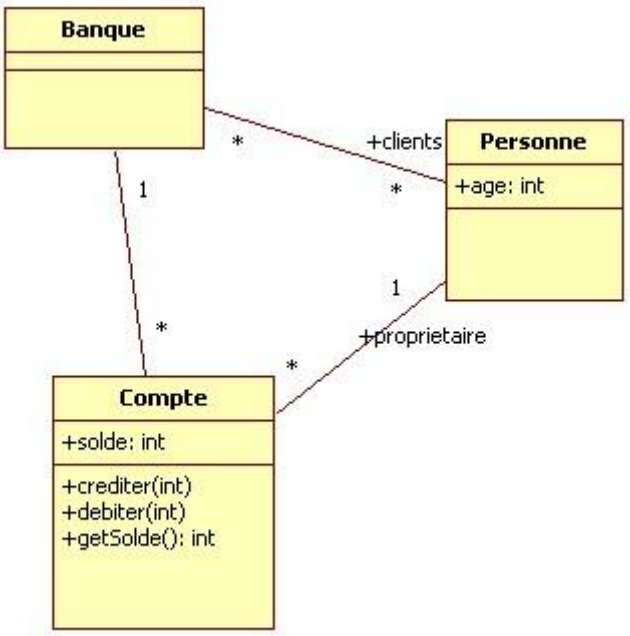
OCL

Guillaume PANNETIER

1 Exemple utilisé



2 Eléments d'une expression OCL :

[Contexte] (Mot-clef **context**)

[Invariants] (Mot-clef **inv**)

Fixe des conditions qui doivent toujours être respectées.

[Pré-conditions] (Mot-clef **pre**)

Fixe des conditions sur l'état initial du contexte.

Seulement pour les méthodes

[Post-conditions] (Mot-clef **post**)

Fixe des conditions sur l'état final du contexte.

monAttribut@pre correspond à la valeur lors de la pré-condition.

Result référence la valeur retournée par l'opération.

Seulement pour les méthodes

[Résultat d'une opération] (Mot-clef **body**)

Spécifie le résultat d'une opération.

Seulement pour les méthodes

[Valeur initial ou dérivée] (Mot-clef **init** ou **derive**)

Init : Spécifie la valeur initiale d'un attribut ou d'une association

Deriv : Spécifie la règle de dérivation d'un attribut ou d'une association

Seulement pour les attributs ou les associations

[Variables globale] (Mot-clef **def**)

Variable accessible partout. (sinon voir **let…in** pour une variable locale)

Exemple : def : argent : int = compte.solde->sum()

2.1 Le contexte

Défini l'endroit où appliquer les contraintes, cela peut-être :

- Une classe (ex : **context Compte**)
- Une méthode (ex : **context Compte :: debiter(somme : Integer)**)
    - Dans ce cas on peut utiliser les pré-conditions et les post-conditions ;
    - Le mot-clef **result** référence la valeur retournée par l'opération.

3 Navigation dans le modèle

3.1 Navigation dans l'objet

- Accès à l'état interne de l'objet (ses attributs) ;
- On utilise pour cela le nom de l'attribut.

  ex :

  context : Compte

  inv : solde > 0

3.2 Navigation dans le diagramme

- On peut naviguer entre les classes en utilisant les associations qui existent entre-elles. Pour cela, deux manières existent :
    - Utiliser le nom de l'association (s'il y en a un) ;
    - Utiliser le nom de la classe cible **en minuscule**.

      ex :

      context : Compte

      inv : proprietaire.age >= 18

      inv : personne.age >= 18

4 Types

4.1 Types de base

- **Integer** :
    - Operations associées : *, +, -, /, abs()
- **Real** :
    - Opérations associées : *, +, -, /, floor()
- **String** :
    - Opérations associées : concat(), size(), substring()
- **Boolean** :
    - Opérations associées : and, or, xor, not, implies, if-then-else

4.2 Types de collections d'objets

- **Set** : ensemble au sens mathématiques, pas de doublons, pas d'ordre ;
    - Exemple : {1, 4, 3, 5}
- **OrderedSet** : Idem mais avec ordre ;
- **Bag** : comme Set mais avec possibilité de doublons ;

- Exemple : {1, 4,1, 3, 5, 4}

- **Sequence** : Bag dont les éléments sont ordonnés ;
  - Exemple : {1, 1, 3, 4, 4, 5}

  → Possibilité de transformer un type de collection en un autre type de collection avec opérations OCL dédiées.

- **Collections de collections** ;
  - Manipulations comme telles ;
  - Manipulation comme une unique collection, on applatit le contenu → Opération **flatten()** ;
- **Tuples /n-uplet** : Données contenant plusieurs champs.
  - Exemple : tuple {nom : String = 'toto', age : Integer = 21}

5 Conformité de types

- Opérations sur les types :
  - **oclIsTypeOf(type) :** Vrai si l'objet **est du type *type*** ;
  - **oclIsKindOf(type) :** Vrai si l'objet **est du type *type* ou un de ses sous-types** ;
  - **oclAsType(type) : Cast** de l'objet en type *type*.
- Types internes à OCL :
  - Collection est le super-type de Set, Bag et Sequence ;
  - Conformité entre collection et types des objets contenus : Set(T1) est conforme à Collection(T2) si T1 est sous-type de T2 ;
  - Integer est un sous-type de Real.

6 Opérations sur objets et collections

- **size()** : retourne le nombre d'éléments de la collection ;
- **isEmpty()** : retourne vrai si la collection est vide ;
- **notEmpty()** : retourne vrai si la collection n'est pas vide ;
- **includes(obj)** : retourne vrai si la collection inclut l'objet *obj* ;
- **excludes(obj)** : retourne vrai si la collection n'inclut pas l'objet *obj* ;
- **including(obj)** : la collection référencée doit être cette collection en incluant l'objet *obj* ;
- **excluding(obj)** : idem en excluant l'objet *obj* ;
- **includesAll(ens)** : la collection contient tous les éléments de la collection *ens* ;
- **excludesAll(ens)** : la collection ne contient aucun des éléments de la collection *ens ;*
- **Self** : pseudo-attribut référençant l'objet courant.
- **oclIsNew()** : primitive indiquant qu'un objet doit être créé pendant l'appel de l'opération (à utiliser dans une post-condition) ;
- **and** :
  - permet de définir plusieurs contraintes pour un invariant, une pré ou post-condition ;
  - équivalent au « et logique » : vrai si toutes les expressions reliées sont vraies.
  - Exemple :

    context Banque ::creerCompte(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.

- **Union** : retourne l'union entre 2 collections (Ex : (col1->union(col2))->notEmpty()) ;
- **Intersection** : retourne l'intersection de 2 collections.

7 Opérations sur les éléments d'une collection

7.1 Usages

- **col->primitive(expr)** : Primitive appliquée aux éléments de la collection. Pour chacun d'eux, l'expression est vérifiée. On accède aux attributs/relations d'un élément directement.
- **col->primitive(elem : type|expr)** : Le type des éléments de la collection est explicité. On accède aux attributs/relations de l'élément courant en utilisant *elem*.
- **col->primitive(elem|expr)** : l'attribut courant *elem* est nommé mais sans préciser son type.

7.2 Liste des opérations

- **select** et **reject** : retourne le **sous-ensemble** de la collection dont les éléments (ne) respectent (pas) la contrainte spécifiée.
  - **Syntaxe** : col->select(expr)
    col->select(elem : type|expr)
    col->select(elem|expr)
  - **Exemple (context Banque)** : compte->select(c | c.solde>1000)
    compte->reject(solde>1000)

- **collect** :Retourne une **collection** (de taille identique) construite à partir des éléments de la collection initiale. Le type des éléments contenus dans la nouvelle collection peut être différent de celui de la collection initiale.
  - **Remarque** : *collect* renvoie toujours un **bag**.
  - **Syntaxe** : col->collect(expr)
    col->collect(elem : type | expr)
    col->(elem|expr)
  - **Exemple (context Banque)** : compte->collect(c : Compte | c.solde) : retourne une collection contenant l'ensemble des soldes de tous les comptes.

- **exists** : Retourne **vrai** si **au moins** un élément de la collection respecte la contrainte spécifiée.
  - **Exemple (context Banque)** : **inv : not**(clients->exists(age<18))

- **forAll** : Retourne **vrai si tous les élément de la collection** respectent la contrainte spécifiée.

- **Exemple (context Personne)** : Personne.**allInstances**()->**forAll**(p1, p2 | p1<>p2 implies p1.nom <> p2.nom)
- **allInstances()** est une primitive qui **s'applique à une classe** (pas à un objet) et retourne toutes les instances de la classe référencée.

- **iterate :** Forme générale d'une itération sur une collection et permet de redéfinir les opérateurs vus précédemment.
  - **Syntaxe :** col->iterate(elem : type ; reponse : type = <valeur> | <expression_avec_elem_et_reponse>)
  - **Exemple :**
    - **context** Banque
      def : moyenneDesComptes : Real = copte->collect(c : Compte | c.solde)->sum() / compte->size()
      -- est identique à
      context Compte
      def : moyenneDesComptes : Real = compte->iterate(c : Compte ; lesComptes : Bag{Integer} | lesComptes->including(c.solde))->sum() / compte->size()

8 Les conditionnelles

- **if** expr1 **then** expr2 **else** expr3 **endif**
- expr1 **implies** expr2

9 Commentaires et nommage de contraintes

- Commentaire en OCL : utilisation de –
- On peut également nommer des contraintes
  - **Exemple :**
    context Compte ::debiter(somme : **Integer**)
    pre : sommePositive : somme > 0
    post : sommeDebitee : solde = solde@pre-somme

10 Appels d'opération des classes

- En plus d'accéder aux attributs, on peut aussi utiliser une opération d'une classe dans une contrainte → **Attention aux effets de bords.**
  - **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.**

11 Syntaxes

Nom	Syntaxe	Description
Contexte	• context nomClasse • context nomClasse ::nomMéthode (listeParam) : typeRetour • context nomClasse ::nomAttribut : typeAttr	Indique dans quel objet (classe, méthode ou attribut) les contraintes sont appliquées.
Invariant	inv : expression	Contrainte qui doit toujours être vérifiée.
Pré-condition	pre : expression	Contrainte qui doit être vérifiée avant l'appel de **l'opération**.
Post-condition	post : expression	Contrainte qui doit être vérifiée au retour de l'appel de **l'opération**. **monAttr@pre** : Référence la valeur de l'attribut avant l'appel. **result** : référence la valeur du résultat de l'opération.
Accès à un attribut	objet.nomAttr exemple : **context** Compte **inv :** self.solde > 0	Permet d'accéder à un attribut du contexte ou d'un objet référencé.
Accès à une classe associée	Nom du rôle (s'il existe) nom de la classe (**minuscule**) exemple : **context** Compte **inv :** proprietaire.age >= 18	Permet d'accéder à une autre classe du diagramme en navigant dans les associations.
Résultat d'une opération	**body** : expression	Spécifie le résultat d'une opération
Valeur initiale	**init :** expression	Valeur initiale d'un attribut ou d'une association
Valeur dérivée	**derive :** expression Exemple : **context** Personne ::argentPoche : Integer **init** : parents.salaire->sum() * 1% **derive** : **if** mineur **then** parents.salaire->sum()*1% **else** job.salaire **endif**	Valeur dérivée spécifie la règle de dérivation d'un attribut ou d'une association.
Variables	• **let** nomVar = valeur **in** expression • **def :** nomVar : typevar = valeur	Peut faciliter l'utilisation de certains attributs ou calculs de valeurs.
Appel à une primitive	Objet->**primitive()**	

Opération sur les éléments d'une collection	col->primitive(expr) col->primitive(elem : type|expr) col->primitive(element|expr)	Le « | » se lit « tel que ».
Appels d'opération des classes	objet.nomMethode()	**Attention** : Cette-fois-ci on utilise le point « . » et non pas la flèche « -> ».