UNIVERSITÉ DE PAU ET DES PAYS DE L'ADOUR

Laboratoire LIUPPA – Equipe Self-*

Contrats de transformation pour la validation de raffinement de modèles

Eric Cariou, Nicolas Belloir, Franck Barbier

Laboratoire LIUPPA – Equipe Self-*
Université de Pau et des Pays de l'Adour

5èmes journées IDM – 25 Mars 09 – Nancy

LIUPPA

Contexte

- Processus logiciel basé sur l'ingénierie dirigée par les modèles
  - Composé d'un ensemble de transformations
  - Exécutées / réalisées en séquence
    - De manière entièrement automatique ou pas
  - Le concepteur peut intervenir sur les modèles générés par les transformations
- Nécessité de vérifier
  - Que les transformations sont valides
  - Encore plus important quand le concepteur intervient sur les modèles

Contexte

- Transformations
  - Dans un processus IDM, beaucoup correspondent à des raffinements
    - Ajout de détails, d'informations mais sans changer le but, le sens du modèle
  - Nous nous plaçons dans ce contexte
  - Et de manière plus générale : contexte des transformations endogènes
    - A méta-modèle constant
    - Un raffinement est un cas particulier de transformation endogène

Objectif

- Pouvoir valider qu'un couple de modèles est le résultat valide d'une transformation
- De la manière la plus ouverte possible
  - Doit pouvoir manipuler des modèles obtenus par une grande variété d'outils
  - Pas d'a priori sur la façon dont la transformation est réalisée
    - Potentiellement « à la main »
  - Définition d'une méthode générale
- Notre solution
  - Définition de contrats de transformation de modèles écrits en OCL (*Object Constraint Language* [OMG])

OCL pour l'expression des contrats

- OCL : *Object Constraint Language* [OMG]
- Justification de l'utilisation d'OCL
  - Naturellement adapté à l'écriture de contrats
    - Invariants, pré, post-conditions sur opérations
  - Ouvert
    - Peut s'appliquer sur des modèles UML, MOF, Ecore ...
  - De plus en plus utilisé
    - Intégré dans des AGLs, utilisé ou étendu dans des langages de transformations (ATL, QVT...)
  - Formel mais relativement simple d'utilisation
    - Accessible au concepteur « lambda »

Contrat de transformation

- Conception par contrat
  - Spécification d'un élément logiciel
    - Invariants qui doivent être respectés en permanence
    - Pré et post-conditions sur opérations
      - Contraintes d'utilisation
      - Contraintes sur le résultat attendu
- Contrat de transformation
  - Application de ces principes à une opération de transformation de modèles
    - Spécification de la transformation

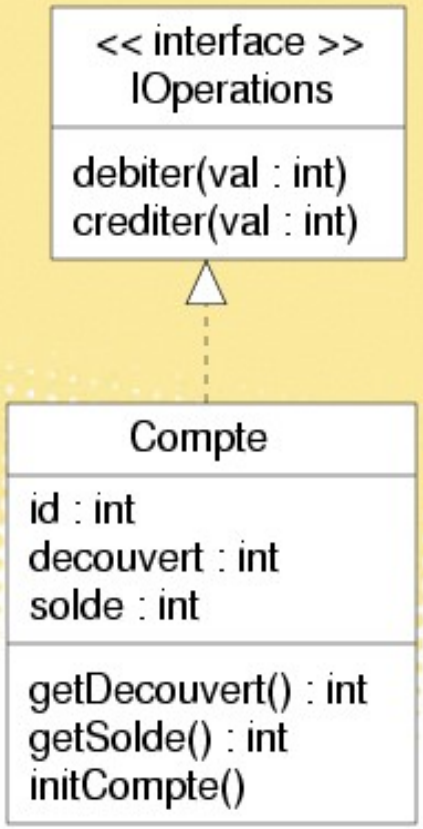
Contrat de transformation

- Transformation
  - Prend en entrée un *modèle source* et fournit en sortie un *modèle cible*
- Spécification d'un contrat de transformation
  - Contraintes à respecter par le modèle source
    - Pour pouvoir être transformé
  - Contraintes à respecter par le modèle cible
    - Pour pouvoir être considéré comme un résultat valide de la transformation par rapport au modèle source
    - Se décompose en deux ensembles de contraintes
      - Contraintes « générales » sur le type de modèle cible
      - Contraintes de relations/évolutions entre le modèle source et le modèle cible

Exemple de raffinement

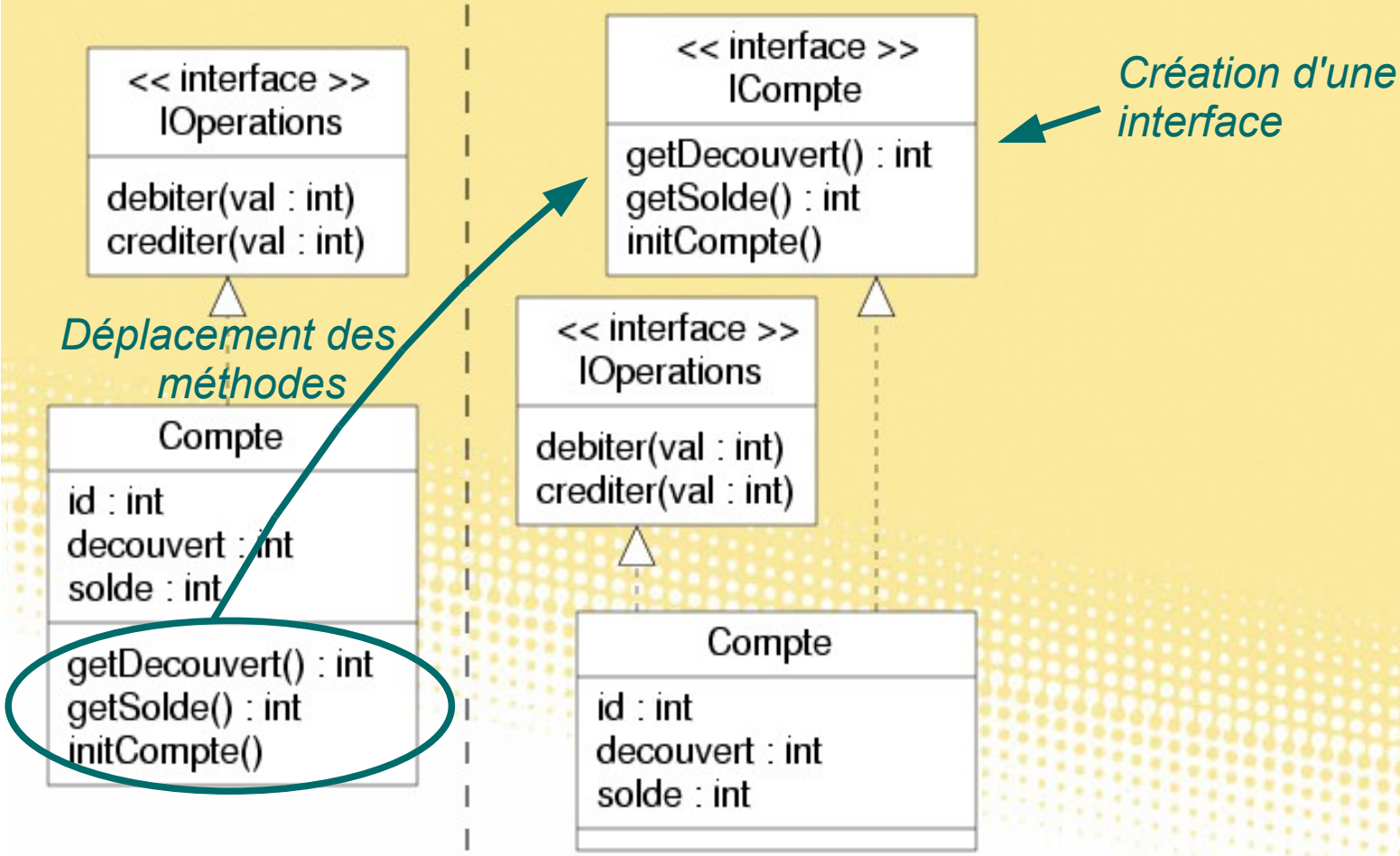
- Sur un diagramme de classe « à la UML »
  - Ajout d'interface sur chacune des classes
- Réalisation du raffinement
  - Première étape : génération automatique d'une interface
    - Création d'une interface par défaut
    - Déplacement des méthodes de la classe vers l'interface
  - Deuxième étape : ré-agencement éventuel du diagramme par le concepteur

Exemple de raffinement



Modèle initial (source)

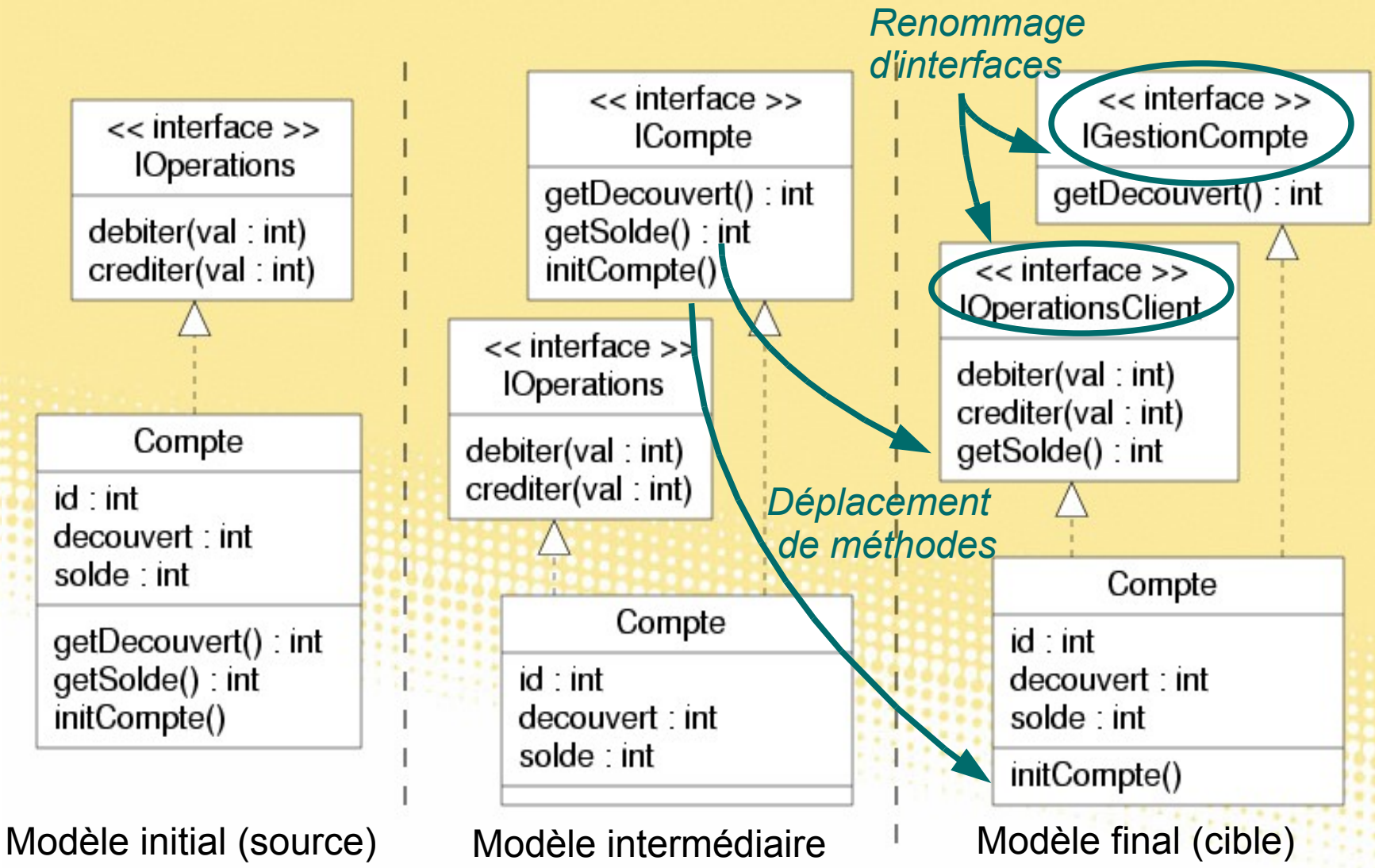
Exemple de raffinement



Modèle initial (source)

Modèle intermédiaire

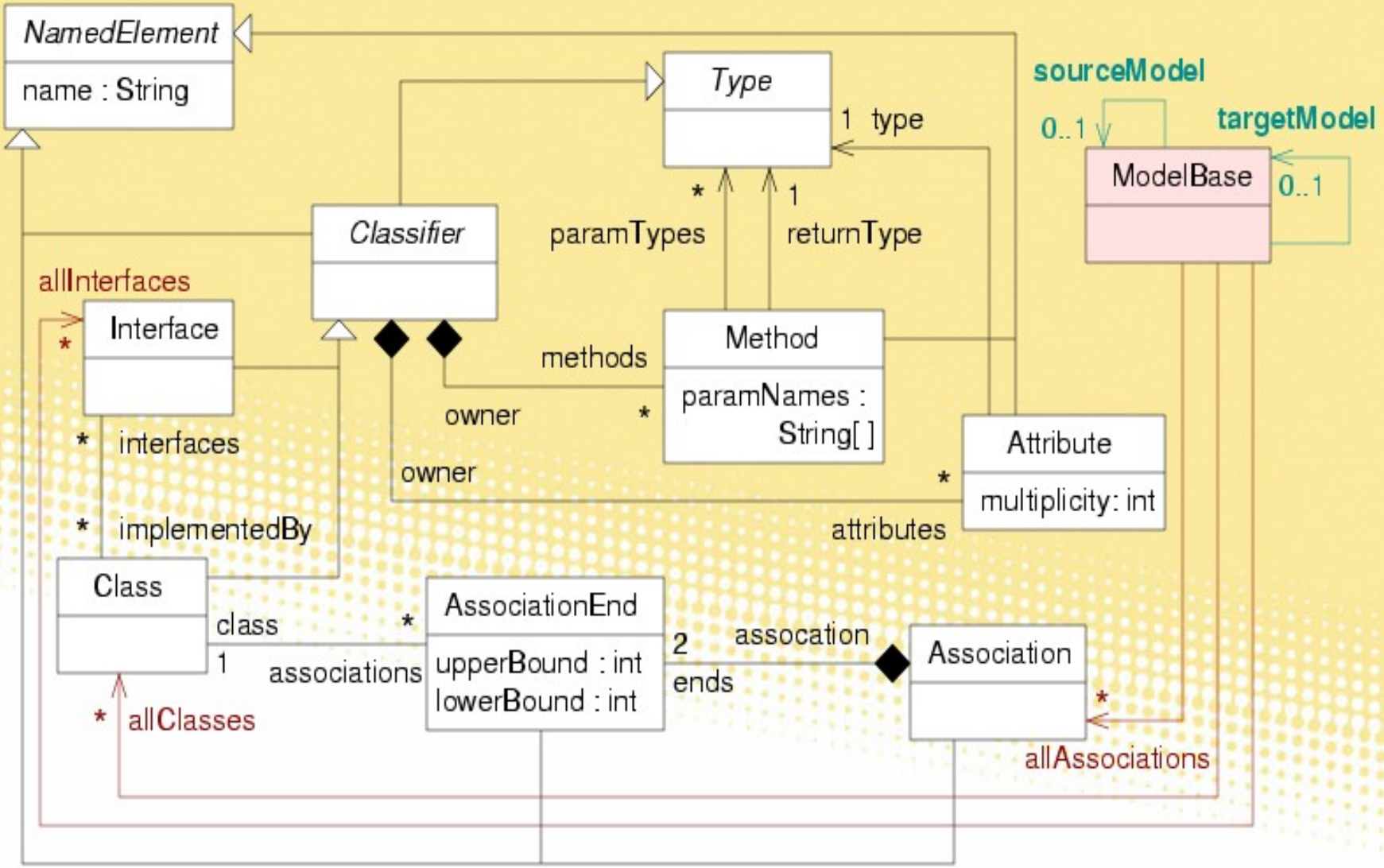
Exemple de raffinement



Exemple de raffinement

- Contrat associé à ce raffinement
  - Contraintes sur le modèle source
    - Aucune, peut raffiner n'importe quel diagramme
  - Contraintes générales sur le modèle cible
    - Chaque classe doit implémenter au moins une interface
  - Contraintes d'évolution entre le modèle source et le modèle cible
    - Chaque classe après le raffinement doit implémenter les mêmes méthodes qu'avant
      - Directement ou via ses interfaces

Méta-modèle de diagramme de classe simplifié



Contrat pour l'ajout d'interfaces

- Contraintes générales sur le modèle cible
  - Une classe implémente au moins une interface
  - S'exprime facilement en OCL

    context Class
    inv: interfaces -> notEmpty()
- Contraintes d'évolution entre source et cible
  - Chaque classe implémente toujours les mêmes méthodes
  - Deux problèmes pour écrire ces contraintes
    - Pouvoir manipuler les 2 modèles simultanément
    - Pouvoir définir que telle classe du cible correspond à telle classe du source

Manipulation simultanée des 2 modèles

- Idée intuitive
  - Spécifier l'opération de transformation via un couple de pré et post-condition
    - Pré-condition : référence modèle source
    - Post-condition : référence modèle cible
    - Opérateur @pre d'OCL : dans post-condition, peut référencer des éléments du cible et du source
  - Outil de transformation vérifie les contraintes
  - Exemple
    - **context** ModelBase::addInterfaces()
      pre: - - *contraintes sur le modèle source*
      post: - - *contraintes sur le modèle cible et*
      - - *contraintes d'évolution entre le source et le cible*
      allClasses -> size() = allClasses@*pre* -> size() **and** ...

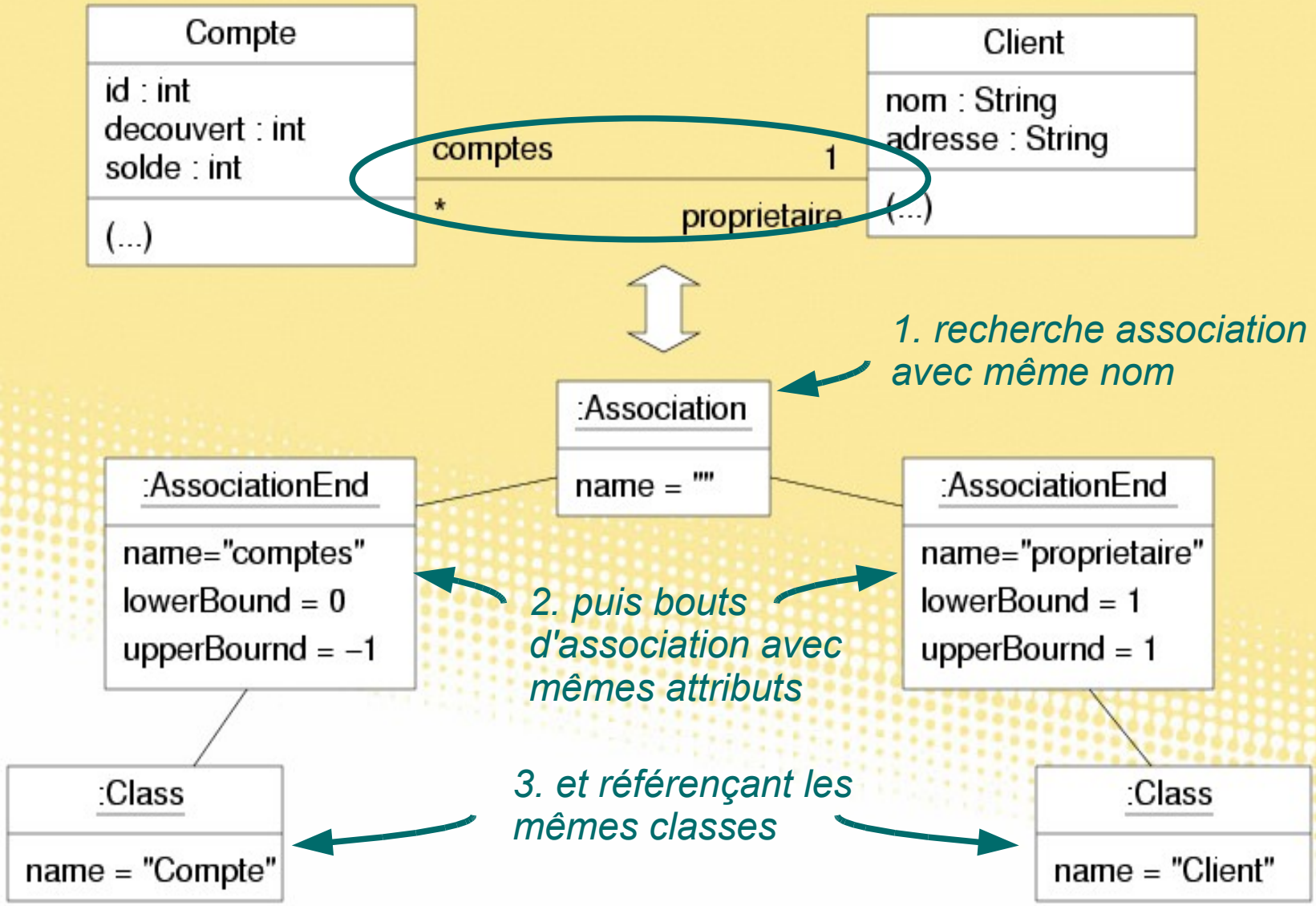
Manipulation simultanée des 2 modèles

- Limites fortes de l'approche par pré/post
  - Valide que l'*exécution* de la transformation est correcte
    - Ne permet pas une validation *a posteriori*
    - Pas de prise en compte des modifications manuelles
- Autre solution
  - Concaténation des 2 modèles en un 3ème plus global
  - Vérification de contraintes OCL sur ce modèle global
    - Ensemble d'invariants
  - Prévoir un mécanisme de concaténation
    - Ex. diagrammes UML : un modèle avec 2 packages
    - Lors de la définition d'un DSL : à prévoir dès le départ
      - Pour notre exemple : associations source/targetModel

Correspondance entre éléments

- Pour un élément, besoin de récupérer son élément correspondant du même type dans l'autre modèle
- Exemples
  - La classe Compte du source correspond à la classe Compte du cible
    - Cas simple : on cherche une classe avec le même nom
    - Car unicité des noms de classe
  - Une association entre 2 classes
    - Ne peut pas se contenter de comparer le nom (pas d'unicité des noms ni même obligation de nommage)
    - Doit vérifier que leurs bouts d'associations sont les mêmes
      - Mêmes noms, multiplicités et classes associées
    - Doit vérifier que les classes sont les mêmes

Exemple : correspondance association



Correspondance entre éléments

- Nécessité de définir explicitement des fonctions de correspondances
  - Qui s'appellent de manière transitive
- 3 types de correspondances
  - Totale
    - Tous les attributs et références d'un type d'élément sont en correspondance
  - Partielle
    - Une partie des attributs et références sont en correspondance
  - Pas de correspondance

Exemples de correspondance

- Pour éléments de type Class
  - Correspondance partielle
    - S'applique que sur les attributs de la classe et ses associations
    - Car la liste des méthodes et des interfaces change
- Pour éléments de type Association
  - Correspondance totale
    - Sur son nom, ses bouts d'association
- Pour éléments de type Interface
  - Pas de correspondance
    - Les interfaces changent pendant le raffinement

Fonctions de correspondance

- Ecriture en OCL des correspondances
  - Simple mais fastidieux
    - Nombreuses fonctions selon le niveau de transitivité
  - Problème relatif car leur génération est entièrement automatisable
- Exemple : fonctions de correspondance pour Class
  - **context** Class **def**: classMapping(cl : Class) : Boolean = self.name = cl.name **and** self.sameAttributes(cl)
  - **context** Class **def**: hasMappingClass(mb : ModelBase) : Boolean = mb.allClasses -> exists(cl | self.classMapping(cl))
  - **context** Class **def**: getMappedClass(mb : ModelBase) : Class = mb.allClasses -> any (cl | self.classMapping(cl))

Exemple : contraintes d'évolution

- Exemple d'ajout d'interface
  - Ecriture de la partie du contrat sur l'évolution des éléments
    1. Pour chaque classe du cible, vérifier qu'elle est en correspondance partielle avec une classe du source
       - Si ça n'est pas le cas, le contrat n'est pas respecté
    2. Récupèrer la liste des méthodes de la classe et de sa classe en correspondance
       - Les méthodes de la classe et de ses interfaces
    3. Vérifier que ces listes contiennent les mêmes méthodes
       - Si ça n'est pas le cas, le contrat n'est pas respecté
  - Utilise des fonctions de correspondance pour
    - Classe, attribut, méthode, ensemble de méthodes…

Détail sur contrat : évolution des classes

- **context** ModelBase **inv** checkInterfaceContract:
  targetModel.sameClasses(sourceModel)
- **context** ModelBase **def**: sameClasses(mb : ModelBase) : Boolean =
  self.allClasses -> size() = mb.allClasses -> size() **and**
  self.allClasses -> **forAll**(c |
  if c.hasMappingClass(mb)
  then
  let myMethods : Set(Method) = c.interfaces -> collect(i |
  i.methods) -> union(c.methods) -> flatten() **in**
  let eqClass : Class = c.getMappedClass(mb) **in**
  let eqClassMethods : Set(Method) = eqClass.interfaces ->
  collect(i | i.methods) -> union(eqClass.methods) -> flatten() **in**
  c.sameMethodSet(myMethods, eqClassMethods)
  else
  false
  endif)

Contrôle de l'intervention manuelle

- Intervention manuelle du concepteur
  - Généralement limitée à un cadre précis
  - Pour notre exemple
    - Peut modifier les interfaces et déplacer les méthodes
    - Mais pas de changement des attributs des classes, des associations ...
- Autre intérêt des fonctions de correspondance
  - Permettent facilement de définir un contrat vérifiant la non modification d'une partie du modèle
    - Pour certains types d'élément, il faut une correspondance de chaque élément entre le source et le cible
  - Génération automatisable de ce contrat

Résumé méthode de validation

- Prévoir un mécanisme pour concaténer 2 modèles en un seul
- Pour une transformation, définir un contrat
  - 3 ensembles de contraintes
    - Sur le source, le cible et l'évolution entre les deux
    - Se basent sur des fonctions de correspondance
  - Contrat peut inclure la vérification de la non-modification d'une partie du modèle
- Pour un couple de modèles, validation du contrat
  - Réalisée par un évaluateur OCL standard
  - Contraintes d'évolution sont vérifiées sur le modèle concaténant le source et le cible

Conclusion

- Proposition
  - D'une méthode générale de validation de transformations endogènes
    - La plus ouverte possible
  - Basée sur des contrats de transformations
    - Avec application en OCL
- Mise en avant
  - De la nécessité de définir explicitement les correspondances entre éléments des 2 modèles
    - Structuration du contrat
  - Et permettront de valider facilement qu'une partie du modèle n'est pas modifiée

Perspectives

- Développement d'outils de génération automatique de fonctions de correspondance
  - Stage Master Recherche en cours
- Généralisation aux transformations exogènes
  - Problème principal
    - Expression de contraintes OCL pour 2 méta-modèles différents à la fois
    - Idée : concaténation de 2 méta-modèles en un 3ème plus global