Conception par aspects fonctionnels

Des objets aux composants

© Olivier Caron

8 novembre 2001

LIFL

Plan

√ Le cadre de conception **objet**

- Contexte d'application, description, résultats
- Conservation des aspects à l'exploitation

√ Vers un cadre de conception **composant**

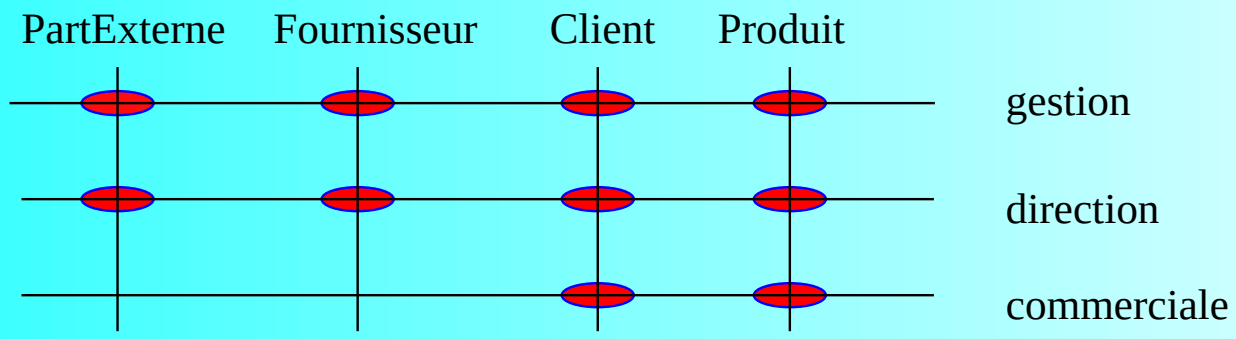
- Vers la notion de composants multi-fonctionnels de conception
- Critiques et limites de l'approche objet
- Les apports : réutilisation, assemblage
- Vers des **Composants fonctionnels** à l'exploitation
- Support UML

Le cadre de conception objet

- √ Cadre de conception, pas une méthodologie (eq. UML)
  Quelques guides (use case)
- √ Contexte : orienté SI (bases de données)
- √ Double finalité :
  - Objectif 1 : concevoir le SI d'une entreprise
    Apport du modèle objet : structuration en fonctions du SI au sein de l'objet (séparation des aspects fonctionnels)
  - Objectif 2 : préserver les objets fonctionnels à l'exploitation (tracabilité, évolution)

Séparation des aspects fonctionnels

√ Orthogonalité objets/fonctions



√ Contextes fonctionnels (contexte d'intervention)

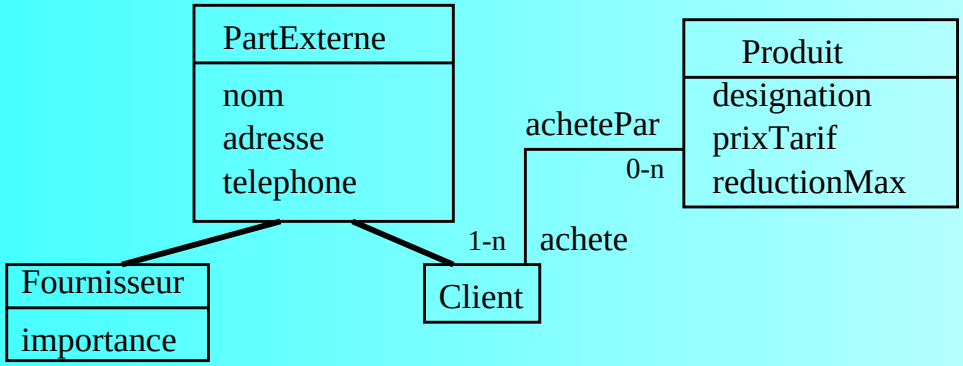
- Conséquences intensionnelles
- Conséquences extensionnelles

Cadre de conception

√ Conception par plans :

- Plan de base : référentiel des entités
- Plan fonctionnel : adaptation du référentiel au contexte
- Contexte fonctionnel : plan de base + plan fonctionnel + extension fonctionnelle

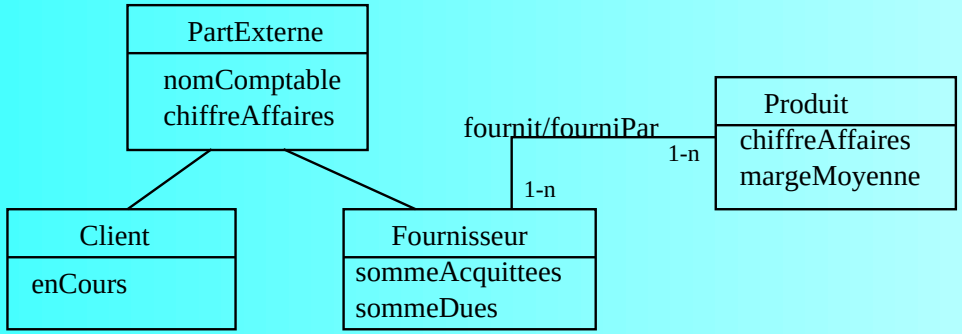
Illustration : plan de base



√ Informations communes : le référentiel du schéma

√ Définition de la structure

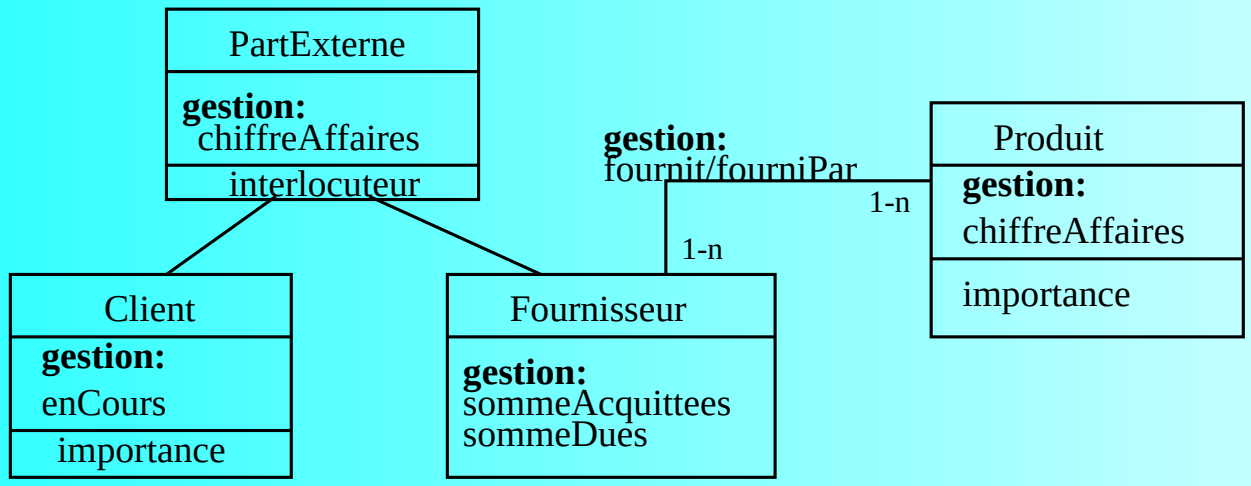
Illustration : plan fonctionnel



Plan fonctionnel de la fonction "gestion"

- √ Préservation structure hiérarchique
- √ Enrichissement implicite par héritage $\Longrightarrow$ intension fonctionnelle
- √ Introduction logique métier : affinement des informations (cardinalités, navigation,. . .)

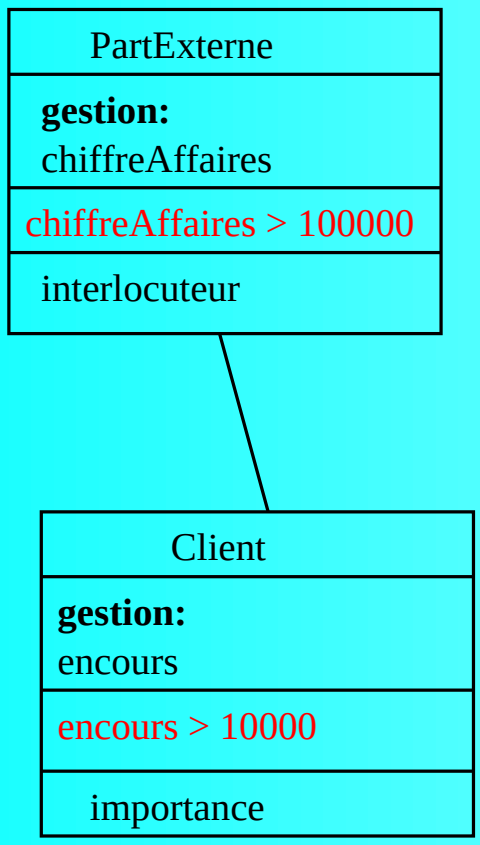
Partage d'informations inter-contextes



Plan fonctionnel "direction"

- √ En plus du partage avec le plan de base
- √ Notion d'accès inter-plan

Extension fonctionnelle



Fonction "direction"

- √ prédicat de participation associé à chaque classe pour chaque fonction
  - attributs issus du plan de base
  - attributs provenant d'accès inter-plans
- √ prise en compte pour le calcul de l'extension:
  - héritage des prédicats (conjonction)
  - contextes fournisseurs des accès inter-plans
  - cardinalités association, intégrité [Dexa2000]

Préservation des objets fonctionnels à l'exploitation

- √ Favoriser la tracabilité (évolution des fonctions, ajout)
- √ Un modèle abstrait favorise les multiples implémentations :
  - ▶ Programmation fonctionnelle :
    - Environnement développement SmallTalk [L'Objet 1997]
    - Langage ROME [Thèse Gilles 1999]
  - ▶ Système d'information :
    - Systèmes de vues : Cocoon, Multiview [Inforsid 97, Thèse laurent 98]
    - Modèle ODMG, bases de données objets java [LMO 2000]
    - Objets Corba [Tools 2000]

Vues, contextes et aspects

√ Les points communs :

- Séparation des aspects
- Dimension fonctionnelle des aspects :
  - Clause `introduction` dans AspectJ
  - `Role method` dans JAC

Vues, contextes et aspects

√ Les points communs :

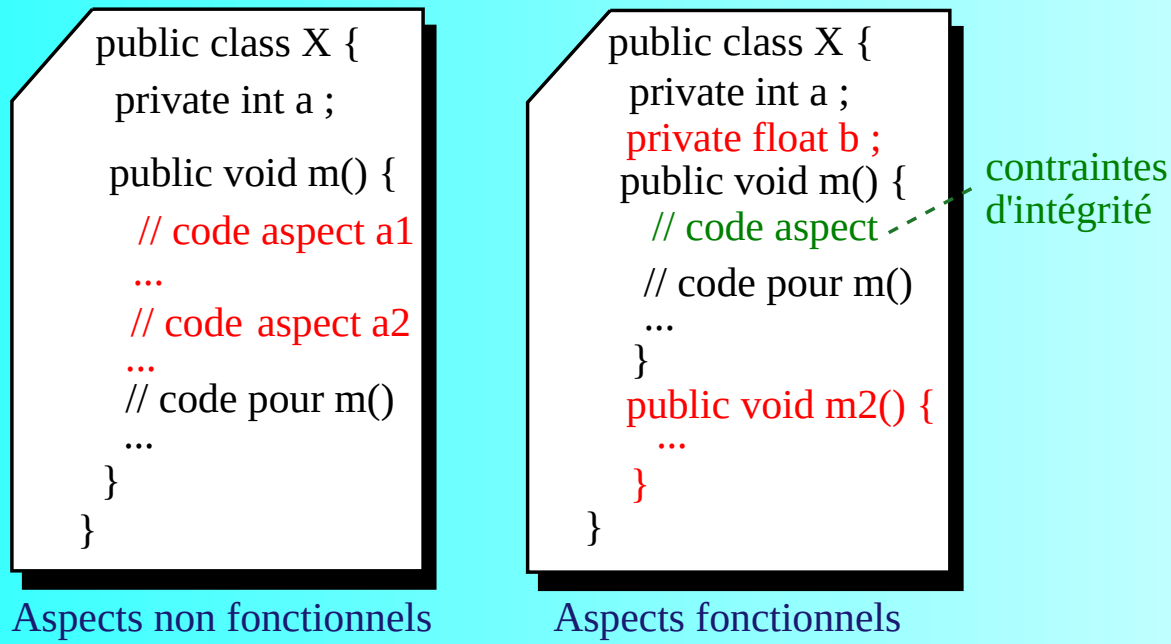
- Séparation des aspects
- Dimension fonctionnelle des aspects :
  - Clause `introduction` dans AspectJ
  - `Role method` dans JAC

√ Les distinctions :

- Pas uniquement aspects métiers
- Indépendance aspects (techniques) - application cible
- A.O.P et A.O.D (ex: UMLAUT)

√ *Roles, Subjects and Aspects, How do they relate* [Bardou (LIRMM), Ecoop'98]

Un mot sur le tissage



- √ L'aspect fonctionnel tient compte du plan de base (non indépendance)
- √ Refus d'homonymie (Outil CromeBrowser)

Des objets aux composants
Objectif : Réutilisation de composants fonctionnels

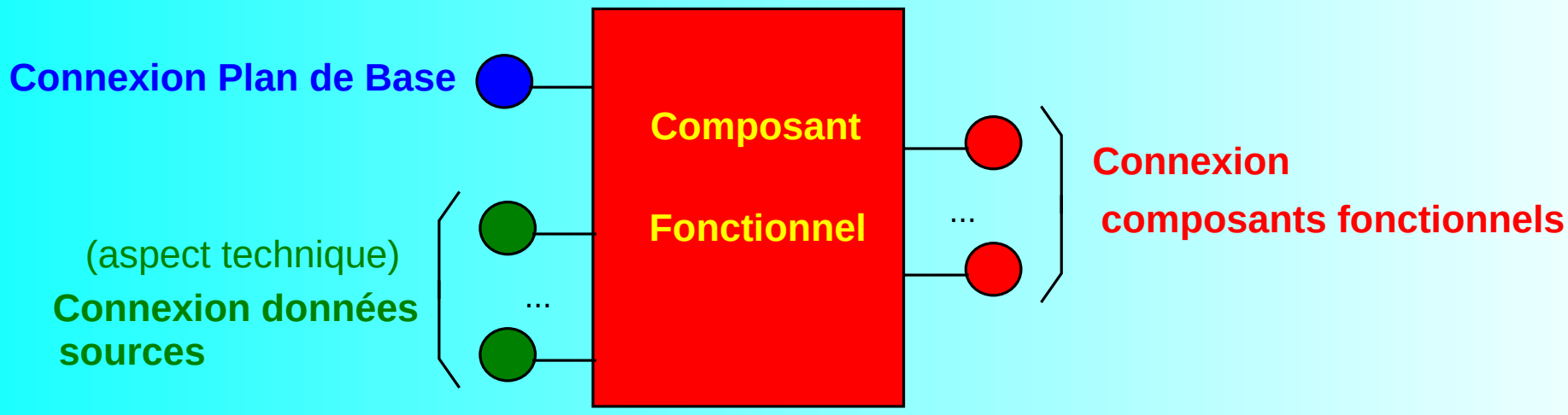
√ Préserver les avantages de la démarche objet :

- Structuration par plans adaptée aux architectures logicielles multi-niveaux
- Séparation des aspects (fonctionnels)
- Evolution
- Préservation des objets de conception à l'exploitation

√ Limites de la démarche objet actuelle :

- Conception from "scratch" (tissage plus facile :-))
- Plan de base et plan fonctionnels sont trop liés (réutilisation)

Le contrat d'assemblage composant fonctionnel

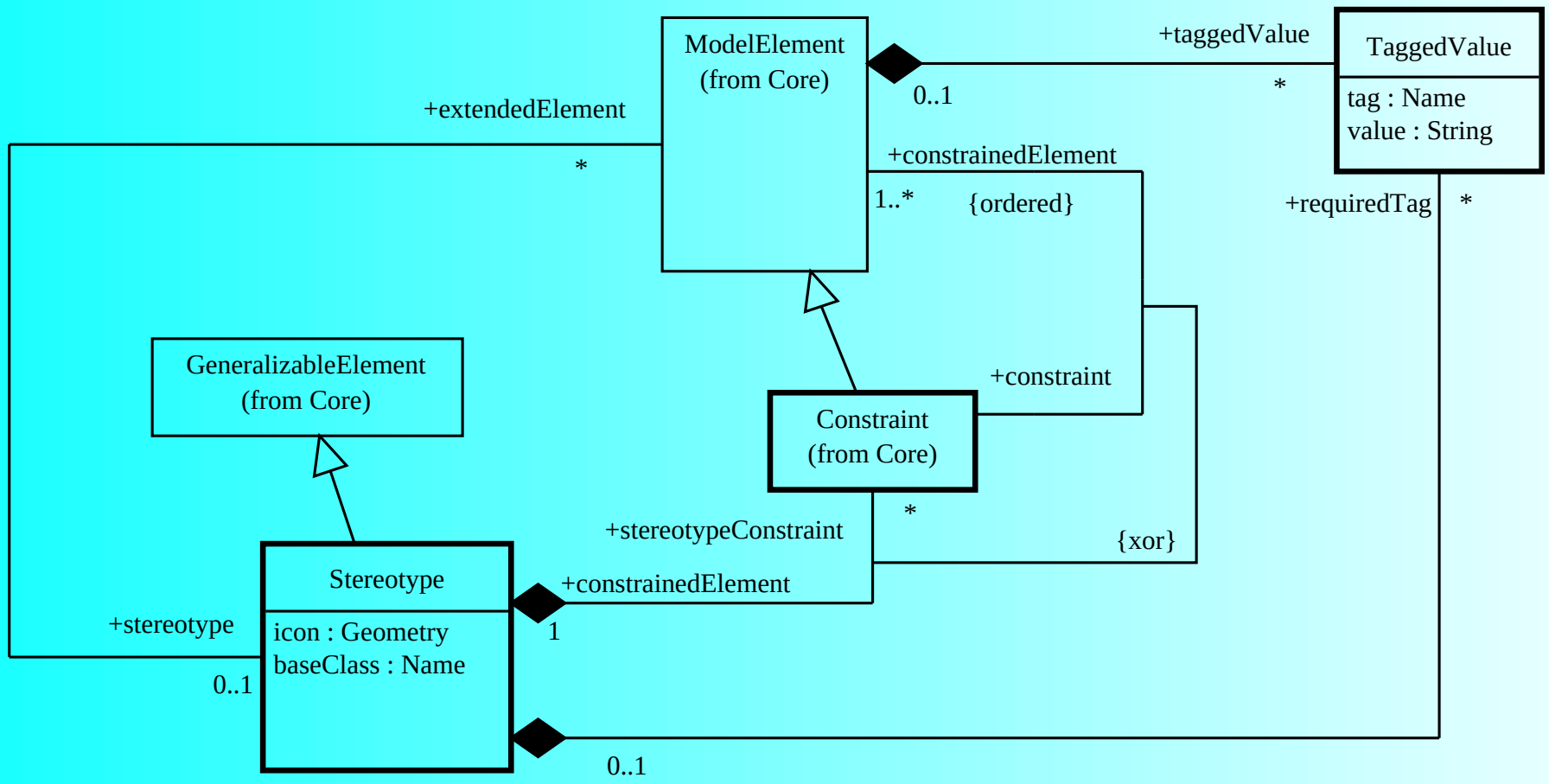


Vers une plateforme UML supportant la notion de composants fonctionnels

√ Les avantages d'un profil UML :

- On reste "standard"
- Exploitation d'outils du commerce (ex: Objecteering) :
  - Vérification de la cohérence du schéma UML "profilé" (OCL, J)
  - Génération de code (EJB, IDL3, UML,. . .)

Les profils UML



Le profil UML-Vue

- √ Découvrir l'Outil `Profile UML Builder` Objecteering
- √ Le profil réalisé :
  - Permet de décrire complètement le plan de base et les aspects fonctionnels.
  - Conception du profil : rendre les aspects plus autonomes (vers de vrais aspects !)
  - Vérification automatique cohérence du schéma
  - Génération de code (appliqué à Java RMI)
- √ Obtention d'un module utilisable avec version gratuite d'Objecteering

Le profil UML-Vue : définition du plan de base

- √ Définition d'un paquetage stéréotypé par `<<viewBasePackage>>`
- √ Contient un schéma de classes représentant les données du ou des bases de données sources

Le profil UML-Vue : Définition d'un aspect fonctionnel (1/2)

√ Construction d'un paquetage stéréotypé par `<<viewPackage>>`

√ Ce paquetage contient un schéma de classes, on distingue :

- Les classes propres au contexte (pas d'annotation spécifique).
- Les classes nécessaires au contexte et qui seront issues d'un plan de base (stéréotypée par `<<viewClass>>`)

Le profil UML-Vue : Définition d'un aspect fonctionnel (2/2)

- √ Les attributs d'une classe locale ne nécessitent pas d'annotation spécifique
- √ Les attributs d'une classe `<<viewClass>>` :
  - `<<viewBaseAttribute>>`
  - `<<viewAttribute>>`
  - `<<viewContextAttribute>>`
- √ Au niveau association, on distingue :
  - Celles issues du plan de base (`<<viewBaseAssociation>>`)
  - Celles introduites localement (`<<viewAssociation`)
  - Celles issues d'autres contextes (`<<viewContextAssociation>>`)

Remarques sur les paquetages fonctionnels

- √ La conception d'un aspect fonctionnel est relativement indépendante (on dispose de toutes les données).
- √ Autorise la conception par le haut :
  cas d'utilisation → aspect fonc. → connexion aux données sources (plan de base)

- √ Autorise la conception par le bas :
  Syst d'info. existant → plan de base → connexion aspects fonc.

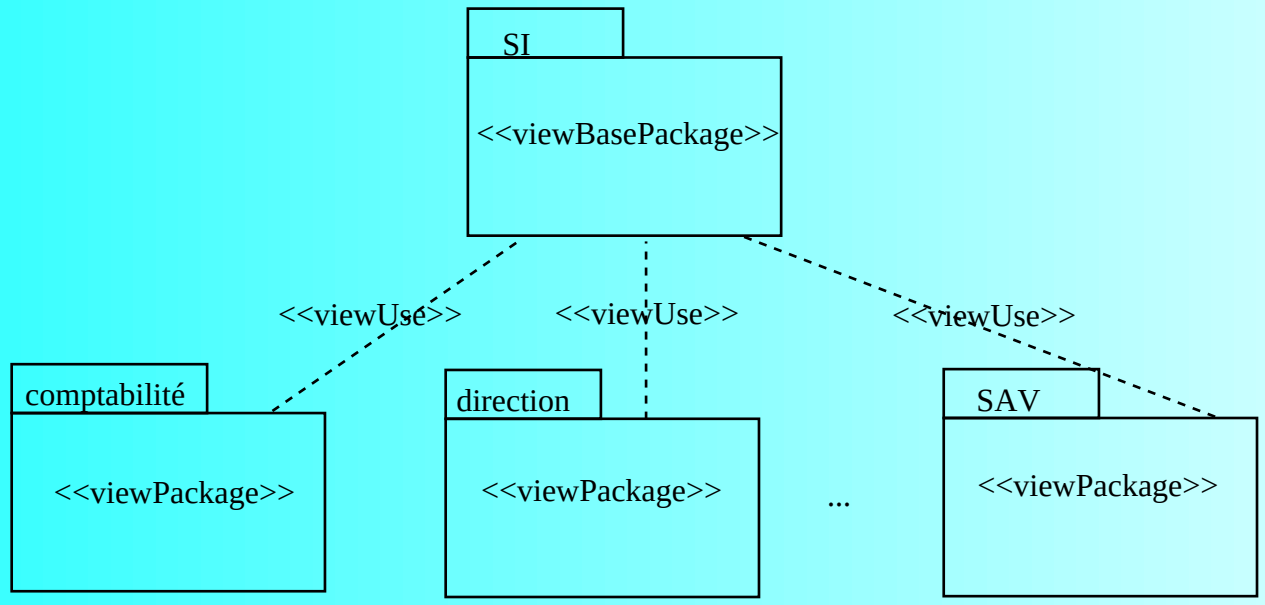
Eléments nécessaires au tissage

√ Pour construire une application, il est nécessaire de :

- connecter le ou les aspects fonctionnels à un plan de base
  - relier les `<<viewClass>>`
  - relier les `<<viewBaseAttribute>>`
  - relier les `<<viewBaseAssociation>>`
- connecter les accès inter-plans :
  - relier les `<<viewContextAttribute>>`
  - relier les `<<viewContextAssociation>>`
- connecter les données fonctionnelles à des données sources (ex: sgbd)

Tissage vers un plan de base (1/2)

√ Utilisation d'un lien de dépendance stéréotypée par `<<viewUse>>` :



Tissage vers un plan de base (2/2)

√ Spécification de valeurs marquées (tagged value) requises par les stéréotypes :

stéréotype	**tagged value**	**signification**
`<<viewClass>>`	`classSrc`	nom classe pb
`<<viewBaseAttribute>>`	`attributeSrc`	nom attribut pb
`<<viewBaseAssociation>>`	`associationSrc`	nom asso. pb

√ Renommage possible (indépendance aspect), non obligatoire

Tissage vers composants fonctionnels

√ Spécification de valeurs marquées (tagged value) requises par les stéréotypes :

stéréotype	**tagged value**	**signification**
`<<viewContextAttribute>>`	`contextSrc`	nom contexte
	`classSrc`	nom classe
	`attributeSrc`	nom attribut
`<<viewContextAssociation>>`	`contextSrc`	nom contexte
	`assoSrc`	nom asso

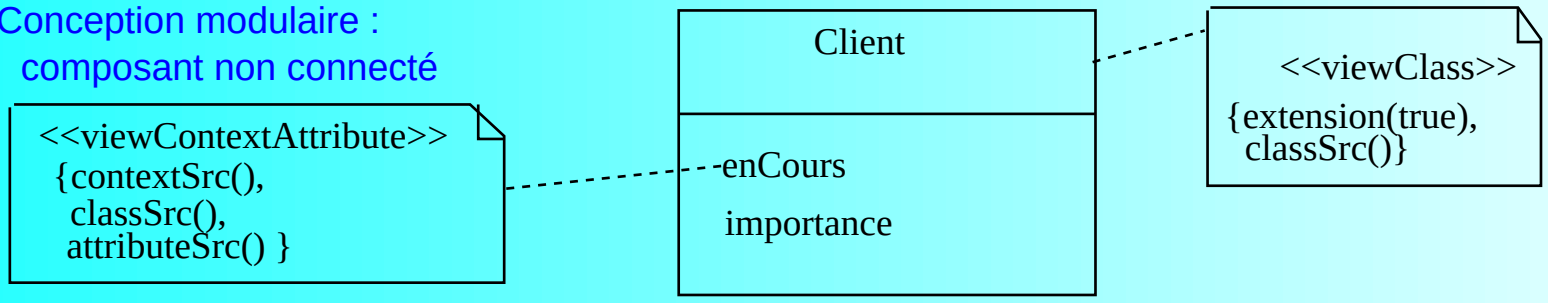
Tissage vers des données fonctionnelles

√ Spécification de valeurs marquées (tagged value) requises par les stéréotypes :

stéréotype	**tagged value**	**signification**
`<<viewAttribute>>`	`attributeSrc`	requête d'accès base (style OQL)
`<<viewAssociation>>`	`associationSrc`	requête d'accès base

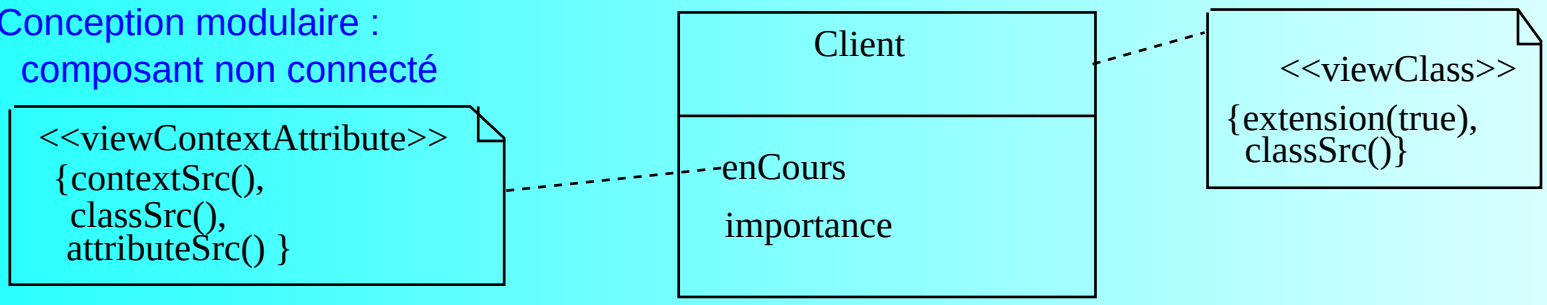
Illustration

Conception modulaire :
composant non connecté

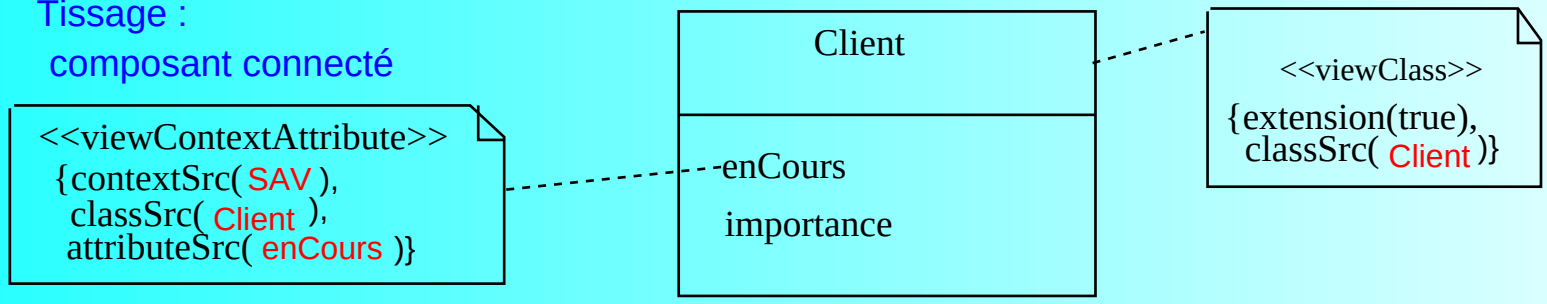


Illustration

Conception modulaire :
composant non connecté



Tissage :
composant connecté



Cycle de vie du composant fonctionnel

1. Phase de conception autonome au sein d'un paquetage fonctionnel

   √ Définition du schéma de classes
   √ Définition des stéréotypes, contraintes,. . .
   √ Outil UML : vérification de la cohérence du schéma

   ex: un stéréotype `viewAttribute` ne s'applique qu'aux attributs d'une classe stéréotypée par `viewClass`.
2. Connexion du composant fonctionnel à un composant de base et aux composants fonctionnels (accès inter-plan).

   √ 'Remplissage' des différentes valeurs marquées requises par les stéréotypes.
   √ Outil UML : Vérification du composant connecté (similitude `configuration_complete` de CCM)

Evolutions du profil

- √ le module objecteering fonctionne mais :
- √ Améliorer le lien entre schéma et données sources (paquetage `<<viewBasePackage>>`)
- √ Mieux formaliser l'ensemble des stéréotypes (oql ?, ocl ?)
- √ Génération de code

Conception par aspects et les modèles de composants

- √ But : préserver les composants fonctionnels à l'exploitation
- √ Deux modèles étudiés : CCM et EJB
- √ Etude des caractéristiques de chacun et proposition d'un mapping adapté au modèle

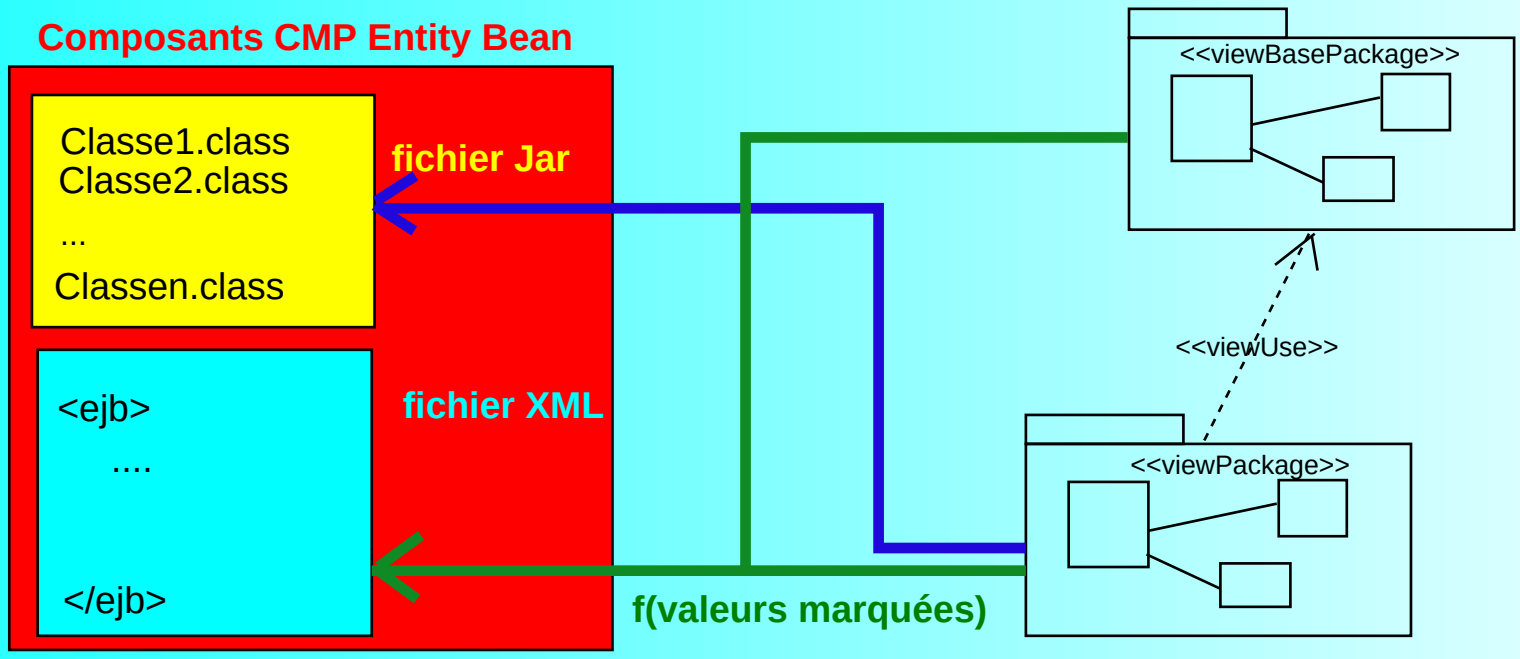
Le modèle EJB 2.0

- √ Définition de composants orientés données (Entity Bean)
- √ Définition d'un schéma de persistance abstrait (entre CMP Entity Bean)
  - Définition des relations (cardinalité, navigation).
  - Gestion automatique de l'intégrité référentielle.
- √ Définition d'un langage de requête EJB-QL (lien avec SGBD)
- √ Possibilité d'avoir deux interfaces : une interface distante et une interface locale.

Des composants fonctionnels de conception aux composants EJB

- √ Schéma UML du paquetage fonctionnel → schéma abstrait de persistance
- √ Une classe du paquetage fonctionnel → CMP Entity Bean
- √ Informations issues du plan de base → requêtes EJB-QL
- √ Informations d'autres contextes → requêtes EJB-QL

Le composant fonctionnel EJB

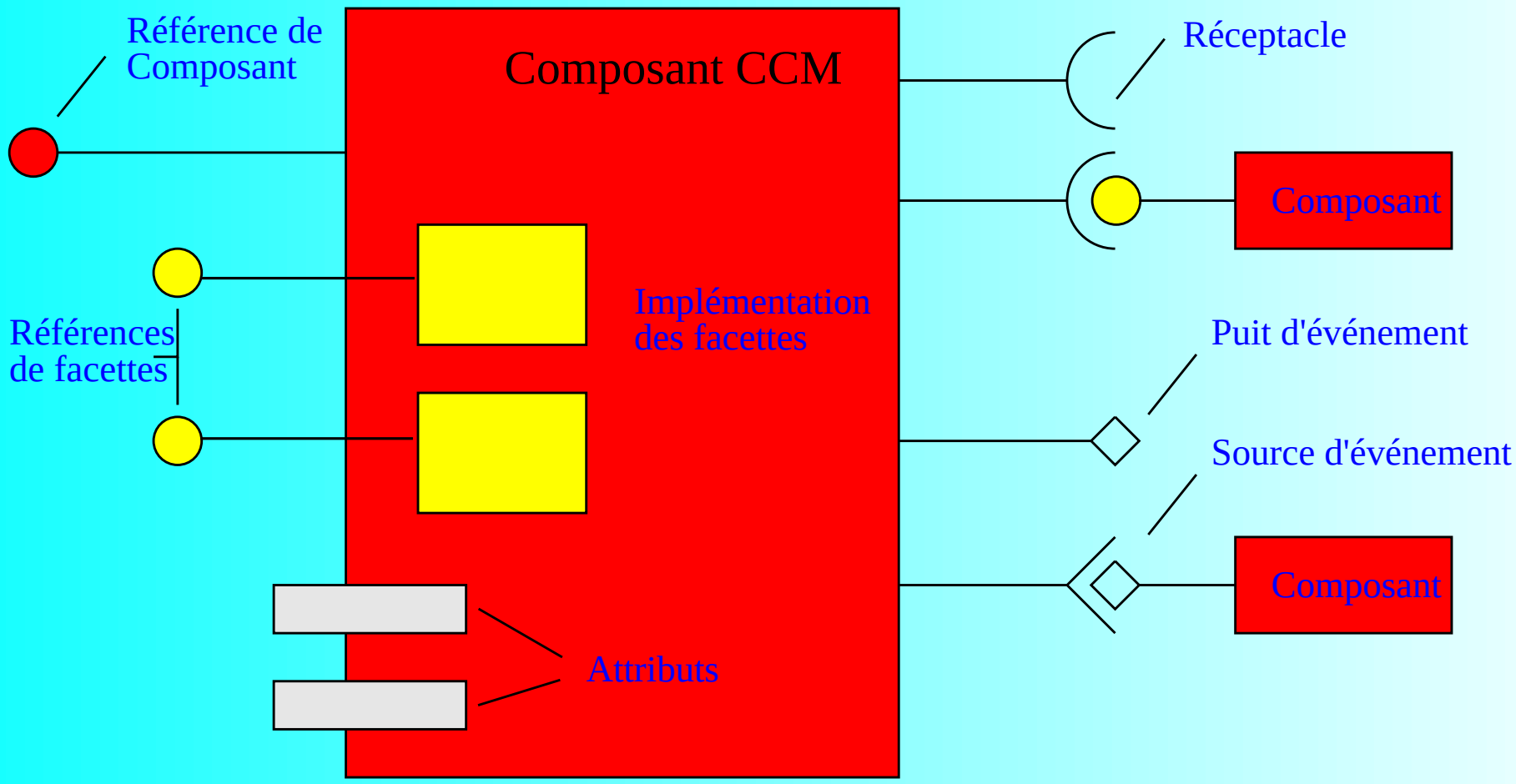


- √ Les fichiers Java ne sont pas modifiés (robuste, réutilisation)
- √ Modification de certaines sections du descripteur XML

Quelques remarques

- √ Pas de couche logicielle supplémentaire
- √ Ajout/retrait des composants fonctionnels
- √ Processus automatisable mais :
- √ Forte distinction entre composants de conception et composants à l'exploitation (mapping EJB choisi : plus de composants connectés).
- √ Tracabilité moyenne (sauf si héritage d'interfaces)
- √ Une modification d'un composant peut faire modifier la partie descripteur d'un autre composant :-(
- √ Le cadre abstrait favorise **plusieurs mappings** :
  - ▶ Illustration avec CCM et des composants fonctionnels à l'exploitation connectables

Un petit mot sur CCM (© rafi)

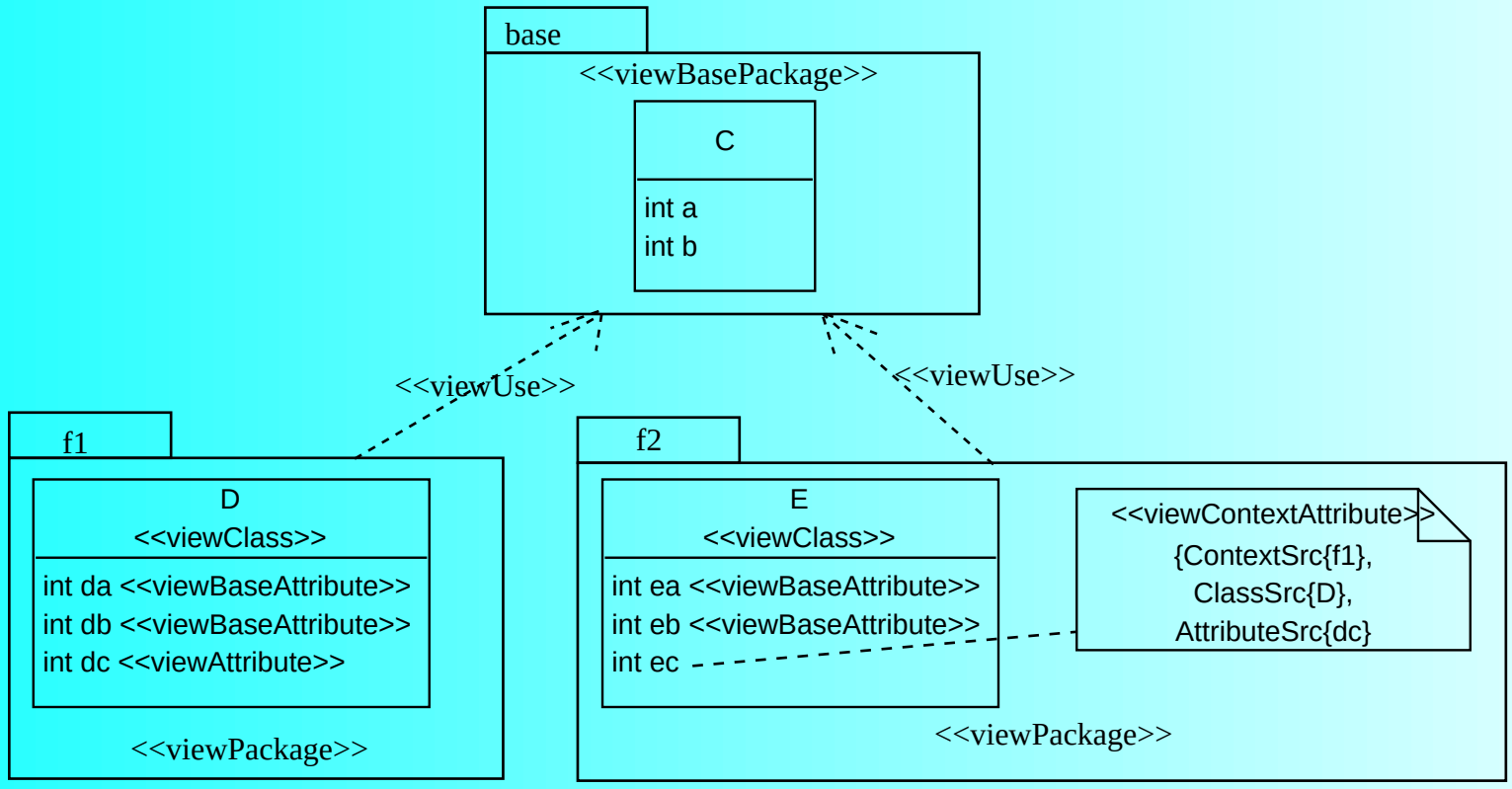


Un mauvais mapping

√ Utilisation des facettes pour décrire chaque vue :

- Les fonctions sont englobées dans le composant d'exploitation
- plus de notion de modularité
- Ajout d'un composant fonctionnel → modifier le composant CCM :-(

Autre solution : un exemple



Mapping corba CCM : composant de base

```
import Components ;

module base {
        interface CFacet {
                attribute long a ;
                attribute long b ;
        } ;
        component C {
                provides CFacet serviceC ;
        } ;
        home CHome manages C { } ;
} ;
```

Mapping corba CCM : composant fonctionnel

```
module f1 {
        interface AdaptateurBase {
          attribute long da ;
          attribute long db ;
        } ;
        interface DFacet  : AdaptateurBase{
          attribute long dd ;
        } ;
        component D {
          provides DFacet serviceD ;
          uses AdaptateurBase connectBase ;
        } ;
        home DHome manages D { } ;
};
```

Mapping corba CCM : composant fonctionnel

```
module f2 {
  interface AdaptateurBase {
     attribute long ea ; attribute long eb ;
  } ;
  interface Adaptateurf1 {
     attribute long ed ;
  } ;
  interface EFacet : AdaptateurBase, Adaptateurf1{} ;
  component E {
     provides EFacet serviceE ;
     uses AdaptateurBase connectBase ; uses Adaptateurf1 connectF1 ;
  } ;
  home EHome manages E { } ;
};
```

Mapping composant CCM fonctionnel

- √ Composant autonome
- √ Génération IDL3 entièrement automatique
- √ Atelier UML équivalent à la BeanBox (génère des adaptateurs)
- √ Ajout/retrait/changement dynamique de composants CCM fonctionnels possible
- √ Tracabilité complète
- √ Ce choix de mapping est applicable aux EJB moyennant une couche logicielle supplémentaire (Etendre `javax.ejb.EJBObject`)

Commentaires

√ Ce qu'on fait :

- Utilisation d'UML et profils UML
- Séparation des aspects fonctionnels
- Génération de composants
- Distinction code (robuste) et fichiers de déploiement (tissage)

Perspectives (1/2)

√ Introduction d'aspect technique

- Spécification complète du fichier déploiement
- modèle abstrait d'aspect techniques
- profil de génération suivant une plateforme cible :
  exemple : EJB → persistance, transactions, sécurité

√ Enrichissement du modèle :

- données calculées,. . .

Perspectives (2/2)

- √ Objectif :Architecture client/serveur n-tiers
- √ Représentation multi-niveaux :
  - Paquetage base : représentation SI partagé
  - Paquetages fonctionnels : objets du domaine contextualisés
  - Paquetage services : services métiers contextualisés
  - Paquetage IHM : ex: composants webs
- √ Indépendance entre chaque niveau (réutilisation)
  ex: réutiliser un composant web d'authentification
- √ Obtenir une démarche complète : modèle d'analyse → modèle de conception → modèle d'exploitation