labsticc.univ-brest.fr/pages_perso/babau/

Expression de contraintes OCL

(intégration dans Eclipse Modeling Tools)

Jean-Philippe Babau

Département Informatique, UFR Sciences, UBO
Laboratoire Lab-STICC

Modèles et méta-modèles

- Introduction aux modèles
  - Ca sert à quoi de faire des modèles ?
- Outils de modélisation et de méta-modélisation
  - Editeurs de méta-modèles et de modèles (EMF)
  - Vérification de modèles (OCL)
    - Pourquoi des contraintes
    - Object Constraint Language
    - Intégration dans EMF
  - Editeurs de modèles graphiques et textuels (Sirius, XText)
  - Transformation de modèles M2T (Acceleo) et M2M (ATL, Modif)

UBO

Contraintes avec le diagramme de classe

- Contraintes sur UML2 ou Ecore
  - Liste des concepts manipulables
  - Structuration des concepts
  - Règles de représentation : le langage
  - Interprétation des concepts : le code
- Contraintes sur les diagrammes de classe
  - Typage des attributs : liste de valeurs possibles
  - Structure de graphe des objets via les références
    - *Containment* dans Ecore
  - Cardinalité des attributs et références
    - Nombre minimal et maximal de références et d'attributs
    - *Lower Bound* et *UpperBound* dans Ecore

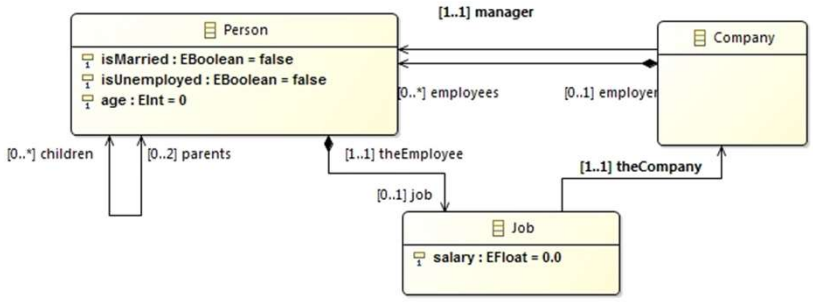
UBO

Contraintes avec le diagramme de classe

- Pas de contraintes sur
  - Les propriétés concernant plusieurs attributs/références d'une classe
  - Les propriétés concernant les attributs/références de plusieurs classes
  - Les paramètres et l'état des objets lors des opérations

UBO

Exemple de diagramme de classe Ecore



Inspiré de la norme OCL

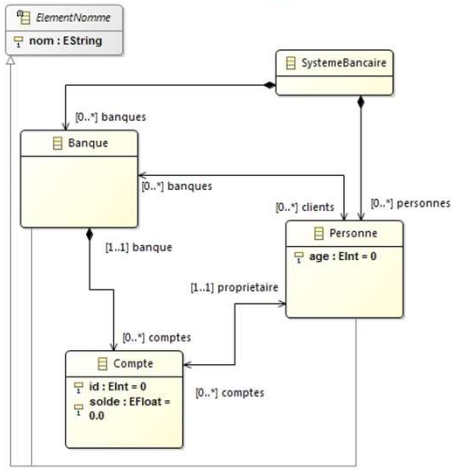
UBO

Exemples de contraintes

- Compagnie
  - Dans une compagnie, un manager doit travailler et avoir plus de 40 ans
- Personne
  - Une personne qui travaille doit avoir des revenus supérieurs à 1000 €
  - on ne peut pas être enfant de soi même
  - *Les deux parents d'un enfant ne sont pas identiques (Ecore : référence Unique)*
  - *Tous les enfants d'une personne ont bien cette personne comme parent et inversement (Ecore : référence EOpposite)*
- Expression ?

UBO

Exemple : diagramme de classe et contraintes

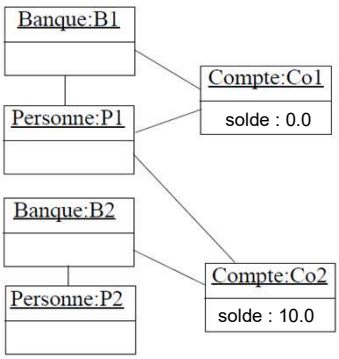


Contraintes

- Le solde est toujours positif
- Une personne possède un compte dans une banque dont elle est cliente

UBO

Exemples d'instanciations …



- (solde>0) : non contrôlé par les outils
- 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

UBO

Expression de contraintes

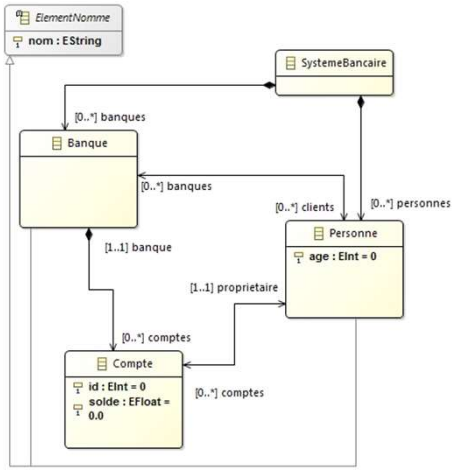
- Commentaires du diagramme de classe : nécessaire mais pas suffisant
  - Sémantique non formalisée
  - Interprétation non automatique
  - Interprétation non assurée
- Expression à l'aide de diagrammes de classe complexes
  - Un cas particulier = une classe
  - Énumération de cas …
- « Bon » nommage (par exemple soldePositif) : nécessaire mais pas suffisant
  - Ambigüe et incomplet
  - Interprétation non fiable
- Absence de contraintes exprimées
  - Modélisation incomplète
  - Peut entrainer des erreurs de modélisation

UBO

Expression de contraintes

- Utiliser une technique de modélisation formelle
  - Non ambigüe
  - Outils automatiques de preuve
- OCL : Object Constraint Language
  - Langage formel
  - Langage de contraintes orienté-objet
  - Standardisé par l'OMG
    - Liens avec UML et le MOF
  - Outillé

Exemples




```
class Compte {

invariant soldePositif  : solde > 0 ;

invariant  comptePourClientBanque :
banque.clients -> includes (proprietaire) ;

}
```


Principes

- Une expression OCL est toujours définie dans un contexte
  - Ce contexte est l'instance d'une classe
- L'expression OCL s'applique à la classe définie par le contexte,
  - C'est-à-dire à toutes les instances de cette classe
- Mot-clé *context*
  - Exemple *context Compte*
- Toujours associer un commentaire en langage naturel à une contrainte OCL
  - OCL interprétable par une machine mais parfois peu lisible pour un humain

Invariant

- Définition
  - Un invariant exprime une contrainte, sur une instance ou un groupe d'instances, qui doit être toujours vrai
  - Expression booléenne
- Mot-clé *invariant*
- Exemples

```
class Compte { …

invariant soldePositif  : solde > 0 ;

invariant comptePourClientBanque : banque.clients -> includes (proprietaire) ;
}
```


Navigation dans le modèle

- Dans une contrainte OCL associée à une classe
  - On peut accéder à l'état interne de l'instance
    - Attributs
    - Type de l'objet
  - Naviguer dans le diagramme
    - Accéder de manière transitive à tous les éléments (et donc à leur état) référencés par la classe de contexte
    - Utilisation des références
- Nommage des éléments dans la contrainte
  - Attribut ou référence de la classe : le nom
    - Impact de la cardinalité
      - Si inférieure 1 : référence null ou un objet
      - Si cardinalité > 1 : référence une collection d'objets
  - Instance référencée par une association
    - nom de la classe associée (en minuscule) ou le nom du rôle d'association du coté de cette classe

Exemples de navigation

- Exemples, dans 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

- Exemple : le propriétaire d'un compte doit avoir plus de 18 ans

```
class Compte { …
invariant proprietaireMajeur : proprietaire.age >= 18 ;
}
```


Types de base et opérations

- Integer
  - 1, -2, 145
  - *, +, -, /, abs()
- Real
  - 1.5, -123.4
  - *, +, -, /
- String
  - 'bonjour'
  - concat(), size(), substring()
- Boolean
  - true, false
  - and, or, not, xor, not, implies, if-then-else

- La plupart des expressions OCL sont de types Boolean
  - Notamment les expressions formant les invariants, les pre et les post

Collections

- 4 types de collection d'objets
  - Set : ensemble au sens mathématique, pas de doublons, pas d'ordre
    - OrderedSet : idem mais ensemble ordonné
  - Bag : comme un Set mais avec possibilité de doublons
    - Sequence : un Bag dont les éléments sont ordonnés
- Passage d'un objet à une collection
  - Utile pour utiliser des opérations sur des collections pour un attribut ou une référence de cardinalité inférieure à 1

    refElement ->asSet()
- Exemples
  - { 1, 4, 3, 5 } : Set
  - { 1, 3, 4, 5 } : OrderedSet
  - { 1, 4, 1, 3, 5, 4 } : Bag
  - { 1, 1, 3, 4, 4, 5 } : Sequence

Opérations sur les collections

- OCL propose un ensemble de primitives utilisables sur les collections

```
col->isEmpty()        -- retourne true si la collection col est vide
col-> notEmpty()      --  retourne true si la collection col n'est pas vide

col-> size()          -- retourne le nombre d'éléments de la collection col

col-> includes(obj)   --  retourne true si la collection col inclut l'objet obj
col-> excludes(obj)   -- retourne true si la collection col n'inclut pas l'objet obj

col-> including(obj)  -- la collection col doit être cette collection en incluant l'objet obj
col-> excluding(obj)  -- idem mais en excluant l'objet obj

col-> includesAll(ens)    -- la collection col contient tous les éléments de la collection ens
col-> excludesAll(ens)    -- la collection col ne contient aucun des éléments de la collection ens
```

- Syntaxe d'utilisation : *Collection -> primitive*

Exemples

- Exemples d'invariants dans le contexte de la classe Compte

```
banque.clients -> size() >= 1
-- une banque a au moins un client (inutile car vérifié par la cardinalité)

banque.clients -> includes(proprietaire)
-- l'ensemble des clients de la banque associée au compte contient le propriétaire du
   compte
```

- self
  - pseudo-attribut référençant l'objet courant

Opérations ensemblistes

- *union*
  - Retourne l'union de deux collections
- *intersection*
  - Retourne l'intersection de deux collections
- *allInstances*
  - Primitive s'appliquant sur une classe (et non pas un objet) et retournant toutes les instances de la classe référencée

- Exemples

```
(col1 -> intersection(col2)) -> isEmpty()
-- renvoie true si les collections col1 et col2 n'ont pas d'élément en commun

col1 = col2 -> union(col3)
-- la collection col1 doit être l'union des éléments de col2 et de col3

Personne.allInstances()
-- l'ensemble des instances de la classe Personne
```

UBO

Opérations de sélection

- Définition d'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 (contrainte) -- retourne le sous-ensemble de la collection *col dont les éléments* respectent la contrainte spécifiée

reject (contrainte) -- idem mais ne garde que les éléments ne respectant pas la contrainte

collect (contrainte) -- 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 (contrainte) -- retourne true si au moins un élément de *col respecte la* contrainte spécifiée, et false sinon

forAll (contrainte) -- retourne true si tous les éléments de *col respectent la contrainte* spécifiée (pouvant impliquer à la fois plusieurs éléments de la collection)

UBO

Opérations de sélection

- 3 syntaxes possibles

collection -> primitive(expression)

– La primitive s'applique aux éléments de la collection et pour chacun d'entre eux, l'expression *expression est* vérifiée. On accède aux attributs/relations d'un élément

collection -> primitive(elt : type | expression)

– On fait explicitement apparaître le type des éléments de la collection (ici *type). On accède aux attributs/relations de* l'élément courant en utilisant *elt (c'est la référence sur* l'élément courant)

collection -> primitive(elt | expression)

– On nomme l'attribut courant (*elt) mais sans préciser son* type

Exemples

```
compte -> select(c : Compte | c.solde > 1000)
-- Retourne une collection contenant tous les comptes bancaires dont le solde est supérieur à 1000 €
compte -> reject(c : Compte | c.solde > 1000)
-- Retourne une collection contenant tous les comptes bancaires dont le solde est inférieur à 1000 €
compte -> collect( c : Compte | c.solde)
-- Retourne une collection contenant l'ensemble des soldes de tous les comptes

class Banque { …
-- il n'existe pas de clients de la banque dont l'age est inferieur à 18 ans
invariant pasDeMineurs : not( clients -> exists (age < 18) ) ; }

class Personne { …
-- si une personne est cliente d'une banque, elle a un compte dans cette banque
invariant compteDansBanque:
    banques->forAll (b : Banque |
    ( b.comptes->select(c : Compte | self.comptes->includes(c)) )   -> notEmpty()
                    ) ; }
```


Collections

- collect() renvoie toujours un Bag
  - Ensemble non ordonnée, doublons possibles
- Cast de collections
  - des opérations OCL dédiées pour transformer un type de collection en un autre type de collection
- Collections imbriquées
  - Via la navigation, on peut récupérer des collections ayant contenant d'autres collections
  - Deux modes de manipulation
    - Explicitement comme une collection de collections [de collections ...]
    - Collection unique : on « aplatit » le contenu de toutes les collections imbriquées en une seule à un seul niveau
      - Opération *flatten() pour aplatir une collection de collections*

UBO

Conformité de types

- Prise en compte des spécialisations entre classes du modèle
  - héritage
- Opérations OCL dédiées à la gestion des types

  oclIsTypeOf(type) -- retourne true si l'objet est du type *type*

  oclIsKindOf(type) -- retourne true si l'objet est du type *type ou un de ses sous-types*

  oclAsType(type) -- l'objet est « casté » en type *type*

- Types internes à OCL
  - Conformité entre les types de collection
    - 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

UBO

Contraintes conditionnelles

- Certaines contraintes sont dépendantes d'autres contraintes
- Deux modes d'expression

  if expr1 then expr2 else expr3 endif
  - Si l'expression *expr1* est *true* alors *expr2* doit être *true* sinon *expr3* doit être *true*

  expr1 implies expr2
  - Si l'expression *expr1* est *true*, alors *expr2* doit être *true* également
  - Si *expr1* est *false*, alors l'expression complète est *true*

Exemple de contraintes conditionnelles

```
class Personne {…
-- 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
invariant SeulementMajeurAUnCompte :
if age < 18  then compte -> isEmpty() else compte -> notEmpty() endif ; }

class Personne {…
-- Si une personne possède au moins un compte bancaire, alors elle est cliente d'au moins une banque
invariant banqueSiCompte:
compte -> notEmpty() implies banque -> notEmpty() ; }
```

Personne
String nom

```
class Personne {…
-- 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
invariant tousLesNomsSontDifferents :
Personne.allInstances() -> forAll(p1, p2 | p1 <> p2 implies p1.nom <> p2.nom) ; }
```


Déclarations

- Déclaration de variables : *let … in …*
  - Pour simplifier l'écriture d'une contrainte

```
class Personne {…
-- Une personne majeure doit avoir de l'argent
invariant globalPasEnNegatif:
let argent = compte.solde -> sum() in
age >= 18 implies argent > 0 ; }
```

sum() : fait la somme de tous les objets de la collection

Exemples de contraintes

```
class Personne {…
-- Une personne considérée comme au chômage ne doit pas avoir des revenus supérieurs à 500
invariant chomeurPauvre:
let money : Real = self.job.salary->sum() in
if isUnemployed then
      money < 500
else
      money >= 500
endif ; }
```


Exemples de contraintes (inutiles)

- Personne
  - *Les deux parents d'un enfant ne sont pas identiques (référence Unique)*

```
class Person {…
invariant ParentsConstraint:
let parent1 : Person = parents -> at(0) in
let parent2 : Person = parents -> at(1) in
if parent1.oclIsUndefined()
then
      if parent2.oclIsUndefined() then true
      else parent2.children->includes(self)
      endif
else
      if parent2.oclIsUndefined() then parent1.children->includes(self)
      else (parent1 <> parent2) and (parent1.children->includes(self)) and (parent2.children->includes(self))
      endif
endif ;
}
```

Exemples de contraintes (inutiles)

- Tous les enfants d'une personne ont bien cette personne comme parent et inversement (*référence EOpposite)*

```
class Person {…
invariant :
children -> notEmpty() implies children -> forAll ( p : Person | p.parents -> includes(self)) ;
invariant :
parents -> forAll ( p : Person | p.children -> includes (self)) ;
}
```

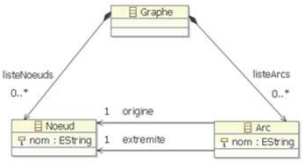

Exemples de contraintes

- Compagnie
  - Dans une compagnie, un manager doit travailler et avoir plus de 40 ans
- Personne
  - Une personne qui travaille doit avoir des revenus supérieurs à 1000 €
  - On ne peut pas être enfant de soi même

```
class Company
{…

invariant ManagerConstraint:
(manager.job->notEmpty()) and (manager.age > 40);

}

class Person
{…

invariant JobConstraint:
(job->notEmpty()) implies (job.salary > 1000.0);

invariant pasEnfantDeSoiMeme:
not children->includes(self);
}
```

Intégration dans EMF

- Installer OCL Tools
  - Help
  - Install Modeling Components -> OCL Tools
- Créer un invariant
  - Sélectionner le fichier ecore
  - Click droit / *Open With -> OCLinEcore Editor*
  - Intégrer l'invariant dans un classe



Graphe | Graphe.ecore | Graphe.ecore

```
package graphe : graphe = 'http://graphe/1.0'
{
    class Graphe
    {
        property listeNoeuds#graphe : Noeud[*] { ordered composes };
        property listeArcs : Arc[*] { ordered composes };
    }
    class Noeud
    {
        invariant toManyOutput:
            graphe.listeArcs->select(origine=self)->size()<=2;
        attribute nomNoeud : String { ordered };
        property graphe#listeNoeuds : Graphe { ordered };
    }
```


Intégration dans EMF

- platform:/resource/Graphes/model/Graphe.ecore
  - graphe
    - Ecore
      - invocationDelegates -> http://www.eclipse.org/emf/2002/Ecore/OCL/Pivot
      - settingDelegates -> http://www.eclipse.org/emf/2002/Ecore/OCL/Pivot
      - validationDelegates -> http://www.eclipse.org/emf/2002/Ecore/OCL/Pivot
    - Graphe
    - Noeud
      - Ecore
        - constraints -> toManyOutput
      - Pivot
        - toManyOutput -> graphe.listeArcs->select((origine = self))->size() <= 2
      - nomNoeud : EString
      - graphe : Graphe
    - Arc

UBO

Vérification

- Vérification de la contrainte
  - En édition séparée (lancement d'un éditeur Eclipse ou export du plugin)
    - Pas d'accès en mode « mise au point par création rapide de modèle »
  - Edition du modèle à vérifier, puis *Validate*
- Vérification de la correction de l'écriture de la contrainte
  - A l'exécution uniquement
  - Erreur de syntaxe OCL si pas de fenêtre de retour sur l'activation de *Validate*
- Retour d'erreur sur une violation de la contrainte
  - Nom de la contrainte
  - => attention au nommage des contraintes
    - Pas d'espace, pas de lettre accentuée dans un nom de contrainte

UBO

Références utilisées pour faire ce cours

- OMG
  - http://www.omg.org/
- Eclipse
  - http://www.eclipse.org/
- Christian W. Damus, IBM Rational Software « Implementing Model Integrity in EMF with MDT OCL », Feb. 2007
  - http://www.eclipse.org/articles/article.php?file=Article-EMF-Codegen-with-OCL/index.html
- Cours d'Eric Cariou
  - http://web.univ-pau.fr/~ecariou/
- Tutoriel Developpez
  - http://sbachmann.developpez.com/eclipse/exprimer-valider-contraintes-ocl-emf/

  « Exprimer et valider des contraintes OCL depuis un modèle EMF »