Chapitre 5 Langage de contraintes d'objet (OCL)

1. Introduction

UML (Unified Modeling Language) fournit des mécanismes permettant d'exprimer différents types de

contraintes, qui peuvent être classés comme suit :

- **Contraintes structurelles :** elles incluent les attributs au sein des classes, les différents types de relations entre les classes, la multiplicité (cardinalité) et la navigabilité des propriétés structurelles.
- **Contraintes de typage :** elles impliquent les types de données attribués aux propriétés.
- **Contraintes diverses :** Cette catégorie englobe les contraintes de visibilité (par exemple, publique, privée), ainsi que la définition de méthodes et de classes abstraites.

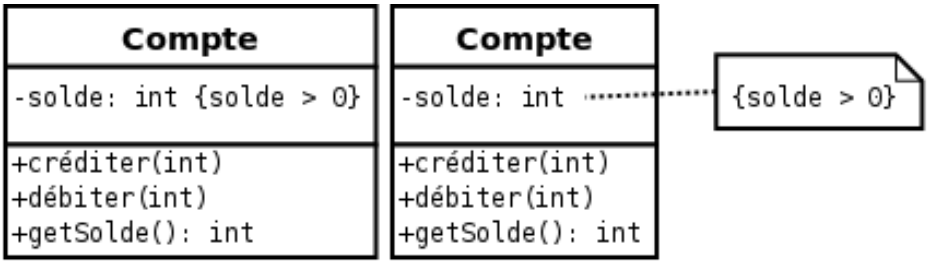
2. contraintes

Une contrainte est une condition ou une restriction sémantique exprimée sous forme d'énoncé dans un langage textuel. Ce langage peut être naturel ou formel. La chaîne de texte elle-même constitue le corps de la contrainte, qui peut être écrit dans divers langages :

- Une **langue naturelle** (par exemple, l'anglais, le français).
- Un **langage de contraintes dédié,** tel que l'Object Constraint Language (OCL).
- Ou même **la syntaxe directement issue d'un langage de programmation.**

2.1. Représentation des contraintes et des contraintes prédéfinies

ML propose plusieurs façons d'associer une contrainte à un élément de modèle, comme illustré dans les figures ci-dessous.



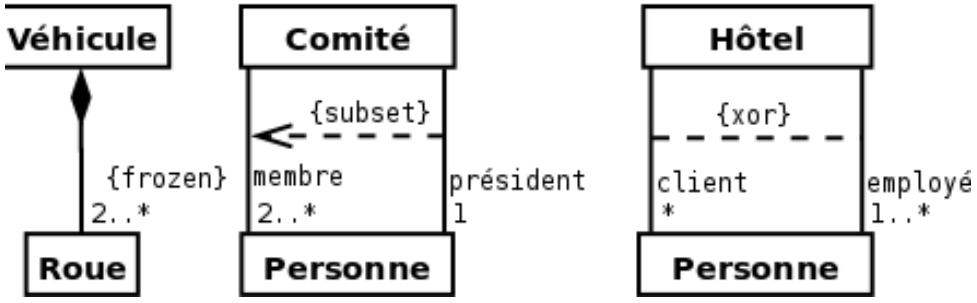


Figure 1 : Cette figure montre différentes méthodes d'application de contraintes en UML. Dans les deux diagrammes supérieurs, une contrainte spécifie qu'un attribut doit être positif. En bas à gauche, la contrainte {frozen} indique que le nombre de roues d'un véhicule ne peut pas changer. Au centre, la contrainte {subset} spécifie que le président est également membre du comité. En bas à droite, la contrainte { xor } (ou exclusif) stipule que les employés de l'hôtel ne sont pas autorisés à réserver une chambre dans le même hôtel où ils travaillent.

Figure 2 : Ce diagramme exprime les contraintes suivantes : une personne est née dans un pays, et cette association ne peut être modifiée. Elle a visité plusieurs pays, dans un ordre précis, et le nombre de pays visités ne peut qu'augmenter. Elle dispose d'une liste de pays qu'elle souhaite visiter, et cette liste est classée (probablement par préférence).
Comme nous venons de le voir, UML inclut plusieurs contraintes prédéfinies ({frozen} , {subset} , { xor } , {ordered} et { addOnly }). Pour pallier ces limitations, le langage OCL (Object Constraint Language) offre une solution élégante et plus performante.

3. Types de contraintes OCL

3.1 Contraintes invariantes

Une condition qui doit toujours être vraie pour toutes les instances d'une classe.

Syntaxe:

contexte ClassName
inv InvariantName : expression_booléenne

Exemple:

contexte Compte bancaire
inv PositiveBalance : solde >= 0
inv Limite de découvert : solde >= - découvert autorisé

3.2 Contraintes de précondition

Une condition qui doit être vraie avant qu'une opération soit exécutée.

Syntaxe:

contexte ClassName :: operationName (paramètres) : ReturnType
pre PreconditionName : expression_booléenne

Exemple:

contexte CompteBanque :: retirer(montant : réel)
pre PositiveAmount : montant > 0
pré Solde suffisant : solde - montant >= - découvert autorisé

3.3 Contraintes de postcondition

Une condition qui doit être vraie après l'exécution d'une opération.

Syntaxe:

contexte ClassName :: operationName (paramètres) : ReturnType
post PostconditionName : expression_booléenne

Exemple:

contexte CompteBanque :: retirer(montant : réel)
post mis à jour : solde = solde@pré - montant

4. Diagramme de classes avec OCL

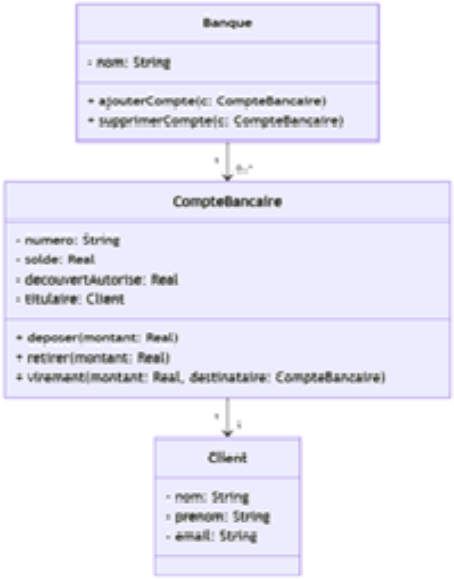


Figure 3 diagramme de classes pour un système bancaire simple.

4.1. Contraintes OCL associées

Contraintes du compte bancaire :

contexte Compte bancaire
inv Solde non négatif : solde >= - découvert autorisé
inv DécouvertValide : autoriséDécouvert >= 0

contexte CompteBanque :: dépôt(montant : réel)
pre ValidAmount : montant > 0
post augmenté : solde = solde@pré + montant

contexte CompteBanque :: retirer(montant : réel)
pre ValidAmount : montant > 0
pré Solde suffisant : solde - montant >= - découvert autorisé
post- diminué : solde = solde@pré - montant

Contrainte sur la banque :
contexte Banque
inv UniqueAccounts : compte-> pour tous (c1, c2 | c1 <> c2 implique c1.number <> c2.number)

5. Types de données et collections dans OCL

5.1 Types de base

Type OCL	Description	Exemple
Booléen	Valeurs logiques	vrai , faux
Entier	Nombres entiers	5 , -10 , 0
Réel	Nombres réels	3,14 , -2,5
Chaîne	Séquences de caractères	"Bonjour"

5.2 Collections

Types de collections :

- **Ensemble :** Collection non ordonnée sans doublons.
- **OrderedSet :** Collection ordonnée sans doublons.
- **Séquence :** Liste ordonnée qui autorise les doublons.
- **Sac :** Collection non ordonnée qui autorise les doublons (multiset).

Exemples:
Déclarations de recouvrement
Ensemble { 1, 2, 3, 2} -- Résultat : {1, 2, 3}
Séquence { 1, 2, 3, 2} -- Résultat : [1, 2, 3, 2]

Sac{ 1, 2, 3, 2} -- Résultat : Sac{1, 2, 2, 3}

5.3. Opérations fondamentales sur les collections

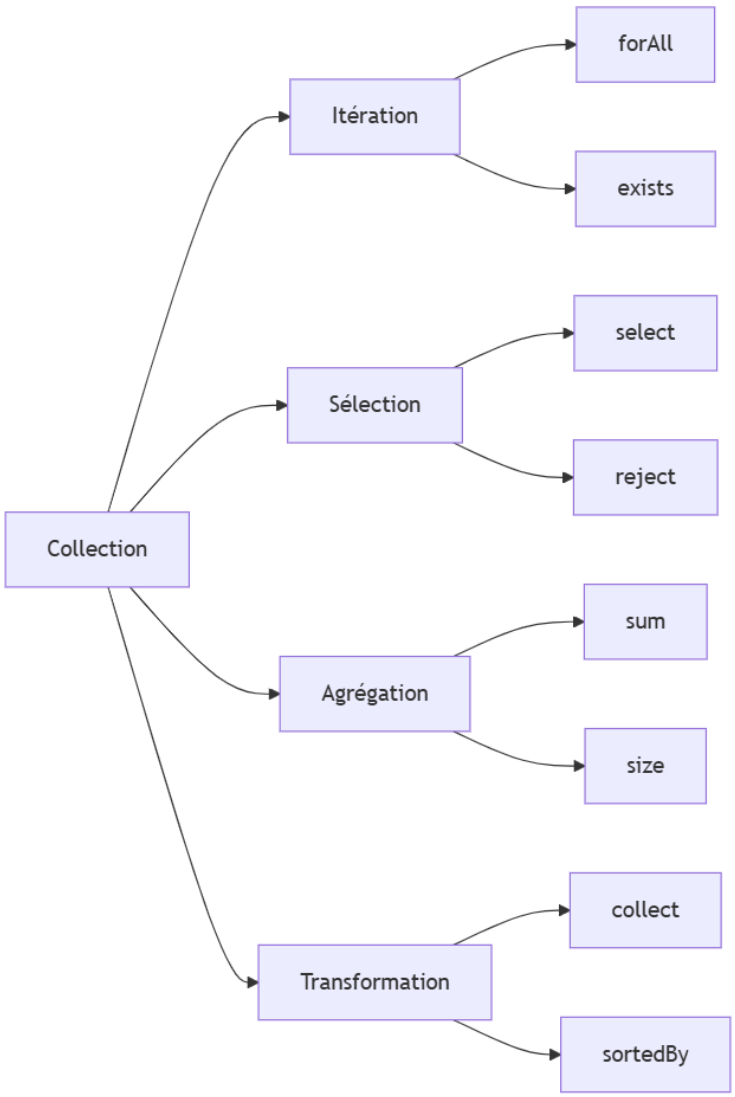


Figure 4 diagramme résumant les opérations de collecte OCL courantes.

5.4. Exemples d'opérations

En supposant une collection : `clients : Set(Client)`

Sélection

contexte Banque

def : majorClients = clients-> select(âge >= 18)

Rejet
contexte Banque
def : minorClients = clients-> rejeter(âge >= 18)

Quantification existentielle
contexte Banque
inv : atLeastOneRichClient : clients-> existe(solde > 100000)

Quantification universelle
contexte Banque
inv : allValidClients : clients-> forAll (c | c.name.size () > 0)

Agrégation
contexte Banque
def : totalDeposits = comptes.balance ->sum()
(Pour chaque banque, le total des dépôts est défini comme la somme des soldes de l'ensemble de ses comp tes).
Transformation
contexte Banque
def : customerNames = clients.name-> asSet ( )
(Pour chaque banque, la liste des noms de clients est définie comme l'ensemble des noms de tous ses client s, après suppression des doublons).

6. Navigation dans les associations

6.1 Navigation simple

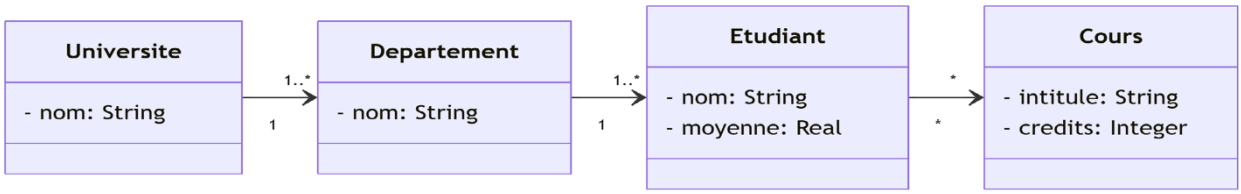


Figure 5 Diagramme de classe montrant les associations entre l'université, le département et l'étudiant.

Contraintes OCL :
contexte universitaire

- Tous les départements doivent avoir au moins 10 étudiants

inv : départements-> forAll (d | d.students ->size() >= 10)

-- Au moins un élève doit avoir une moyenne supérieure à 15
inv : départements.étudiants ->existe(moyenne > 15)

-- Navigation complexe
(Pour chaque département, le total des crédits étudiants est défini comme la somme de l'ensemble des créd its de tous les cours suivis par tous ses étudiants).
contexte Département
def : totalStudentCredits = étudiants.cours .credits ->sum()

6.2 Navigation avec des conditions complexes

contexte universitaire
-- Étudiants inscrits à au moins 5 cours
def : activeStudents = départements.étudiants ->select(s | s.cours ->size() >= 5)

-- Note moyenne des étudiants par département
def : averagePerDepartment = départements-> collect(d | d.students.average -> avg ())

7. Étude de cas : Système de réservation

7.1 Diagramme de classe

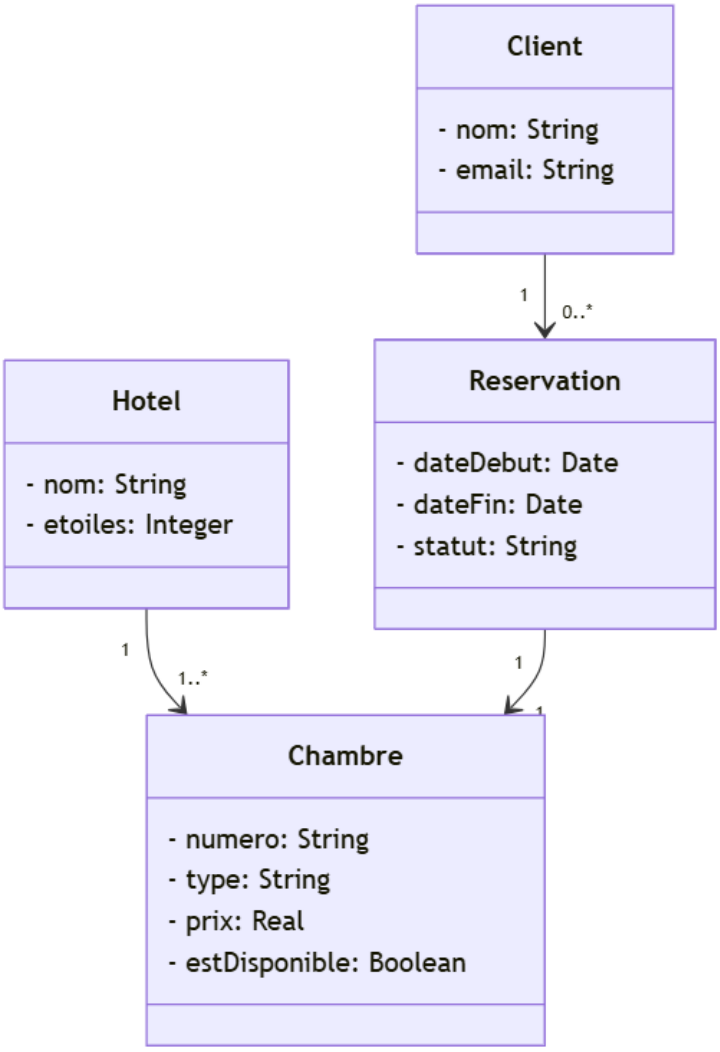


Figure 6 Diagramme de classes pour un système de réservation d'hôtel.

7.2 Contraintes commerciales dans OCL

Invariant validStars :

"Le nombre d'étoiles d'un hôtel doit être compris entre 1 et 5."

Invariant uniqueRooms :

"Toutes les chambres d'un hôtel doivent avoir des numéros différents."

contexte Hôtel

inv : validStars : étoiles >= 1 et étoiles <= 5
inv : uniqueRooms : rooms-> forAll (r1, r2 | r1 <> r2 implique r1.number <> r2.number)

Invariant positivePrix :

"Le prix d'une chambre doit être strictement positif."

Invariant validType :

"Le type d'une chambre doit être 'Single', 'Double' ou 'Suite'."

contexte Salle
inv : positivePrix : prix > 0
inv : validType : type = 'Single' ou type = 'Double' ou type = 'Suite'

Invariant validDates :

"La date de début d'une réservation doit être antérieure à sa date de fin."

Invariant maxDuration :

"La durée d'une réservation ne peut pas dépasser 30 jours."

contexte Réservation
inv : validDates : date de début < date de fin
inv : maxDuration : (endDate - startDate) <= 30 -- Maximum 30 jours

Pré-condition roomAvailable :

"Pour créer une réservation, la chambre doit être disponible."

Pré-condition datesInFuture :

"Pour créer une réservation, la date de début doit être dans le futur."

Post-condition roomReserved :

"Après création d'une réservation, la chambre n'est plus disponible."

Post-condition statutConfirmé :

"Après création d'une réservation, son statut est 'Confirmé'."

contexte Réservation:: create(startDate : Date, endDate : Date, room: Room)
pré : roomAvailable : room.isAvailable = true
pré: datesInFuture : startDate > Date:: now()
post : roomReserved : room.isAvailable = false
post : statutConfirmé : statut = 'Confirmé'

Invariant réservations non superposées :

"Aucune des réservations d'un client ne doit chevaucher une autre réservation du même client."

contexte Client
inv : réservations non superposées :
réservations-> forAll (r1, r2 |
r1 <> r2 implique
(r 1.endDate < r2.startDate ou r2.endDate < r1.startDate)
)

8. Conclusion

Le langage OCL se révèle être un composant indispensable dans l'écosystème de modélisation MDA, venant **compléter UML** en apportant la précision et la rigueur formelle qui manquaient aux seuls diagrammes visuels. À travers ses **trois principaux types de contraintes** - les invariants pour garantir l'intégrité permanente des objets, les préconditions pour valider les entrées des opérations, et les postconditions pour spécifier leurs effets - OCL permet d'exprimer sans ambiguïté les règles métier les plus complexes. Sa capacité de **navigation puissante** à travers les associations du modèle et son **riche ensemble d'opérations** sur les collections permettent de formuler des contraintes sophistiquées qui seraient difficiles à exprimer autrement. Enfin, son caractère **déclaratif et sans effets secondaires** en fait un outil fiable pour la spécification, la validation et la génération de code, assurant ainsi la cohérence entre la conception et l'implémentation tout au long du processus de développement dirigé par les modèles.

Références:

- Spécification OMG OCL 2.4
- « UML 2 et MDE » par Joël Champeau