CENTRALE
L Y O N

Génie Logiciel

OCL:

Object Constraint Language

Le langage de contraintes d'UML

BTD/GL/OCL 1

CENTRALE
L Y O N

Bertrand DAVID : Génie Logiciel

Plan

1. Pourquoi OCL ? Introduction par l'exemple
2. Les principaux concepts d'OCL
3. Exemple d'application sur un autre modèle
4. Utilisation en pratique d'OCL lors d'un développement logiciel

BTD/GL/OCL 2

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

Exemple d'application (vu dans Eiffel)

- Application bancaire :
  - ➔ Des comptes bancaires
  - ➔ Des clients
  - ➔ Des banques
- Spécification :
  - ➔ Un compte doit avoir un solde toujours positif
  - ➔ Un client peut posséder plusieurs comptes
  - ➔ Un client peut être client de plusieurs banques
  - ➔ Un client d'une banque possède au moins un compte dans cette banque
  - ➔ Une banque gère plusieurs comptes
  - ➔ Une banque possède plusieurs clients

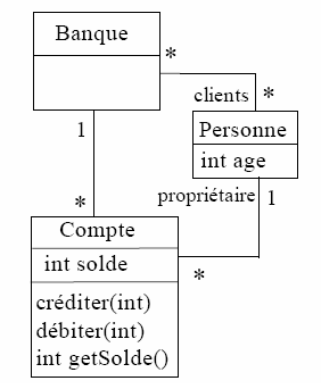
BTD/GL/OCL

3

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

Diagramme de classes



BTD/GL/OCL

4

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

Manque de précision

- Le diagramme de classes ne permet pas d'exprimer tout ce qui est défini dans la spécification informelle
- Exemple :
  - ➔ Le solde d'un compte doit toujours être positif
  - ➔ ajout d'une contrainte sur cet attribut
- Le diagramme de classe permet-il de détailler toutes les contraintes sur les relations entre les classes ?

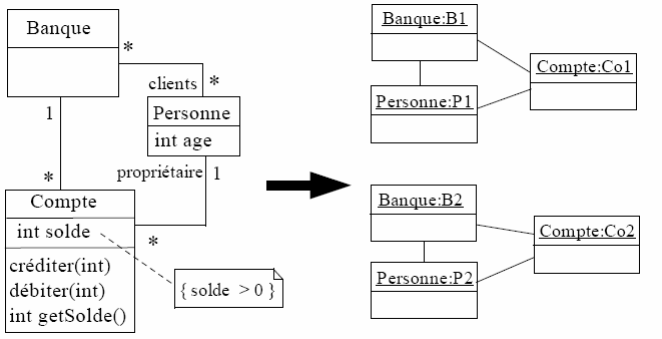
BTD/GL/OCL

5

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

Diagramme d'instances



- Diagramme d'instances valide vis-à-vis du diagramme de classes et de la spécification attendue

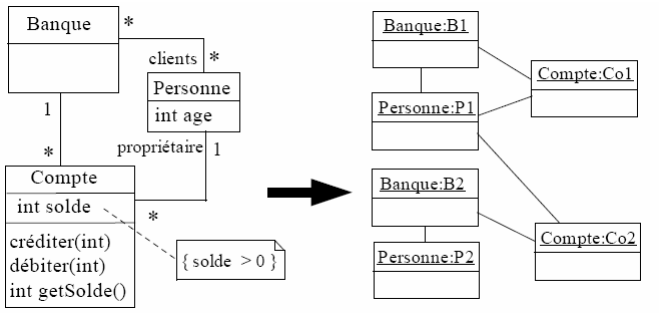
BTD/GL/OCL

6

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

Diagramme d'instances



- Diagramme d'instances valide vis-à-vis du diagramme de classes mais ne respectant pas la spécification attendue :
  - ➔ Une personne a un compte dans une banque où elle n'est pas cliente
  - ➔ Une personne est cliente d'une banque mais sans y avoir de compte

BTD/GL/OCL

7

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

Diagrammes UML insuffisants

- Pour spécifier complètement une application :
  - ➔ Diagrammes UML seuls sont généralement insuffisants
  - ➔ Nécessité de rajouter des contraintes
- Comment exprimer ces contraintes ?
  - ➔ Langue naturelle mais manque de précision, compréhension pouvant être ambigüe
  - ➔ Langage formel avec sémantique précise : par exemple OCL
- OCL : *Object Constraint Language*
  - ➔ Langage de contraintes orienté-objet
  - ➔ Langage formel (mais simple à utiliser) avec une syntaxe, une grammaire, une sémantique (manipulable par un outil)
  - ➔ S'applique sur les diagrammes UML

BTD/GL/OCL

8

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

Plan

1. Pourquoi OCL ? Introduction par l'exemple
2. *Les principaux concepts d'OCL*
3. Exemple d'application sur un autre modèle
4. Utilisation en pratique d'OCL lors d'un développement logiciel

BTD/GL/OCL

9

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

Le langage OCL

- OCL fait partie de la norme UML 1.3 (et sup.) de l'OMG (*Object Management Group*)
- OCL en version 2.0 : spécification à part de la norme UML 2.0, en cours de normalisation par l'OMG
- OCL permet principalement d'exprimer deux types de contraintes sur l'état d'un objet ou d'un ensemble d'objets :
  - ➔ Des invariants qui doivent être respectés en permanence
  - ➔ Des pré et post-conditions pour une opération :
    - Précondition : doit être vérifiée avant l'exécution
    - Postcondition : doit être vérifiée après l'exécution
- Attention : une expression OCL décrit une contrainte à respecter et pas le « *code* » d'une méthode

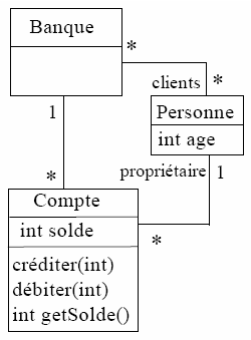
BTD/GL/OCL

10

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

Usage d'OCL sur l'application banquaire



context Compte
inv: solde > 0
context Compte :
débiter(somme : int)
pre: somme > 0
post: solde = solde@pre - somme
context Compte
inv: banque.clients ->
includes (propriétaire)

Avantage d'OCL : langage formel permettant de préciser clairement de la sémantique sur les modèles UML

BTD/GL/OCL

11

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

Utilisation d'OCL

- OCL peut s'appliquer sur la plupart des diagrammes UML
- Il sert, entre autres, à spécifier des :
  - ➔ Invariants sur des classes
  - ➔ Pré et post-conditions sur des opérations
  - ➔ Gardes sur transitions de diagrammes d'états ou de messages de diagrammes de séquence/collaboration
  - ➔ Des ensembles d'objets destinataires pour un envoi de message
  - ➔ Des attributs dérivés
  - ➔ Des stéréotypes
  - ➔ ...

BTD/GL/OCL

12

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

Contexte

- Une expression OCL est toujours définie dans un contexte
- Ce contexte est une instance d'une classe
- Mot-clé : context
- Exemple :
  - ➔ **context** Compte
  - ➔ L'expression OCL s'applique à la classe Compte, c'est-à-dire à toutes les instances de cette classe

BTD/GL/OCL

13

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

Invariants

- Un invariant exprime une contrainte sur un objet ou un groupe d'objets qui doit être respectée en permanence
- Mot-clé : inv:
- Exemple :
  - ➔ **context** Compte
  - ➔ **inv**: solde > 0
  - ➔ Pour toutes les instances de la classe Compte, l'attribut solde doit toujours être positif

BTD/GL/OCL

14

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

Pré et postconditions

- Pour spécifier une opération :
  - ➔ Précondition : état qui doit être respecté avant l'appel de l'opération
  - ➔ Postcondition : état qui doit être respecté après l'appel
  - ➔ Mots-clés : pre: et post:
- Dans la postcondition, deux éléments particuliers sont utilisables :
  - ➔ Attribut result : référence la valeur retournée par l'opération
  - ➔ mon_attribut@pre : référence la valeur de mon_attribut avant l'appel de l'opération
- Syntaxe pour préciser l'opération :
  - ➔ context ma_classe::mon_op(liste_param) : type_retour

BTD/GL/OCL

15

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

Pré et postconditions

- Exemples :
  - ➔ **context** Compte::débiter(somme : int)
    pre: somme > 0
    post: solde = solde@pre – somme
  - ➔ La somme à débiter doit être positive pour que l'appel de l'opération soit valide
  - ➔ Après l'exécution de l'opération, l'attribut solde ***doit*** avoir pour valeur la différence de sa valeur avant l'appel et de la somme passée en paramètre
  - ➔ **context** Compte::getSolde() : int
    post: result = solde
- Attention : on ne décrit pas comment l'opération est réalisée mais des contraintes sur l'état avant et après son exécution

BTD/GL/OCL

16

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

Accès aux objets, navigation

- Dans une contrainte OCL associée à un objet, on peut :
  - ➔ Accéder à l'état interne de cet objet (ses attributs)
  - ➔ Naviguer dans le diagramme : accéder de manière transitive à tous les objets (et leur état) avec qui il est en relation
- Nommage des éléments :
  - ➔ Attributs ou paramètres d'une opération : utilise leur nom directement
  - ➔ Objet(s) en association : utilise le nom de la classe associée (en minuscule) ou le nom du rôle d'association du coté de cette classe
- Si cardinalité de 1 pour une association : référence un objet
- Si cardinalité > 1 : référence une collection d'objets

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

Accès aux objets, navigation

- Exemples, dans le contexte de la classe Compte :
  - ➔ solde : attribut référencé directement
  - ➔ banque : objet de la classe Banque (référence via le nom de la classe) associé au compte
  - ➔ propriétaire : objet de la classe Personne (référence via le nom de rôle d'association) associée au compte
  - ➔ banque.clients : ensemble des clients de la banque associée au compte (référence par transitivité)
  - ➔ banque.clients.age : ensemble des âges de tous les clients de la banque associée au compte
- Le propriétaire d'un compte doit avoir plus de 18 ans :

  context Compte

  inv: propriétaire.age >= 18

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

Opérations sur objets et ensembles

- OCL propose un ensemble de primitives utilisables sur les ensembles :
  - ➔ size() : retourne le nombre d'éléments de l'ensemble
  - ➔ isEmpty() : retourne vrai si l'ensemble est vide
  - ➔ notEmpty() : retourne vrai si l'ensemble n'est pas vide
  - ➔ includes(obj) : vrai si l'ensemble inclut l'objet obj
  - ➔ excludes(obj) : vrai si l'ensemble n'inclut pas l'objet obj
  - ➔ including(obj) : l'ensemble référencé doit être cet ensemble en incluant l'objet obj
  - ➔ excluding(obj) : idem mais en excluant l'objet obj
  - ➔ includesAll(ens) : l'ensemble contient tous les éléments de l'ensemble ens
  - ➔ excludesAll(ens) : l'ensemble ne contient aucun des éléments de l'ensemble ens
- Syntaxe d'utilisation : objetOuCollection -> primitive

BTD/GL/OCL

19

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

Opérations sur objets et ensembles

- Exemples, invariants dans le contexte de la classe Compte
  - ➔ propriétaire -> notEmpty() : il y a au moins un objet Personne associé à un compte
  - ➔ propriétaire -> size() = 1 : le nombre d'objets Personne associés à un compte est de 1
  - ➔ banque.clients -> size() >= 1 : une banque a au moins un client
  - ➔ banque.clients -> includes(propriétaire) : l'ensemble des clients de la banque associée au compte contient le propriétaire du compte
  - ➔ banque.clients.compte -> includes(self) : le compte appartient à un des clients de sa banque
- self : pseudo-attribut référençant l'objet courant

BTD/GL/OCL

20

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

Opérations sur objets et ensembles

- Autre exemple :

  context Banque :: créerCompte(p : Personne) : Compte

  post: result.oclIsNew() **and**

  compte = compte@pre -> including(result) **and**

  p.compte = p.compte@pre -> including(result)

  ➔ Un nouveau compte est créé. La banque doit gérer ce nouveau compte. Le client passé en paramètre doit posséder ce compte. Le nouveau compte est retourné par l'opération.
- oclIsNew() : primitive indiquant qu'un objet doit être créé pendant l'appel de l'opération (à utiliser dans une postcondition)
- and : permet de définir plusieurs contraintes pour un invariant, une pré ou postcondition
- and = « et logique » : l'invariant, pré ou postcondition est vrai si toutes les expressions reliées par le « and » sont vraies

BTD/GL/OCL

21

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

Relations ensemblistes entre collections

- union : retourne l'union de deux ensembles
- intersection : retourne l'intersection de deux ensembles
- Exemples :

  ➔(ens1 -> intersection(ens2)) -> isEmpty()
  - Les ensembles ens1 et ens2 n'ont pas d'élément en commun

  ➔ens1 = ens2 -> union(ens3)
  - L'ensemble ens1 doit être l'union des éléments de ens2 et de ens3

BTD/GL/OCL

22

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

Opérations sur les éléments d'une collection

- OCL permet de vérifier des contraintes sur chaque élément d'une collection ou de définir une sous-collection à partir d'une collection en fonction de certaines contraintes
- Primitives offrant ces services et s'appliquant sur une collection col :
  - ➔ **select :** retourne le sous-ensemble de la collection col dont les éléments respectent la contrainte spécifiée
  - ➔ **reject :** idem mais ne garde que les éléments ne respectant pas la contrainte
  - ➔ **collect :** retourne une collection (de taille identique) construite à partir des éléments de col. Le type des éléments contenus dans la nouvelle collection peut être différent de celui des éléments de col.
  - ➔ **exists :** retourne vrai si au moins un élément de col respecte la contrainte spécifiée et faux sinon
  - ➔ **forAll :** retourne vrai si tous les éléments de col respectent la contrainte spécifiée (pouvant impliquer à la fois plusieurs éléments de la collection)

BTD/GL/OCL

23

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

Opérations sur les éléments d'une collection

- Syntaxe de ces opérations :
- ensemble -> primitive(expression)
  - ➔ La primitive s'applique aux éléments de l'ensemble et pour chacun d'entre eux, l'expression *expression* est vérifiée. On accède aux attributs/relations d'un élément directement.
- ensemble -> primitive(elt : type | expression)
  - ➔ On fait explicitement apparaître le type des éléments de l'ensemble (ici type). On accède aux attributs/relations de l'élément courant en utilisant elt (c'est la référence sur l'élément courant)
- ensemble -> primitive(elt | expression)
  - ➔ On nomme l'attribut courant (elt) mais sans préciser son type

BTD/GL/OCL

24

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

Opérations sur les éléments d'une collection

- Dans le contexte de la classe Banque :
  - ➔ compte -> select(c | c.solde > 1000)
    - Retourne une collection contenant tous les comptes bancaires dont le solde est supérieur à 1000 €
  - ➔ compte -> reject(solde > 1000)
    - Retourne une collection contenant tous les comptes bancaires dont le solde n'est pas supérieur à 1000 €
  - ➔ compte -> collect(c : Compte | c.solde)
    - Retourne une collection contenant l'ensemble des soldes de tous les comptes
  - ➔ (compte -> select(solde > 1000))
    -> collect(c | c.solde)
    - Retourne une collection contenant tous les soldes des comptes dont le solde est supérieur à 1000 €

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

Opérations sur les éléments d'une collection

- **context** Banque
  inv: not(clients -> exists (age < 18))
  - ➔ Il n'existe pas de clients de la banque dont l'age est inférieur à 18 ans
  - ➔ not : prend la négation d'une expression
- **context** Personne
  inv: Personne.allInstances() -> forAll(p1, p2 | p1 <> p2 **implies** p1.nom <> p2.nom)
  - ➔ Il n'existe pas deux instances de la classe Personne pour lesquelles l'attribut nom a la même valeur : deux personnes différentes ont un nom différent
- allInstances() : primitive s'appliquant sur une classe (et non pas un objet) et retournant toutes les instances de la classe référencée (ici la classe Personne)

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

Types de collection

- 3 types de collection d'objets :
  - ➔ Set : ensemble au sens mathématique, pas de doublons, pas d'ordre
  - ➔ Bag : comme un Set mais avec possibilité de doublons
  - ➔ Sequence : un Bag dont les éléments sont ordonnés
- Exemples :
  - ➔ { 1, 4, 3, 5 } : Set
  - ➔ { 1, 4, 1, 3, 5, 4 } : Bag
  - ➔ { 1, 1, 3, 4, 4, 5 } : Sequence
- Possibilité de transformer un type de collection en un autre type de collection
- Note1 : un collect() renvoie toujours un Bag
- Note2 : en OCL 2.0, possibilité de collections de collections et de tuples

BTD/GL/OCL

27

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

Conditionnelles

- Certaines contraintes sont dépendantes d'autres contraintes. Deux formes pour gérer cela :
  - ➔ **if** expr1 **then** expr2 **else** expr3 **endif** : si l'expression expr1 est vraie alors expr2 doit être vraie sinon expr3 doit être vraie
  - ➔ expr1 **implies** expr2 : si l'expression expr1 est vraie, alors expr2 doit être vraie également. Si expr1 est fausse, alors l'expression complète est vraie

BTD/GL/OCL

28

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

Conditionnelles

- **context** Personne **inv**:
  if age < 18
  then compte -> isEmpty()
  else compte -> notEmpty()
  endif
  ➔ Une personne de moins de 18 ans n'a pas de compte bancaire alors qu'une personne de plus de 18 ans possède au moins un compte

- **context** Personne **inv**:
  compte -> notEmpty() **implies** banque -> notEmpty()
  ➔ Si une personne possède au moins un compte bancaire, alors elle est cliente d'au moins une banque

BTD/GL/OCL

29

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

Commentaires et nommage de contraintes

- Commentaire en OCL : utilisation de --
  ➔ Exemple :

```
context Personne inv:
if age < 18 -- vérifie l'age de la personne
then compte -> isEmpty() -- pas majeur : pas de compte
else compte -> notEmpty() -- majeur : doit avoi
   -- au moins un compte
endif
```

- On peut nommer des contraintes
  ➔ Exemple :
  - **context** Compte
    inv soldePositif: solde > 0
  - **context** Compte::débiter(somme : int)
    pre sommePositive: somme > 0
    post sommeDébitée: solde = solde@pre - somme

BTD/GL/OCL

30

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

Variables

- Pour faciliter l'utilisation de certains attributs ou calculs de valeurs on peut définir des variables
- Dans une contrainte OCL : let ... in ...
  - ➔ **context** Personne
    inv: **let** argent = compte.solde -> sum() **in** age >= 18 **implies** argent > 0
  - ➔ Une personne majeure doit avoir de l'argent
  - ➔ sum() : fait la somme de tous les objets de l'ensemble
- Pour l'utiliser partout : def
  - ➔ **context** Personne
    def: argent : int = compte.solde -> sum()
  - ➔ **context** Personne
    inv: age >= 18 **implies** argent > 0

BTD/GL/OCL

31

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

Appels d'opération des classes

- Dans une contrainte OCL : accès aux attributs, objets ... « en lecture »
- Possibilité d'utiliser une opération d'une classe dans une contrainte :
  - ➔ Si pas d'effets de bords (de type « query »)
  - ➔ Car une contrainte OCL exprime une contrainte sur un état mais ne précise pas qu'une action a été effectuée
- Exemple :
  - ➔ **context** Banque
    inv: compte -> forAll(c | c.getSolde() > 0)
  - ➔ getSolde() est une opération de la classe Compte. Elle calcule une valeur mais sans modifier l'état d'un compte

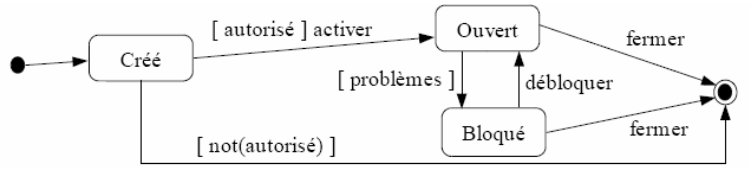
BTD/GL/OCL

32

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

Liens avec diagrammes d'états



- Possibilité de référencer un état d'un diagramme d'états associé à l'objet
- oclInState(etat) : vrai si l'objet est dans l'état etat.
- Pour sous-états : etat1::etat2 si etat2 est un état interne de etat1
- Exemples :

  ➔ **context** Compte :: débiter(somme : int)

  pre: somme > 0 **and** self.oclInState(Ouvert)

  - L'opération débiter ne peut être appelée que si le compte est dans l'état ouvert

BTD/GL/OCL

33

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

Liens avec diagrammes d'états

- On ne peut pas avoir plus de 5 comptes ouverts dans une même banque

  context Compte :: activer()

  pre: self.oclInState(Créé) **and**

  propriétaire.compte -> select(c | self.banque = c.banque) -> size() < 5

  post: self.oclInState(Utilisable)
- On peut aussi exprimer la garde [autorisé] en OCL :

  ➔ **context** Compte

  def: autorisé : Boolean = propriétaire.compte -> select(c | self.banque = c.banque) -> size() < 5

BTD/GL/OCL

34

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

Propriétés

- De manière générale en OCL, une propriété est un élément pouvant être :
  - ➔ Un attribut
  - ➔ Un bout d'association
  - ➔ Une opération ou méthode de type requête
- On accède à la propriété d'un objet avec « . »
- Exemples :
  - ➔ **context** Compte **inv**: self.solde > 0
  - ➔ **context** Compte **inv**: self.getSolde() > 0
- On accède à la propriété d'un ensemble avec « -> »

BTD/GL/OCL 35

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

Accès aux attributs pour les ensembles

- Accès à un attribut sur un ensemble :
  - ➔ Exemple dans contexte de Banque :
    compte.solde
  - ➔ Renvoie l'ensemble des soldes de tous les comptes
- Forme raccourcie et simplifiée de :
  - ➔ compte -> collect (solde)

BTD/GL/OCL 36

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

Règles de précédence

- Pour objets :
  - ➔ oclIsTypeOf(type) : l'objet est du type *type*
  - ➔ oclIsKindOf(type) : l'objet est du type *type* ou un de
  - ➔ ses sous-types
  - ➔ oclInState(état) : l'objet est dans l'état état
  - ➔ oclIsNew() : l'objet est créé pendant l'opération
  - ➔ oclAsType(type) : l'objet est « casté » en type type
- Pour ensembles :
  - ➔ isEmpty(), notEmpty(), size(), sum()
  - ➔ includes(), excludes(), includingAll() ...
  - ➔ ....

BTD/GL/OCL

37

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

Règles de précédence

- Ordre de précédence pour les opérateurs/primitives :
  - ➔ @pre
  - ➔ . et ->
  - ➔ not et -
  - ➔ * et /
  - ➔ + et -
  - ➔ if then else endif
  - ➔ >, <, <= et >=
  - ➔ = et <>
  - ➔ and, or et xor
  - ➔ implies
- Les parenthèses permettent de changer cet ordre

BTD/GL/OCL

38

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

Plan

1. Pourquoi OCL ? Introduction par l'exemple
2. Les principaux concepts d'OCL
3. *Exemple d'application sur un autre modèle*
4. Utilisation en pratique d'OCL lors d'un développement logiciel

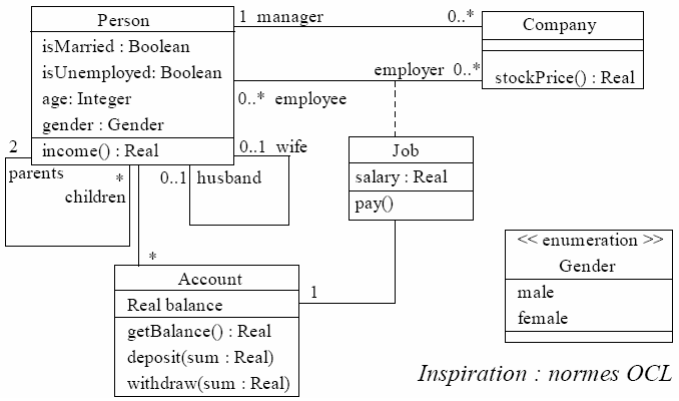
BTD/GL/OCL

39

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

Diagramme de classes



Inspiration : normes OCL

BTD/GL/OCL

40

CENTRALE L Y O N

Bertrand DAVID : Génie Logiciel

Contraintes sur employés d'une compagnie

- Dans une compagnie, un manager doit travailler et avoir plus de 40 ans. Le nombre d'employé d'une compagnie est non nul.

  context Company:

  inv:

  self.manager.isUnemployed = false **and**
  self.manager.age > 40 **and**
  self.employee -> notEmpty()

BTD/GL/OCL

41

CENTRALE L Y O N

Bertrand DAVID : Génie Logiciel

Lien salaire/chômage pour une personne

- Une personne considérée comme au chômage ne doit pas avoir des revenus supérieurs à 100 €

  context Person
  inv:
  let money : Real = self.job.salary->sum() **in**
  If isUnemployed **then**
  money < 100
  else
  money >= 100
  endif

BTD/GL/OCL

42

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

Contraintes sur les parents/enfants

- Un enfant a un père et une mère

```
context Person
def: parent1 = parents -> at(0)
def: parent2 = parents -> at(1)

context Person
inv:
if parent1.gender = #male
then -- parent1 est un homme
    parent2.gender = #female
else -- parent1 est une femme
    parent2.gender = #male
endif
```

BTD/GL/OCL

43

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

Contraintes sur les parents/enfants

- Tous les enfants d'une personne ont bien cette personne comme parent et inversement

```
context Person
inv:
children -> notEmpty() implies
children -> forAll ( p : Person |
    p.parents -> includes(self))

context Person
inv:
parents -> forAll ( p : Person |
    p.children -> includes (self))
```

BTD/GL/OCL

44

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

Contraintes de mariage

- Pour être marié, il faut avoir plus de 18 ans. Un homme est marié avec une femme et une femme avec un homme.

```
context Person inv:
(self.isMarried implies self.age >= 18 and
    self.wife -> union(self.husband) -> size()=1) and
(self.wife -> notEmpty() implies
    self.wife.gender = #female and
    self.gender = #male and
    self.wife.age >= 18 and
    self.wife.isMarried = true and
    self.wife.husband = self)
and (self.husband -> notEmpty() implies
    self.husband.gender = #male and
    self.gender = #female and
    self.husband.age >= 18 and
    self.husband.isMarried = true and
    self.husband.wife = self)
```

BTD/GL/OCL

45

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

Embauche d'un nouvel employé

- Un employé qui est embauché n'appartenait pas déjà à la compagnie

```
context Company::hireEmployee(p : Person)
post:
    employee = employee@pre -> including(p)
    employee@pre -> excludes(p) and
    stockPrice() = stockPrice()@pre + 10
```

- Equivalent à :

```
context Company::hireEmployee(p : Person)
pre: employee -> excludes(p)
post:
    employee -> includes(p) and
    stockPrice() = stockPrice()@pre + 10
```

BTD/GL/OCL

46

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

Revenus selon l'age

- Selon l'age de la personne, ses revenus sont :
  - ➔ 1% des revenus des parents quand elle est mineure (argent de poche)
  - ➔ Ses salaires quand elle est majeure

context Person::income() : Real
post:
if age < 18 **then**
    result = (parents.job.salary -> sum()) * 1%
else
    result = self.job.salary -> sum()
endif

BTD/GL/OCL

47

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

Versement salaire

- Salaire payé :
  context Job::pay()
  post:
  account.balance = account.balance@pre + salary

- En OCL 2.0 : peut aussi préciser que l'opération deposit doit être appelée :
  - ➔ **context** Job::pay()
    post: account^deposit(salary)
  - ➔ objet^operation(param1, ...) : renvoie vrai si un message *operation* est envoyé à objet avec la liste de paramètres précisée (si pas de valeur particulière : utilise « ? : type »)

- Note : s'éloigne des principes d'OCL (langage de contraintes et pas d'actions) et généralement exprimable en UML avec diagrammes d'interactions (séquence, collaboration)

BTD/GL/OCL

48

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

Plan

1. Pourquoi OCL ? Introduction par l'exemple
2. Les principaux concepts d'OCL
3. Exemple d'application sur un autre modèle
4. *Utilisation en pratique d'OCL lors d'un développement logiciel*

BTD/GL/OCL

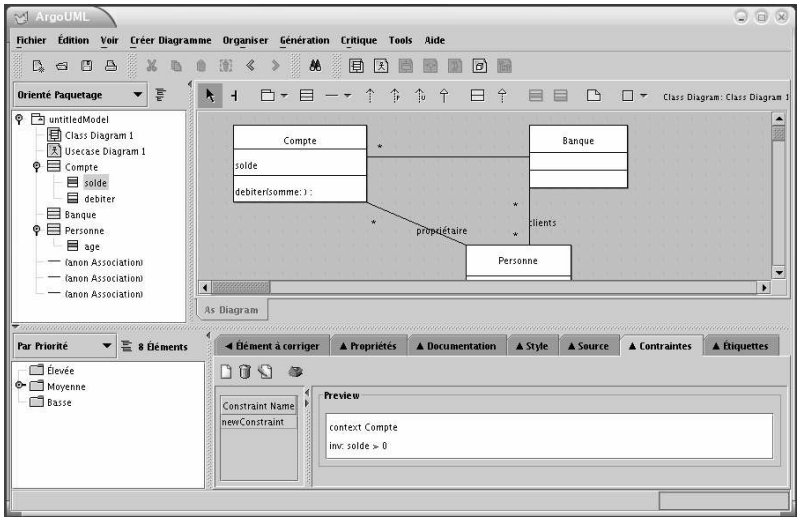
49

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

Utilisation en pratique d'OCL

Outil ArgoUML : spécification diagramme de classe et contraintes OCL



BTD/GL/OCL

50

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

Code généré pour la classe Compte

```
public class Compte {
/**
*
* @invariant newConstraint_0: solde > 0
*/
public int solde;
/* {transient=false, volatile=false}*/
public Personne propriétaire;
public Banque myBanque;
public void debiter(int somme) {
}
}
```

BTD/GL/OCL

51

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

Implémentation utilisant les contraintes OCL

- On termine l'implémentation de la classe Compte
- On utilise un outil pour transformer le code et gérer les contraintes OCL dans le code
- A l'exécution, si une contrainte n'est pas respectée, une exception est levée (fonctionnement à la Eiffel ou JML)
- Pour plus d'infos : http://dresden-ocl.sourceforge.net/
- Cycle de vie de l'utilisation d'OCL lors d'un développement :
  - ➔ Spécification des contraintes sur les diagrammes UML
  - ➔ Génération de squelettes de code
  - ➔ Implémentation des classes
  - ➔ Transformation du code pour intégrer les contraintes OCL
  - ➔ Vérification des contraintes à l'exécution

BTD/GL/OCL

52

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

Conclusion

- Avantages d'OCL :
  - ➔ Langage formel à la syntaxe simple
  - ➔ Bien adapté à une utilisation dans un contexte objet (UML)
  - ➔ Permet de spécifier clairement des contraintes sur un
  - ➔ ensemble de diagrammes UML
  - ➔ Permet de réaliser des spécifications complètes et non
  - ➔ ambigües
  - ➔ Normalisé par l'OMG
- Inconvénients :
  - ➔ Ecriture pouvant tout de même s'avérer complexe dans certains cas
  - ➔ Peu d'outils permettant de manipuler des contraintes OCL

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

Bibliographie

- *The Object Constraint Language: Getting Your Models Ready for MDA*, Second Edition, Jos Warmer et Anneke Kleppe, Addison-Wesley, 2003
- Soumission OCL 2.0 à l'OMG
  http://www.omg.org/cgi-bin/doc?ad/2003-01-07