // indice du composant vue qui présente l'exclusion  mutuelle avec ce composant vue
};
//La déclaration de la maison du composant composant_base Médiathèque
home MédiathèqueHome manages Médiathèque { };
// La déclaration du composant Loans_View
component Loans_View {
attribute string the_name;
provides LoansInterface Loans_ViewInterface;
};
//déclaration de la maison "home" du composant Loans_View
home Loans_ViewHome manages Loans_View  {  };
// La déclaration du composant Counts_Adherents_View
component Counts_Adherents_View {
attribute string the_name;
provides Counts_AdherentsInterface Counts_Adherents_ViewInterface;
};
//déclaration de la maison "home" du composant Counts_Adherents_View
home Counts_Adherents_View Home manages Counts_Adherents_View { };
// La déclaration du composant Examplaries_View
component Examplaries_View  {
attribute string the_name;
provides ExamplariesInterface Examplaries_ViewInterface ;
};
//déclaration de la maison "home" du composant Examplaries_View
home Examplaries_ViewHome manages Examplaries_View  { };
} ;
```

Tableau 1. *Le fichier de déclaration du* CLM *Médiathèque en VIEW-IDL3*

L'ensemble des définitions que nous avons introduits au langage IDL3 sont résumées dans le tableau ci-dessous (Tableau 2).

CONCEPT	**LA DESCRIPTION DU CONCEPT**
Composant_base	Pour spécifier un type de composant base
Composant_vue	Pour spécifier un type de *composant vue*
Indice	Chaque *composant vue* a son propre indice qui est différent et l'identifie des autres composants vues.
dependance	S'il est différent de zéro alors il existe une dépendance entre deux composants vues. Ce nombre représente l'indice du *composant vue* dépendant avec le *composant vue* courant. S'il est égal à zéro alors il n'y a pas de dépendance entre le *composant vue* courant et un autre composant vue.
exclusion_mutuelle	S'il est différent de zéro alors il existe une exclusion mutuelle entre deux composants vues. Ce nombre représente l'indice du *composant vue* en exclusion mutuelle avec le *composant vue* courant.

	S'il est égal à zéro alors il n'y a pas d'exclusion mutuelle entre le *composant vue* courant et un autre composant vue.
actif	Prend les deux valeurs : True : le *composant vue* est actif False : le *composant vue* est inactif
type	Prend les deux valeurs : synchrone et asynchrone.
port_in	Représente le type de l'interface requise (un réceptacle ou un puits d'événements)
port_out	Représente le type de l'interface fournie (une facette ou source d'événements)

Tableau 2. *Ensemble des définitions introduits au langage IDL3*

4.2.3. Projection de la description d'un CLM de VIEW-IDL3 vers IDL3

La projection de la description d'un CLM, de VIEW-IDL3 en IDL3 par le compilateur **View-IDL3ToIDL3** (Figure 5), suit des règles bien définies. Toutefois la règle principale est qu'un type de CLM est transformé en un ensemble de composants représentant le *composant base*, ses *composants vues* et les *connecteurs de visibilité*.

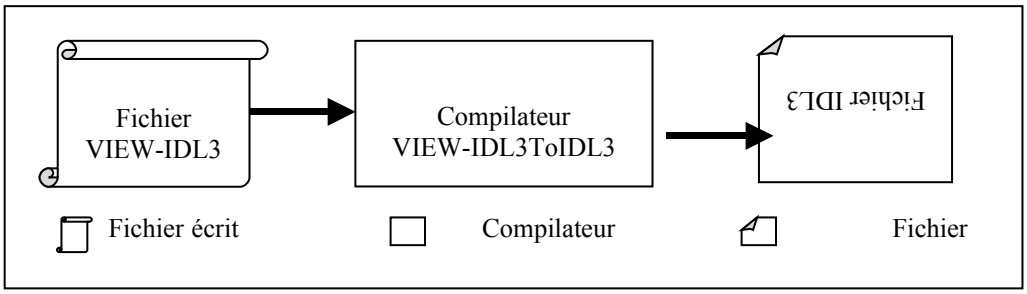

Figure 5. *Processus de projection des fichiers VIEW-IDL3 vers IDL3*

La projection de la description d'un CLM de VIEW-IDL3 vers IDL3 produit :

1. La définition d'un composant qui prend le nom : "nom_composant" avec sa maison *"home"* qui possède le nom "nom_composantHome";

2. Pour chaque description d'un lien, il définit un composant qui dispose du nom du composant suffixé par *Connector.* Le même suffixe sera ajouté à sa maison ;

3. La description des *composant vues* et le *composant base* reste elle-même en gardant aussi, la déclaration de la maison ;

4. En fonction de la valeur de type : si synchrone, il crée soit une facette (si port_in : true) pour le ***composant base*** et un réceptacle pour le **composant vue**. Si asynchrone, il crée une source d'événements pour le *composant base* et un puits d'événement pour le ***composant vue*** ;

5. La déclaration des attributs reste telle qu'elle est ;

6. Les ports d'entrée/sortie gardent le même nombre mais change la façon de déclaration.

Pour illustrer le résultat de la projection de VIEW-IDL3 vers IDL3, nous présentons la projection de l'exemple donné dans la section précédente (exemple d'un CLM Médiathèque) dans le tableau suivant (Tableau 3) :

```
// Le fichier de déclaration du CLM Médiathèque en IDL3
module demoM {
//Définition des interfaces
interface LoansInterface {
liste opérations….. ; };
interface Counts_AdherentsInterface {
liste opérations….. ; };
interface ExamplariesInterface {
liste opérations….. ; };
```

```
//La description du composant composant_base MediathèqueComposant_Base
component MediathèqueComposant_Base {
attribute string the_name;
provides LoansInterface ;
provides ExamplariesInterface ;
provides Counts_AdherentsInterface  ;
uses LoansInterface  To_LoansViews;
uses ExamplariesInterface To_ExamplariesViews ;
uses Counts_AdherentsInterface To_Counts_AdherentsViews  ;
//La déclaration de la maison du composant composant_base MediathèqueComposant_Base
home MediathèqueComposant_BaseHome manages MediathèqueComposant_Base { };
// La déclaration du composant Loans_View
component Loans_View {
attribute string the_name;
provides LoansInterface Loans_ViewInterface;
};
//déclaration de la maison "home" du composant Loans_View
home Loans_ViewHome manages Loans_View { };
// La description du connecteur Loans_ViewConnector
component Loans_ViewConnector {
attribute string the_name;
//La déclaration des variables de la relation du connecteur Loans_ViewConnector
attribute boolean actif;
attribute int indice;
attribute int dependance;
attribute int exclusion_mutuelle;
//La déclaration des interfaces du connecteur Loans_ViewConnector
provides LoansInterface for_MediaLoansCon ;
uses LoansInterface to_LoansCon ;
};
//La déclaration de la maison du connecteur Loans_ViewConnector
home Loans_ViewConnectorHome manages Loans_ViewConnector { };
// La déclaration du composant Examplaries_View
component Examplaries_View {
attribute string the_name;
provides ExamplariesInterface Examplaries_ViewInterface;
};
//déclaration de la maison "home" du composant Examplaries_View
home Examplaries_ViewHome manages Examplaries_View { };
// La description du connecteur Examplaries_ViewConnector
component Examplaries_ViewConnector {
attribute string the_name;
//La déclaration des variables de la relation du connecteur Examplaries_ViewConnector
attribute boolean actif;
attribute int indice;
attribute int dependance;
attribute int exclusion_mutuelle;
//La déclaration des interfaces du connecteur Examplaries_ViewConnector
provides ExamplariesInterface for_MediaExamplariesCon ;
uses ExamplariesInterface to_ExamplariesCon ;
};
//La déclaration de la maison du connecteur Examplaries_ViewConnector
home Examplaries_ViewConnectorHome manages Examplaries_ViewConnector { };
// La déclaration du composant Counts_Adherents_View
component Counts_Adherents_View {
attribute string the_name;
provides Counts_AdherentsInterface Counts_Adherents_ViewInterface;
};
//déclaration de la maison "home" du composant Counts_Adherents_View
home Counts_Adherents_ViewHome manages Counts_Adherents_View  { };
// La description du connecteur Counts_Adherents_ViewConnector
component Counts_Adherents_ViewConnector {
attribute string the_name;
//La déclaration des variables de la relation du connecteur Counts_Adherents_ViewConnector
attribute boolean actif;
attribute int indice;
attribute int dependance;
```

```
attribute int exclusion_mutuelle;
//La déclaration des interfaces du connecteur Counts_Adherents_ViewConnector
provides Counts_AdherentsInterface for_MediaCountsCon ;
uses Counts_AdherentsInterface to_CountsCon;
};
//La déclaration de la maison du connecteur Counts_Adherents_ViewConnector
home Counts_Adherents_ViewConnectorHome manages Counts_Adherents_ViewConnector { };
```

Tabeau 3. *Un extrait de projection de la description de la Médiathèque en IDL3*

4.2.4. L'interconnexion des composants

Après avoir implémenté le code métier du CLM, son *composant base*, ses *composant vues* et ses *connecteurs de visibilité*, nous avons procédé à l'interconnexion de ces composants. Cette opération doit être automatisée dans notre modèle. En effet, un fichier JAVA contenant l'ensemble de connexions entre connecteurs et composants est automatiquement généré. Un outil dédié à cette tâche est en cours implémentation**.**

5. Conclusion

Dans cet article, nous avons présenté un modèle à base de composants logiciels multi-vues. Ce modèle de CLM combine les avantages de l'approche par point de vue et les avantages du développement de logiciels à base de composants.

Le modèle à base de CLM permet la construction d'un composant logiciel par assemblage d'un *composant base* et des composants vues. Le *composant vue* est lié avec le *composant base* se fait par le nouveau type de connecteur appelé *connecteur de visibilité.*

Le modèle proposé est un modèle indépendant de toute plate-forme ce qui conforme à la démarche de développement dirigé par les modèles MDA.

Nous avons également présenté la mise en ouvre de conception d'un composant logiciel multivues en CCM qui a permis de valider le modèle proposé et les étapes à suivre pour concevoir un composant logiciel multi-vues, en commençant de la description jusqu'au lancement d'exécution sur la plate forme OpenCCM. Ainsi, il est à noter que l'interconnexion entre *composant base* avec les composants vues, selon notre modèle, se fait automatiquement. La description du type de CLM a nécessité l'introduction de nouveaux concepts. Ces derniers sont ajoutés à ceux définis par le langage IDL3.

Bibliographie

[ACCORD 2002] Projet ACCORD, *Le modèle de composants CORB*A, Livrable-1.1-4, 31, Mai 2002.

[ACCORD 2003] Projet ACCORD, *Modèle abstrait d'assemblage de composants par contrats*, Livrable-4, Juin 2003.

[Belloir et al. 2001] Belloir N., Bruel J.M., Barbier F., *Formalisation de la relation tout-paire : application à l'assemblage des composants logiciels*, journée composants : flexibilité du système au langage, 2001.

[Carré et al. 1991] Carré B., Geib J.M., The Point of View Notion for Multiple Inheritance, Proc. Of ECOOP/OOPSLA, 1991.

[CCM 3.0] OMG, *CORBA Components* , Version 3.0, Object Management Group, June 2002.

[Coulette et al. 1995] Coulette B., Kriouile A., Marcaillou S., *L'approche par points de vue dans le développement orienté objet des systèmes complexes*, Revue l'Objet, vol. 2, n°4, février 1996, p. 13-20.

[Edwards et al. 2002] W. Edwards, S. Deshpande, *Intégration de CORBA et d'EJB*, Livre Blanc, page 1-14, Janvier 2002.

[EJB 2.0] SUN Enterprise Java Beans Home page, http://java.sun.com/products/ejb/docs.html

[Finkelstein et al. 1990] Finkelstein A., Kramer J., Goedicke M., *Viewpoint Oriented Software Development*, Proc. of S. Eng. and Applications Conference, Toulouse, dec. 1990, p. 337-351.

[Greenfiel 01] Greenfield J., *UML Profile for EJB*, Technical report, Rational Software Corporation, May 2001.

[Hair 05] Hair A., *A UML extension for viewpoint-oriented modeling*, IADIS International Conference IADIS Appied Compting, IADIS'05, Algarve, Portugal, 22-25 February, 2005.

[Hair 04c] Hair A., *Implantation de la relation de visibilité, approche par le patron d'analyse rôle*. Actes de conférence, 7ème colloque africain sur la recherche en informatique, CARI'04, pp 181-189, Hammamet, Tunisie, 22-25 November, 2004.

[Hair 04b] Hair A., *Analysis and design process based on the viewpoint concept*, In RITA - Revista de Informática Teorica e Aplicada -, Vol. X, No. 2, pp 63-75, 2004.

[Hair 04a] Hair A., *U_VBOOM: Unified analysis and design process based on the viewpoint concept*, Actes de conférence, 6th International Conference on Enterprise Information systems (Porto/Portugal) ICEIS'04, 14-17 April, 2004.

[Harrison et al. 1993] Harrison W., Ossher H., Subject-oriented programming : a critique of pure objects, Proc. of OOPSLA'93, Washington D.C., 1993, pp. 411-428.

[Kiczales et al. 1997] Kiczales G., Lampng J., Mendhekar A., Maeda C., Lopes C. V., *Aspect-Oriented Programming*, Proc. of the European Conference on Object-Oriented Programming (ECOOP). Finland. Springer-Verlag LNCS 1241, 1997.

[Kristensen, 1996] Kristensen B. B., *Object-oriented modeling with roles*, *In John Murphy and Brian Stone, editors, Proceedings of the 2nd International Conference on Object-Oriented Information System*s, 1996, Springer-Verlag, p. 57–71.

[Kriouile 1995] Kriouile A., *VBOOM, une méthode d'analyse et de conception par objet fondée sur les points de vue*, Thèse d'état, faculté des sciences de Rabat, 1995.

[Marcaillou et al. 1995] Marcaillou S., Coulette B., Kriouile A., *Visibility : A new relationship for complex system modelling*, TOOLS USA'94, Santa Barbara, August 1-5 1994, Prentice Hall.

[MDA] OMG Model-Driven Architecture Home page, http://www.omg.org/mda/

[Nikunj et al. 2000] Nikunj R.M., Medidovic N., Phadke S., *Towards a Taxonomy of Software Connector*s, In Proceeding of the 22nd International Conference on Software Engineering, Limerick, Ireland, Juin 2000.

[Tarr et al. 1999] Tarr P. L., Ossher H., Harrison W. H., Sutton S. M. Jr., *N Degrees of Separation: Multi-Dimensional Separation of Concerns,* International Conference on Software Engineering, 1999 pages "107-119" Workshop, 1999.

[Traverson et al. 2002] Traverson B., Yahiaoui N., *Connector for CORBA Components,* In 8th International Conference on Object-Oriented Information Systems, September 2002.

[UML 2.0] OMG, *Unified Modeling Language: Superstructure*, Technical Report Version 2.0, Object Management Group, 2003.

[UML CORBA V3.0.3] OMG, *Common Object Request Broker Architecture (CORBA/IIOP),* Document --formal/04-03-01 (CORBA, v3.0.3), Mars, 2004, http:// www.omg.org/cgi-bin/ doc?formal/04-03-01