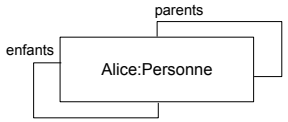
IFT2255 - Génie logiciel

Object Constraint Language (OCL)

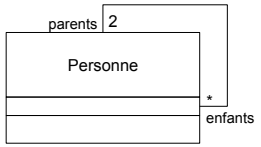
Bruno Dufour
dufour@iro.umontreal.ca

Rappel

- Les diagrammes UML ne nous disent pas tout...



n'est pas une instance valide de



Object Constraint Language (OCL)

- Développé par IBM en 1995
- Fait partie d'UML depuis la version 1.1
- Permet d'exprimer de l'information supplémentaire à propos d'un modèle (un ensemble de diagrammes UML)
- Est un langage de spécification formelle
  - purement déclaratif (pas d'effets de bord)
  - fortement typé

Contextes

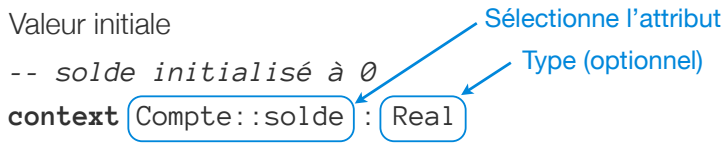
- OCL permet d'associer des contraintes à des **contextes**
- N'importe quel élément du modèle peut servir de contexte
  - paquetage, composant, classe, interface, opération, attribut
- Le contexte est exprimé à l'aide du mot clé **`context`**

Contraintes

- Contrainte: restriction d'une ou plusieurs valeurs dans un modèle UML
  - Formulées en fonction d'un diagramme de classes, appliquées au niveau des objets
- Similaire à des assertions pour les modèles
  - Assertion : prédicat inséré dans un programme afin d'indiquer ce que le développeur croit être toujours vrai à un endroit donné

Contraintes d'attributs

- Valeur initiale

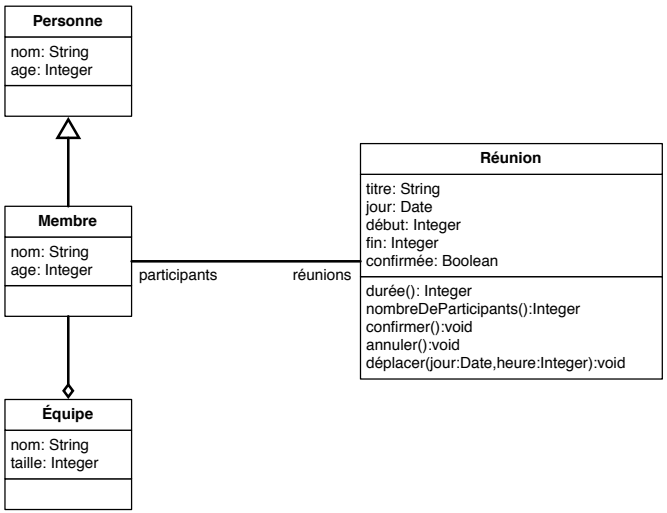



```
-- solde initialisé à 0
context Compte::solde : Real
init: 0
```

- Valeur dérivée

```
-- Intérêt dérivé
context Compte::intérêt : Real
derive: solde * 0.005
```


Exemple



Types de contraintes

- Invariants
- Préconditions
- Postconditions

Invariants

- Une contrainte qui est vraie durant toute la durée de vie d'un objet
- Syntaxe:

  context <classe> **inv** [<nom>]: <expression OCL booléenne>
- ex:
  - **context** Réunion **inv**:

    self.fin > **self**.début

    self : instance pour laquelle la contrainte est évaluée (similaire à **this** en Java)

Précondition

- Une contrainte qui doit être vraie juste avant l'exécution d'une opération
- Syntaxe:

  context <classe>::<opération> (<paramètres>)

  pre [<nom>]: <expression OCL booléenne>
- ex:

  context Réunion::déplacer(jour:Date,heure:Integer)

  pre: heure>0

  pre: **self**.confirmée = **false**

Postcondition

- Une contrainte qui doit être vraie juste après l'exécution d'une opération
- Syntaxe :

  context <classe>::<opération> (<paramètres>)

  post [<nom>]: <expression OCL booléenne>

Postcondition - exemples


```
context Meeting::confirm()
post: self.isConfirmed = true
```

```
context Meeting::duration():Integer
post: result = self.end – self.start
```

fait référence au résultat de l'opération

```
context Compte::dépôt(montant:Real)
post: solde = solde@pre + montant
```

indique une expression qui doit être évaluée dans l'état original (avant l'exécution de l'opération)

Types

- Types primitifs d'UML : String, Real, Integer, etc.
- Collections d'OCL : Set, Bag, Sequence, OrderedSet
- Les types du modèle UML (diagramme de classes)
- OclAny : l'ancêtre de tous les types

Opérations - Types primitifs

- Integer, Real
  - `=, <>, <=, >=, +, -, *, /, x.mod(y), x.div(y)`
- Boolean
  - `and, or, not, xor, =, <>, implies, if <expr> then ... else ... endif`
- String
  - `s.concat(t), s.size(), s.toLower(), s.toUpper(), s.subString(start, end)`
  - L'indexation débute à 1 et non 0

Opérations - Tous les types

- `allInstances()`: retourne l'ensemble de toutes les instances d'une classe
  - `Personne.allInstances()`
- `oclType()`: retourne le type d'un objet
- `oclIsTypeOf(<type>)`: retourne vrai si l'objet est du type spécifié
- `oclIsKindOf(<type>)`: retourne vrai si l'objet est du type spécifié ou d'un de ses sous-types

Opérations - Collection

- `size() : Integer`
- `includes(object : T) : Boolean`
- `excludes(object : T) : Boolean`
- `count(object : T) : Integer`
- `includesAll(c2 : Collection(T)) : Boolean`
- `excludesAll(c2 : Collection(T)) : Boolean`
- `isEmpty() : Boolean`
- `notEmpty() : Boolean`
- `sum() : T`
- `asBag(), asSet(), asOrderedSet(), asSequence()`

Opérations - Set

- `union(s : Set(T)) : Set(T)`
- `intersection(s : Set(T)) : Set(T)`
- `including(object : T) : Set(T)`
  - Nouvelle collection qui inclut tous les objets de la collection d'origine, en plus de l'objet spécifié
- `excluding(object : T) : Set(T)`
  - Nouvelle collection qui inclut tous les objets de la collection d'origine, sauf l'objet spécifié

Opérations - Collections ordonnées

- `append (object: T) : OrderedSet(T)`
- `prepend(object : T) : OrderedSet(T)`
- `insertAt(index : Integer, object : T)`
- `at(i : Integer) : T`
- `indexOf(obj : T) : Integer`
- `first() : T`
- `last() : T`

Types du modèle

- Chaque classe correspond à un type en OCL
- La généralisation définit la hiérarchie de types
- Pour chaque classe du diagramme, on peut accéder à
  - ses attributs (ex: `obj.start`)
  - ses opérations (ex: `obj.duration()`)

Navigation

- À partir d'un objet spécifique, il est possible de naviguer vers d'autres objets en passant par les associations
  - La navigation utilise les étiquettes (noms) d'associations
- La cardinalité de l'association détermine le type de l'expression:
  - ?..1: le type est un objet (ou un ensemble vide s'il n'y a pas d'objet)
  - sinon: le type est une collection

Itération

- OCL définit un itérateur de base pour toutes les collections

```
context Collection(T)::sum() : T
post: result = self->iterate(
    element : T;
    total : T = 0 | total + element
)
```

- D'autres itérateurs plus conviviaux sont disponibles

Itération

- **exists** : vérifie si une expression est vraie pour au moins un élément d'une collection
- **forAll** : vérifie si une expression est vraie pour tous les éléments d'une collection
- **select/reject** : retournent le sous-ensemble de la collection qui satisfait/ne satisfait pas une condition
- **one** : retourne vrai si un seul élement satisfait une condition
- **any** : retourne un élément qui satisfait une condition, ou null si aucun élément ne satisfait la condition

Exemple - exists


```
-- le conseiller d'un client gère au moins un des
-- comptes de ce client
context Person
inv : self.account->exists(c : Account |
      c.administrator->includes(self.counsellor))
```


Exemples - forAll


```
-- un gestionnaire ne gère pas ses propres comptes
context Person
inv : self.managedAccount->forAll(c : Account
       | c.owner <> self)
inv : self.managedAccount->forAll(owner <> self)

-- double itération
context Person
inv : self.contract->forAll(c1,c2
       | c1 <> c2 implies c1.id <> c2.id)
```

Exemples - select / reject


```
context Company
inv: self.employee->select(p
     | p.age > 50)->notEmpty()
inv: self.employee->select(p : Person
     | p.age > 50)->notEmpty()
inv: self.employee->select(age > 50)->notEmpty()
inv: self.employee->reject(isMarried)->isEmpty()
```


collect

- Retourne une collection contenant le résultat d'une expression pour chacun des éléments d'une collection
  - Bag et non Set, puisque certains éléments pourraient présents plus d'une fois
- ex:

```
-- Une collection des âges de tous les employés
self.employees->collect( age )
```

- Notation abrégée: appliquer une propriété à une collection d'objets équivaut à effectuer une opération `collect` sur la collection avec cette propriété

```
self.employees.age
```


closure

- Permet d'accumuler la fermeture transitive d'une expression
  - Récemment ajouté à OCL
  - Pourrait être exprimé par récursion, mais plus compliqué
- ex:

```
parent.closure(children)
```

retourne une collection qui inclut les enfants, les petits enfants, etc.

Variables

- **`let`**: permet de définir une sous-expression utilisée plus d'une fois dans une même contrainte :

```
context Person
inv: let income = self.job.salary->sum() in
if isUnemployed then
    income < 100
else
    income >= 100
endif
```

Variables

- **def**: Permet de définir une sous-expression sur plusieurs contraintes :

```
context Person
def: income : Integer = self.job.salary->sum()
inv: if isUnemployed then
    income < 100
else
    income >= 100
endif
inv: taxes = income * 0.25
```


Définition de requêtes

- OCL peut être utilisé pour définir précisément le résultat d'une opération à l'aide du mot clé **body**
  - Opération sans effet de bord
  - Une sorte de post-condition exacte

Exemple - body


```
  context Compte::getSolde() : Integer
body : self.solde


context Person::getCurrentSpouse() : Person
pre: self.isMarried = true
body: self.marriages->select(m |
      m.ended = false ).spouse


context Person::isSenior() : Boolean
pre: age >= 0
post: age = age@pre -- age is unchanged by operation
body: age >= 65
```


Contraintes et héritage

- Les contraintes sont héritées
  - Un invariant peut être renforcé mais non affaibli dans une sous-classe
  - Une précondition peut être affaiblie mais non renforcée dans une redéfinition de l'opération dans une sous-classe
  - Une postcondition peut être renforcée mais non affaiblie dans une redéfinition de l'opération dans une sous-classe
- Raison: "Substitution Principle" (Liskov)
  - "Wherever an instance of a class is expected, one can always substitute an instance of any of its subclasses."