O C L

Object Constraint Language

Développé par IBM en 1995

Inclu dans standard UML (→ 1.1) en 1997

OCL 2.0 en 2006 / OCL 2.2 en 2010

Objectifs

UML est insuffisant pour décrire totalement un projet

- Langage naturel → verbeux, risque d'ambiguïté

- Langage formel → mathématique
  peu compatible avec notion métier

- OCL → langage de spécification

Caractéristiques

- Langage de pure spécification → Pas d'effet de bord

- Une expression OCL retourne une valeur
→ Pas de modification du modèle

- Ce n'est pas un langage de programmation

- Langage typé

Objectifs

Enrichir les diagrammes objets

- Définition des invariants, pré et post-conditions
- Indication des valeurs ou collections de données
- Expression des contraintes
- Attributs dérivés, ...

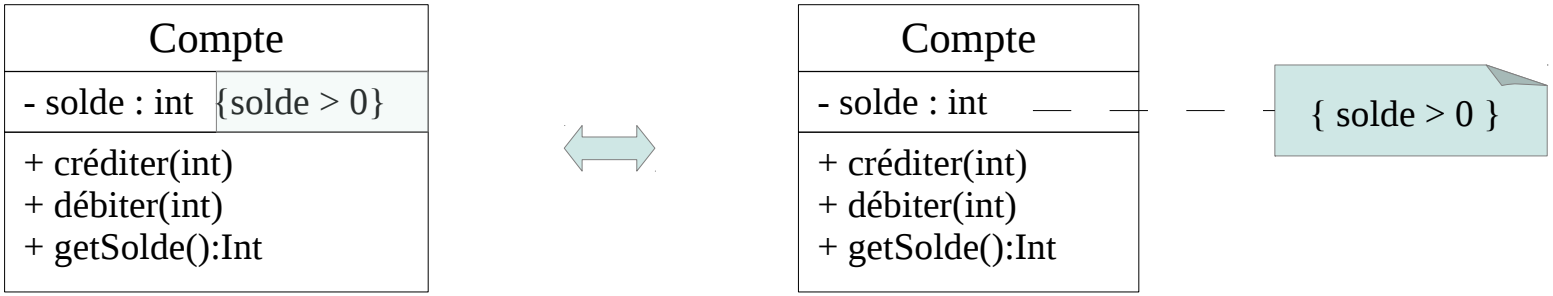
Objectifs

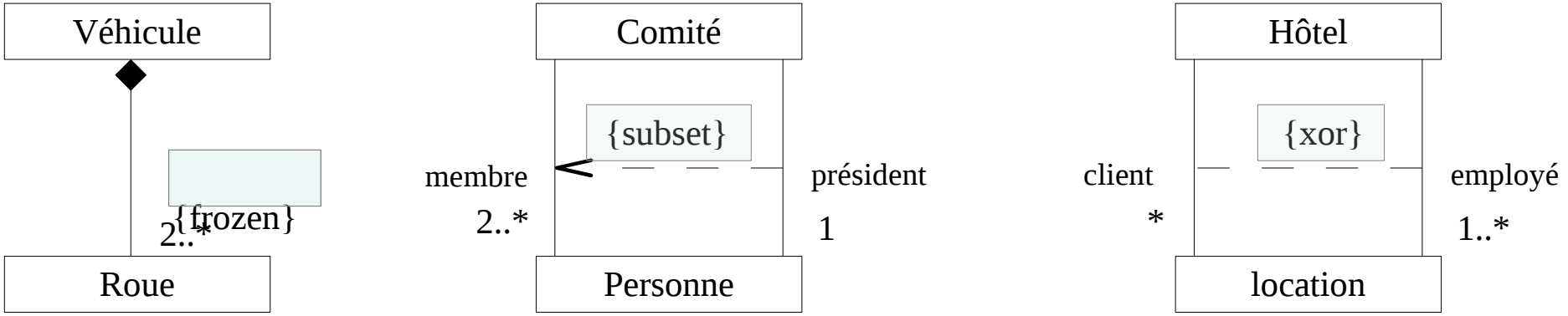
Utilisation dans les diagrammes UML

- Diagramme de séquence *garde*
- Diagramme d'état / transition *garde*
- Diagramme de classe *contrainte*
- ...

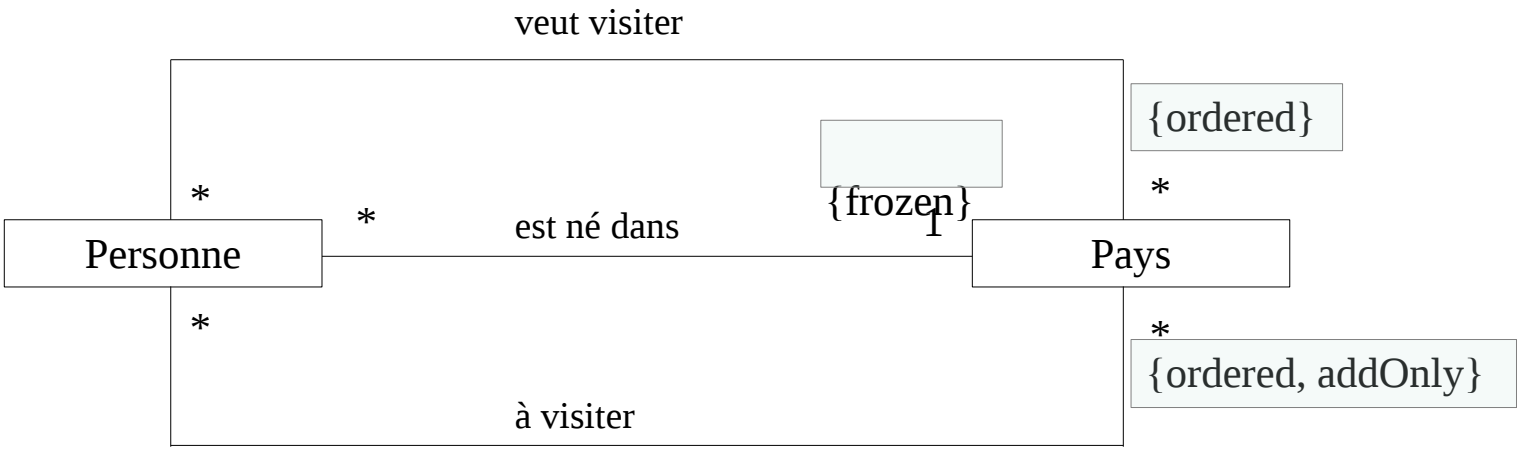
OCL

Exemple d'expression de contraintes

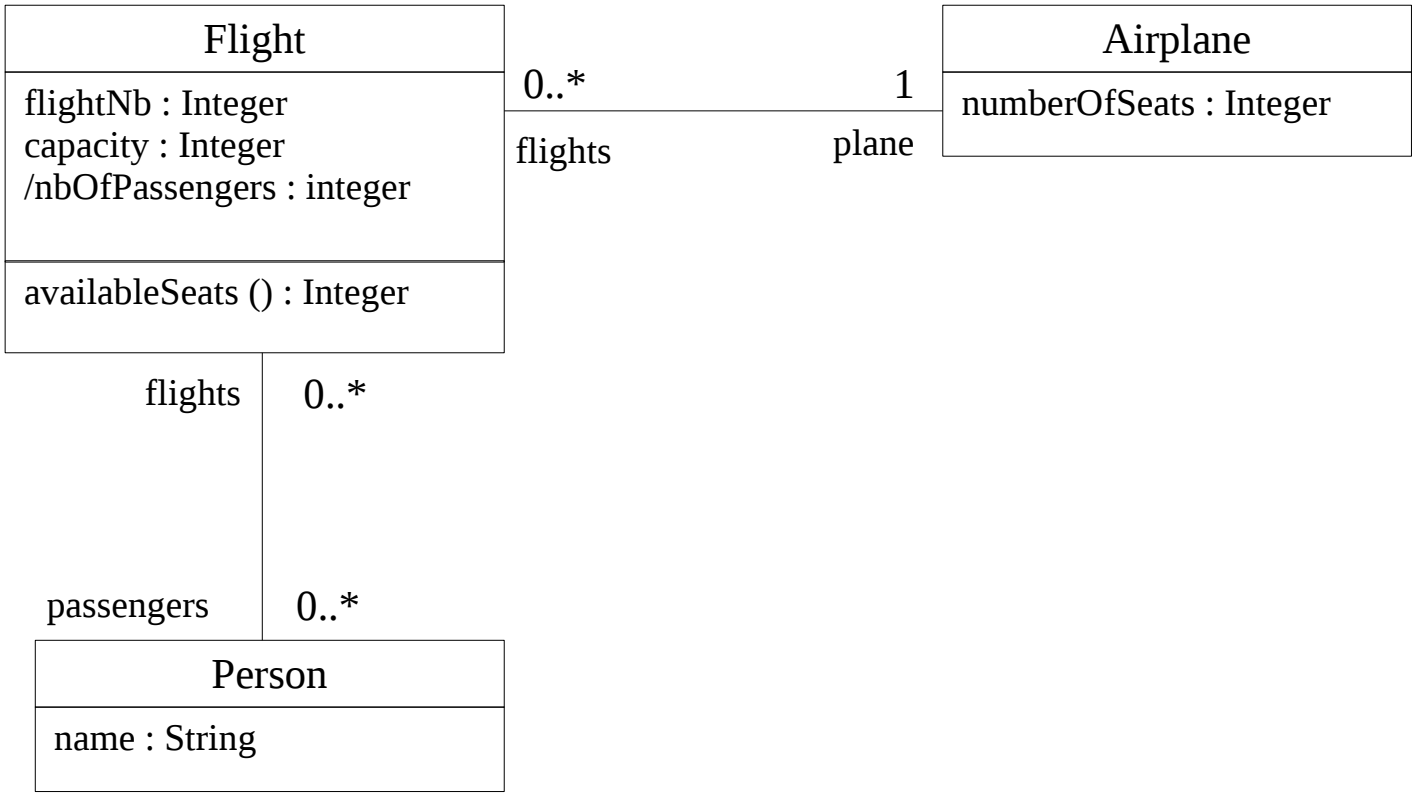




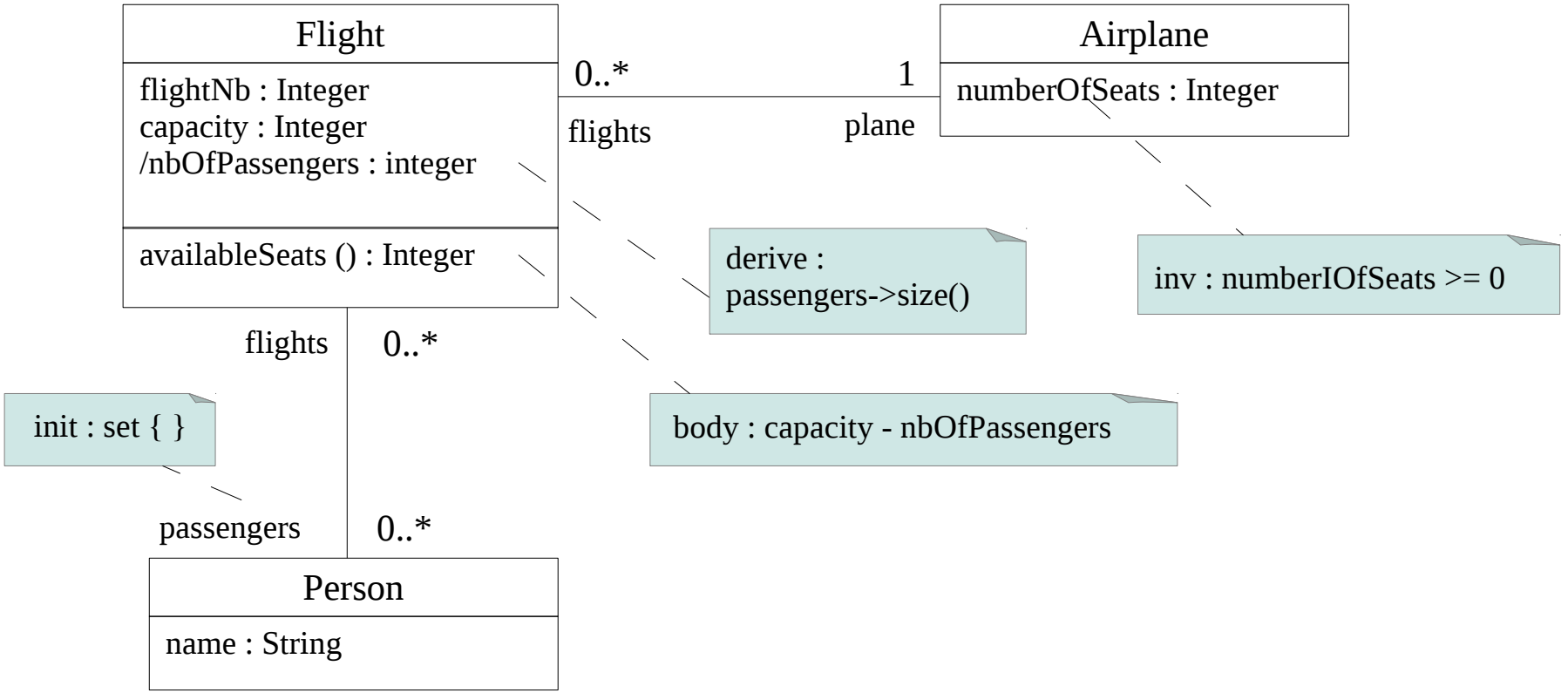
Exemple d'expression de contraintes



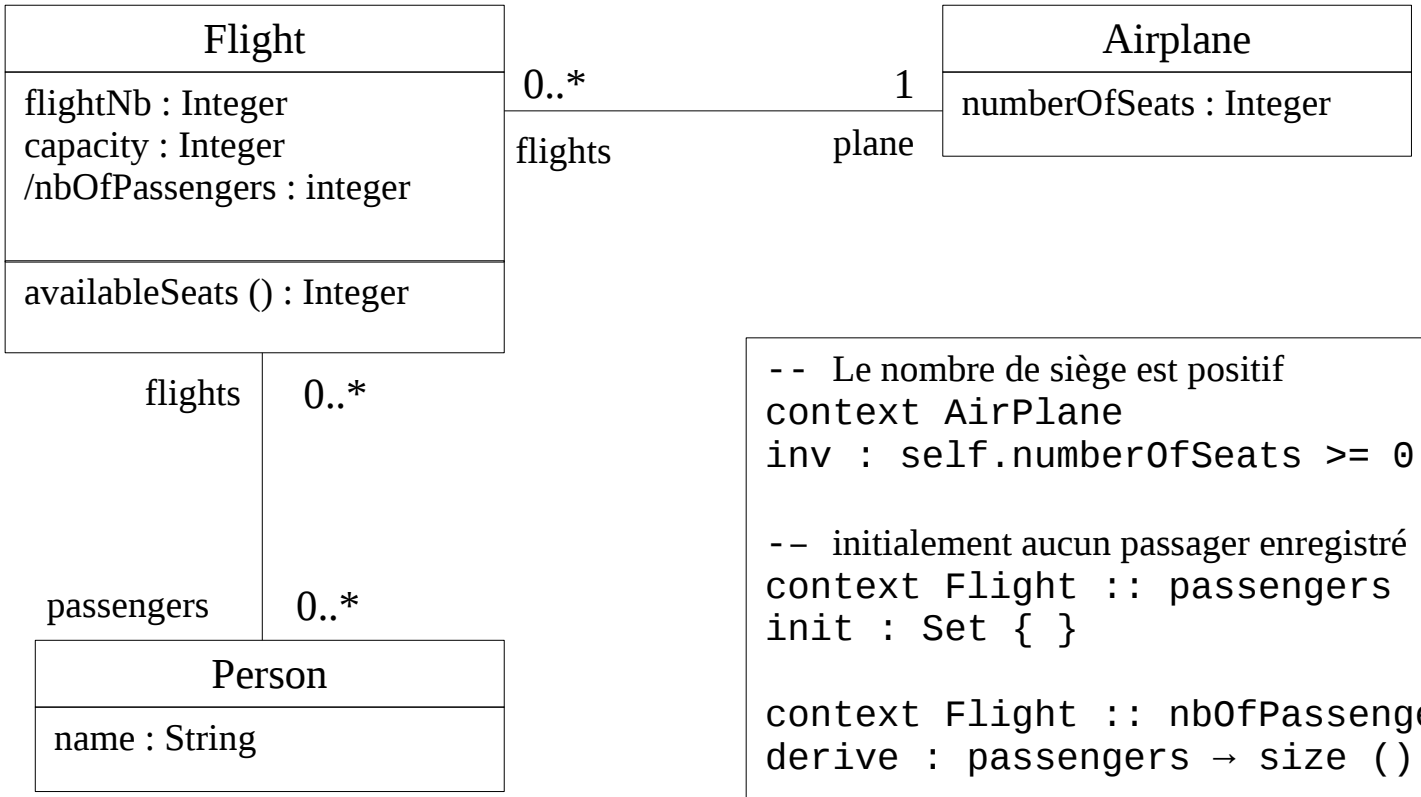
Exemple : Diagramme de classe imprécis



Exemple : Diagramme de classe décoré avec OCL



Exemple : Fichier OCL séparé (version textuelle)




```
-- Le nombre de siège est positif
context AirPlane
inv : self.numberOfSeats >= 0

-- initialement aucun passager enregistré
context Flight :: passengers : Set (Person)
init : Set { }

context Flight :: nbOfPassengers : Integer
derive : passengers → size ()

context Flight :: availableSeats () : Integer
body : capacity - numberOfPassengers
```

Syntaxe OCL

Mots réservés

and	*attr*	*context*	*def*
else	*endif*	*endpackage*	*if*
implies	*in*	*inv*	*let*
not	*oper*	*or*	*package*
post	*pre*	*then*	

Commentaires

```
-- this is a comment
```

Contexte

Elément du modèle auquel est rattachée l'expression

Si contexte implicite → définition optionnelle

exemple :

- *Contexte défini explicitement par l'appel à l'élément*

- *Contexte implicite par le stéréotype*

- *Contexte implicite par le diagramme*

Contexte

Elément du modèle auquel est rattachée l'expression

Le mot clé `self` fait référence à cet élément

```
context p : Person
inv : p.name <> ''
```

```
context Person
inv : name <> ''
```

```
context Person
inv : self.name <> ''
```

Spécification d'une propriété

Notation par 4 points : :

```
context TypeName :: AttributeName : TypeAttribute
```

Exemples :

```
context Flight :: capacity : Integer
contect Flight :: avaliableSeats() : Integer
context Flight :: passengers : Set(Person)
```

Syntaxe générale d'une contrainte

```
context TypeName::operationName(par1:Type1,...):ReturnType
```

Invariant

Stéréotype `<<invariant>>`

- Si invariant associé à un Classifier → considéré comme un type

- Expression OCL = contrainte sur ce type

- La contrainte doit être respectée par toutes les instances du type

- L'expression OCL exprimant un invariant est un booléen

OCL

Invariant

Base

```
context Company inv:
self.numberOfEmployees > 50
```

Contexte nommé

```
context c: Company inv:
c.numberOfEmployees > 50
```

Contrainte nommée

```
context Company inv enoughEmployee:
c.numberOfEmployees > 50
```

Contraintes et contrats

Invariant de classe

```
context Account
inv balanceCorrect : self.balance >= self.min
inv negativeMin : self.min <= 0
```

UML => stéréotype `<<invariant>>`

Expression des contraintes facile

! OCL ne précise pas ce qui se produit en cas de violation du contrat

Pré-conditions

Stéréotype `<<precondtions>>`

- Condition booléenne devant être vraie à l'appel de l'opération
  Doit être vérifiée par l'utilisateur

- Variables évaluée avec leur valeur d'avant l'appel

- Valeur avant appel : mot clé `@pre`

Pré-conditions

```
context Typename::opName(param1:Type1,…):ReturnType
pre : param1 > …
post : result = …
```

`result` exprime le résultat fourni par l'expression s'il existe

```
context Compte :: debiter (montant : Float)
pre : montant > 0 and
      montant <= self.solde@pre - self.mini
```

`self.solde@pre` => valeur avant appel

Post-conditions

Stéréotype `<<postcondtions>>`

- Condition booléenne devant être vraie en fin d'opération

- Possibilité de nommer les pre/post-conditions

- Référencement du résultat de l'opération `result`

```
context Compte :: debiter (montant : Float)
pre cmptValid : self.solde@pre > 0
post : self.solde = self.solde@pre - montant
```

Spécification du corps d'une opération

Stéréotype `<<body>>`

- Une expresion OCL peut définir une opération
- Résultat retourné compatible avec le contexte
- L'expression doit rester conforme aux autres contraintes
- Ordre sans incidence sur les contraintes (pas d'effet de bord)
- Description exacte du résultat par mot clé `body`

Spécification du corps d'une opération

Exemple :

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


context Person::getCurrentSpouse() : Person
pre: self.isMarried = true
body : self.mariages→select(m | m.ended = false).spouse
```

Package

Définition d'un package

```
package Package::SubPackage
context …                          – contenu du package
...
endPackage
```

Let

Réutilisation d'une sous expression dans une contrainte

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

Def

- Réutilisation d'une sous expression au sein d'un même Classifier

- Ne peut contenir qu'une seule variable / expression

- Toutes ces variables / expressions sont connues dans le contexte

```
context Person
def : income : Integer = self.job.salary→sum() in
def : nickname : String = 'Little Red Rooster'
de : hasTitle(t : String) : Boolean
                          = self.job→exists(title = t)
```

Valeurs initiales et dérivées

Valeur initiale `init /` Valeur dérivée `derive`

```
-- Solde initialisé à 0
context Compte :: solde : Float
init : 0

-- Compte rémunéré
context Compte :: remuneration : Float
derive : self.solde * 0,5 %
```

Structures conditionnelles

opérateur `implies / if … then … else … endif`

```
context Compte
inv : self.solde < 100 implies self.remuneration = 0

context Compte
inv : if self.solde < 100
      then self.remuneration = 0.001 * self.solde
      else self.remuneration = 0.005 * self.solde
      endif
```

Variables

Factorisation d'une sous-expression

```
context Compte::ajouterInteret(pourcent:Real):Real
post ajoutEffectue :
   let facteur : Real = 1 - pourcent/100
   in solde = solde@pre * facteur
```

`facteur` n'est utilisable que dans cette post-condition

OCL

Langage typé

Format	Type	Valeur
Booléen	`Boolean`	`true, false`
Entier	`Integer`	`0, -5, 1237`
Réel	`Real`	`1.5, 3.14159`
Chaîne de car	`String`	`'mon texte'`

! Format OCL indépendant des formats des langages de programmation

OCL

Langage typé

Type	Opérateur
`Boolean`	`AND  OR  XOR  NOT  IMPLIES`
`Integer`	`+  -  *  /  abs()  ...`
`Real`	`+  -  *  /  abs()  ...`
`String`	`concat()  size()  substring()  ...`

Comparaisons impossibles entre types différents sauf Integer et Real (Integer est un sous-type de Real)

Collections

`Collection`	-- super type abstrait
`Set`	-- type ensemble mathématique (sans doublon)
`OrderedSet`	-- type set ordonné
`Bag`	-- multi ensemble mathématique (accepte les doublons)
`Sequence`	-- type suite (doublon possible, ordre, …)
Tuple	Eléments nommés

Remarque : UML et OCL 2.0 autorisent Les collections de collections

Collections

exemples :

- `Set(1,2,5,88), Set("pomme", "poire")`

- Sequence { 1 ...(6+4) } 3 écritures identiques
  Sequence { 1 … 10 }
  Sequence { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 }

- Tuple { name:String = 'John', age:Integer = 10 }

  → Les éléments d'un Tuple peuvent être de formats différents

- collection1 -> union(collection2)

Conformance

Conformance error → Si types non repectés dans une expression

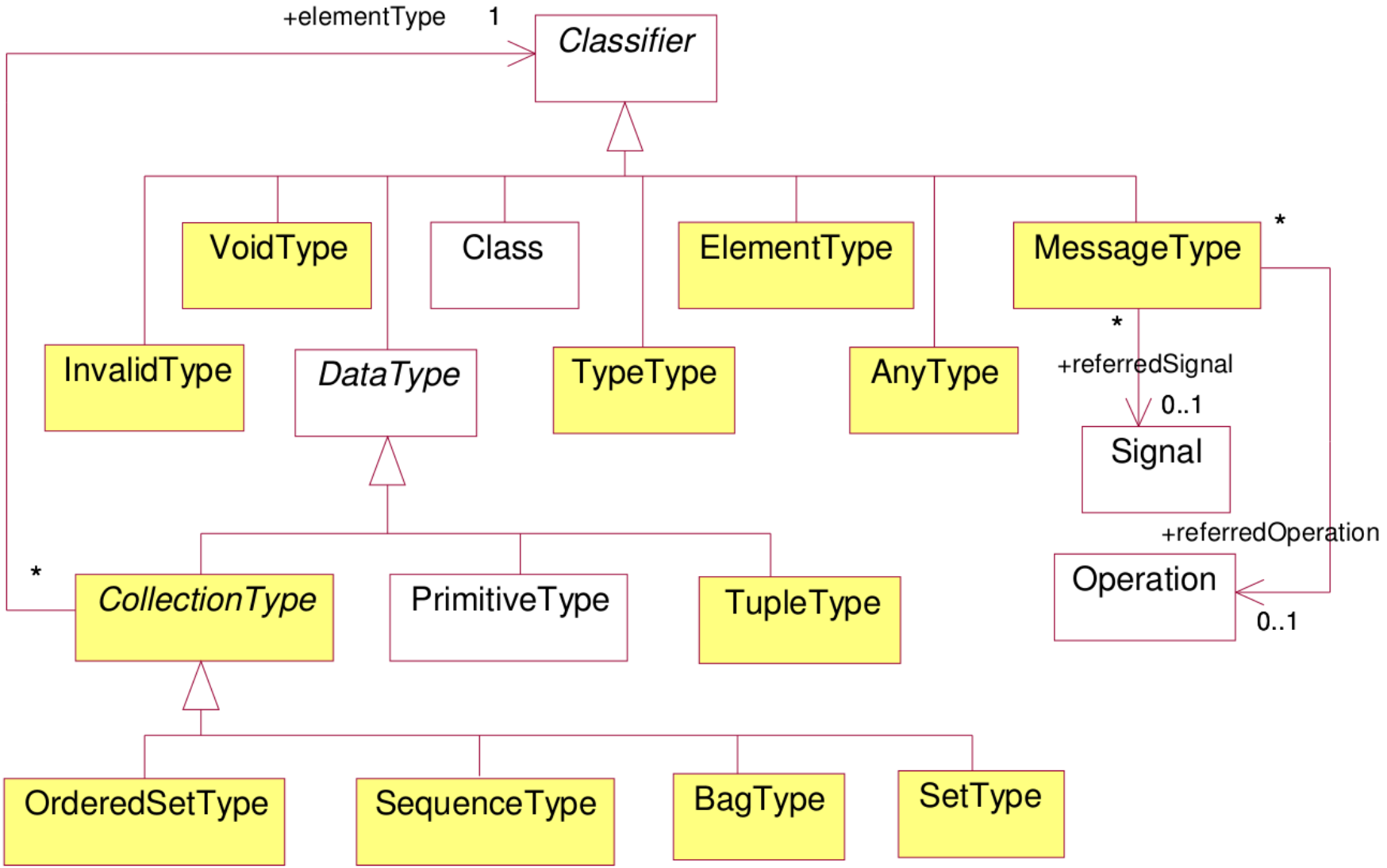
```
Set
Sequence          Subtype of Collection
Bag

Integer           Subtype of Real
```

Cast

object.oclAsType(Type2) Evalue "object" avec le format "Type2"

OCL


+elementType
1
Classifier
VoidType
Class
ElementType
MessageType
*
InvalidType
DataType
TypeType
AnyType
*
+referredSignal
0..1
Signal
+referredOperation
*
CollectionType
PrimitiveType
TupleType
Operation
0..1
OrderedSetType
SequenceType
BagType
SetType

Enumeration

Enumeration est un <<Datatype>> pour UML

- Peut-être nommé comme toute entité

- Contient une liste de valeurs littérales

- Une expression OCL peut faire référence à une valeur d'énumération

exemple

```
context Person inv : gender = Gender::male
```

```
avec Gender : énumération { male, female }
```

OCL

Opérateurs

`::`	Désigne un élément dans un objet englobant
`.`	Accède à une caractéristique d'un objet
`->`	Accède à une caractéristique d'une collection
`self`	Objet désigné par le contexte

`self.client[19503800].nom` Accède au nom du client dont le numéro est 19003800

Ordre de précédence

- @pre
- dot and arrow operations: ‘.’ and ‘->’
- unary ‘not’ and unary minus ‘-’
- ‘*’ and ‘/’
- ‘+’ and binary ‘-’
- ‘if-then-else-endif’
- ‘<’, ‘>’, ‘<=’, ‘>=’
- ‘=’, ‘<>’
- ‘and’, ‘or’ and ‘xor’
- ‘implies’

Fonctions

Opérations sur les types : collections

`size():Integer`	Taille de la collection
`includes(object:T):Boolean`	Vrai si col. contient T
`excludes(object:T):Boolean`	Vrai si col. ne contient pas T
`count(object:T):Integer`	Nombre d'occurences de T
`includesAll(c2:Collection(T)):Boolean`	Vrai si col. contient c2
`excludesAll(c2:Collection(T)):Boolean`	Vrai si col. ne contient pas c2
`isEmpty():Boolean`	Vrai si collection vide
`notEmpty():Boolean`	Vrai si collection non vide
`sum():T`	Retourne la somme des T (T doit accepter +)
`product(c2:Collection(T2)):Set(Tuple(first:T,second:T2))`	=> Colection de Tuples

Opérations sur les types : collections

Remarque : Notation fléchée

```
context Person
def : accountNumber():Integer=self.account->size()
```

Itération sur collection

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

Opérations sur les types : set

`union(s:Set(T)):Set(T)`	Retourne union de self et set
`intersection(s:Set(T)):Set(T)`	Retourne intrsection de self et set
`=(s:Set(T)):Boolean`	Vrai si self = set
`including(object:T):Set(T)`	Renvoie self + set
`excluding(object:T):Set(T)`	Renvoie self - set
`asOrderedSet():OrderedSet(T)`	Conversion self en OrderedSet
`asSequence():Sequence(T)`	Conversion self en Sequence
`asBag():Bag(T)`	Conversion self en Bag

Fonctions identiques pour les formats Bag et Sequence

Génération d'une collection

Génère une collection :

```
collect( [ <élément> [ : <Type> ] | ] <expression> )
```

Exemples :

```
self.employé->collect(date_de_naissance)
```

Remarque : Résultat de type Bag (doublons possibles)

Opérations sur les éléments d'une collection

Génère une sous-collection :

```
collection->select(v:Type | boolean-expression-with-v)

collection->select(v:Type | not(boolean-expression-with-v))

collection->select(boolean-expression)


collection->reject(v:Type | boolean-expression-with-v)
```

Opérations sur les éléments d'une collection

Exemples :

```
context Société
inv: self.employé→select(p:Person | p.age > 50)->notEmpty()

context Société
inv: self.employé->select(individu |
                          individu.age > 50)->notEmpty()

context Société
inv: self.employé->select(individu : Personne |
                          individu.age > 50)->notEmpty()
```

Opérations forAll et Exists

Retourne une expression booléenne :

```
forAll( [ <élément> [ : <Type> ] | ] <expression_logique> )
exists( [ <élément> [ : <Type> ] | ] <expression_logique> )
```

Exemples :

```
context Personne
inv: Personne.allInstances()->forAll(p1, p2 | p1 <> p2
                                        implies p1.nom <> p2.nom)

context Personne
inv :(Personne.allInstances().product(
             Personne.allInstances()))->forAll(
            tuple | tuple.first <> tuple.second
            implies tuple.first.nom <> tuple.second.nom)
```

Collections de toutes les instances

Retourne une collection :

`classe.allInstances()`

Retourne une collection construite avec toutes les instances de la classe désignée (à l'instant de création)

Types OCL avancés

`oclInvalid`	ne contient que `invalid`
`oclVoid`	ne contient que `null`
`oclAny`	tout type OCL

`oclAsType()` Caractéristique redéfinie dans une sous-classe

B hérite de A : propriété p1 définie dans les 2 classes

contexte B → accès à p1 de B `self.p1`

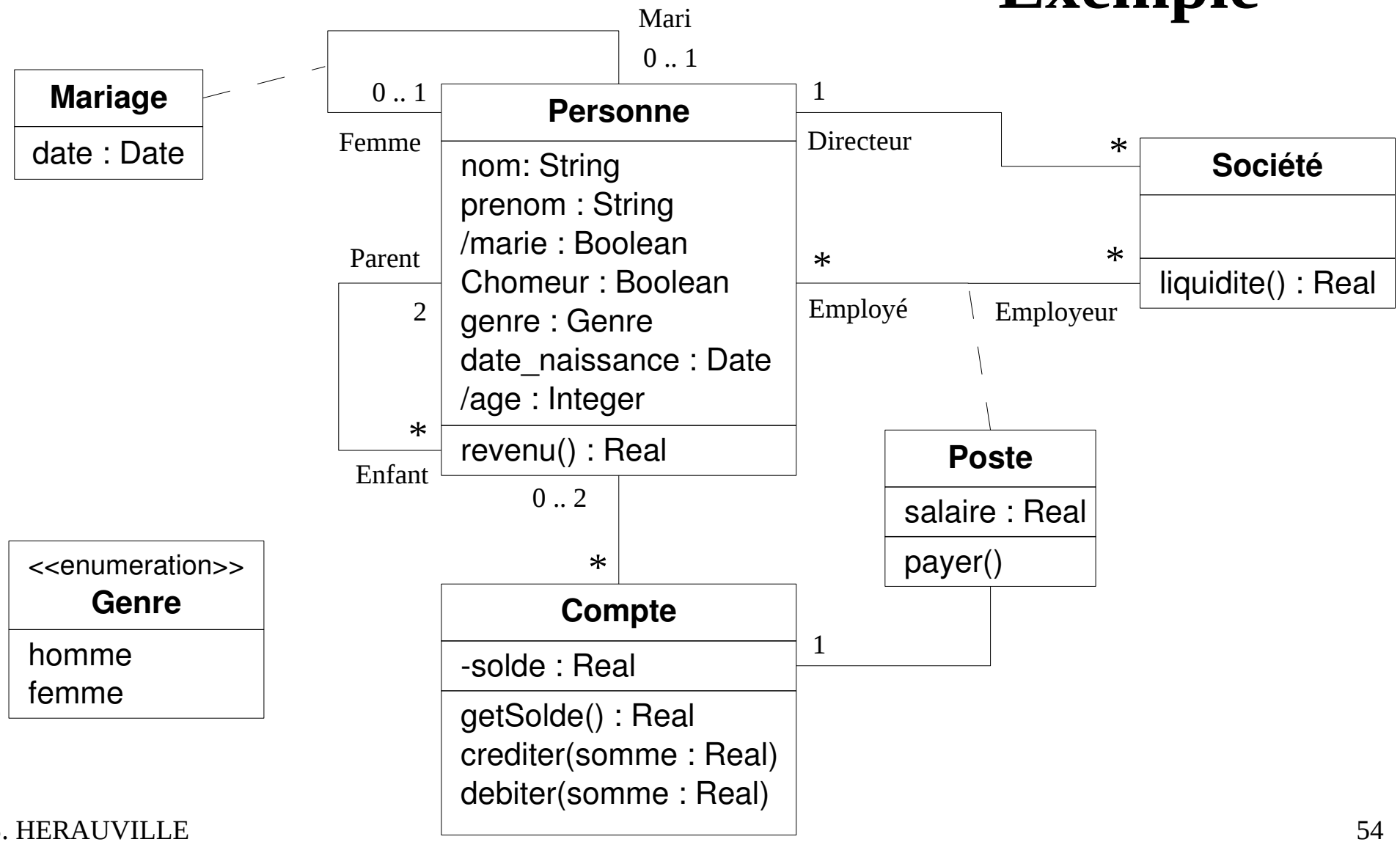
contexte B → accès à p1 de A `self.oclAsType(A).p1`

Types OCL avancés

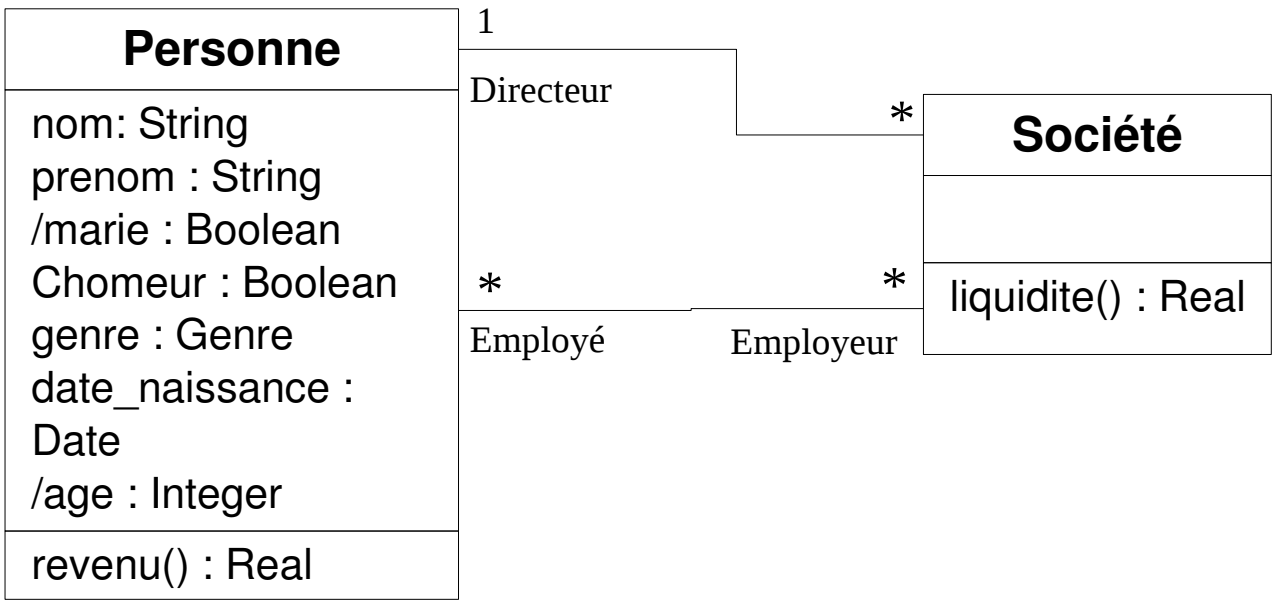
`oclIsTypeOf(t:OclType):Boolean`	Vrai si type = type t
`oclIsKindOf(t:OclType):Boolean`	Vrai si type = type t ou parent
`oclIsNew():Boolean`	Vrai si créé dans opération (utilisé dans post-condition)
`oclInState(s:OclState):Boolean`	Vrai si état s actif dans diagramme (pour diagramme état-transition)

Exemples

Exemple


Mariage
date : Date
Mari
0 .. 1
0 .. 1
Femme
Personne
nom: String
prenom : String
/marie : Boolean
Chomeur : Boolean
genre : Genre
date_naissance : Date
/age : Integer
revenu() : Real
1
Directeur
*
Société
liquidite() : Real
Parent
2
*
Enfant
*
*
Employé
Employeur
Poste
salaire : Real
payer()
0 .. 2
*
<<enumeration>>
Genre
homme
femme
Compte
-solde : Real
getSolde() : Real
crediter(somme : Real)
debiter(somme : Real)
1

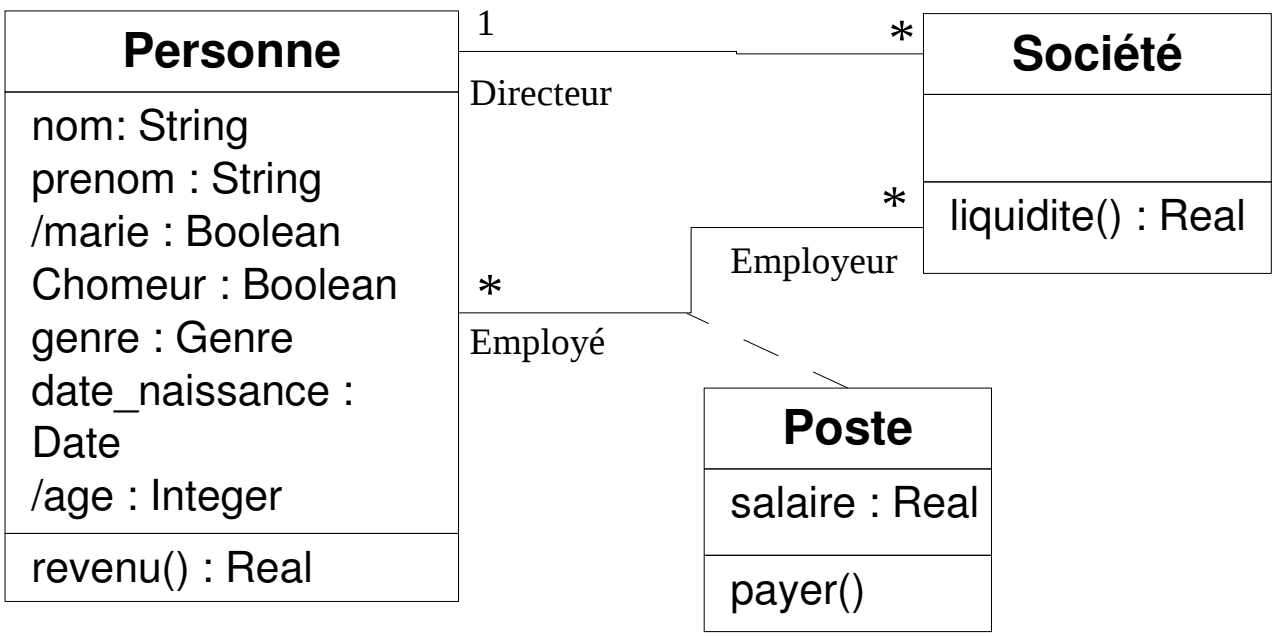
Contraintes




```
context Société
inv : self.directeur->size()=1 and
      not(self.directeur.chômeur) and
      self.directeur.age > 40 and
      self.employé->includes(self.directeur)
```

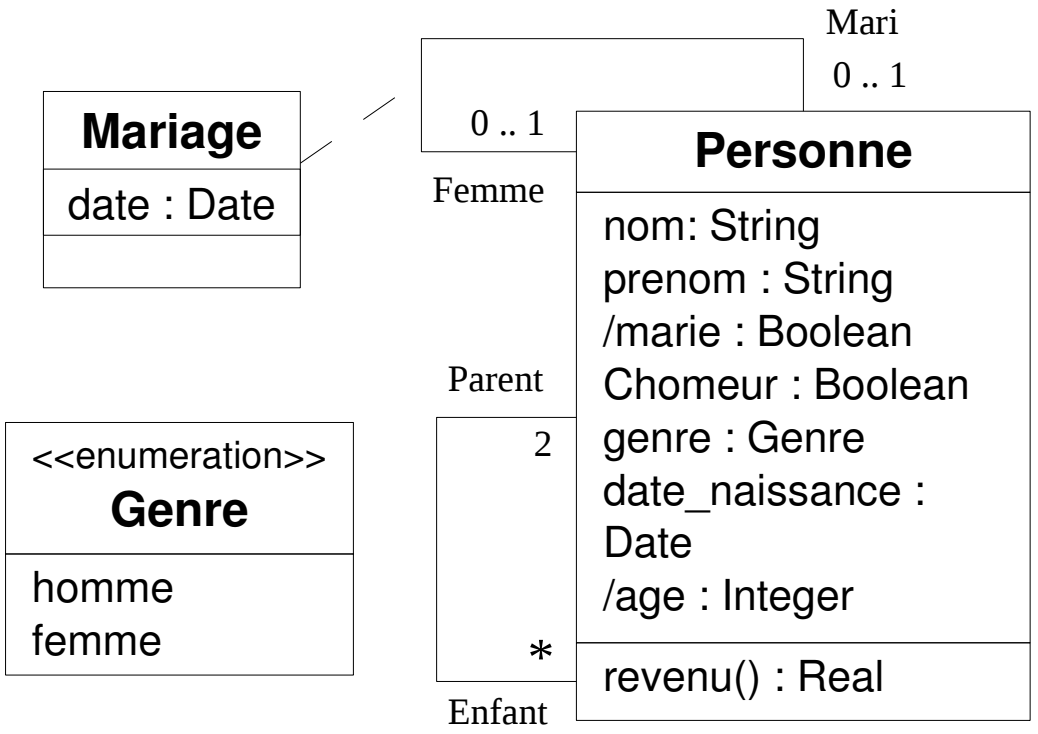
OCL

Contraintes




```
context Personne
inv : let revenus : Real = self.poste.salaire->sum() in
    if chômeur then
      revenus < 100
    else
      revenus >= 100
    endif
```

OCL



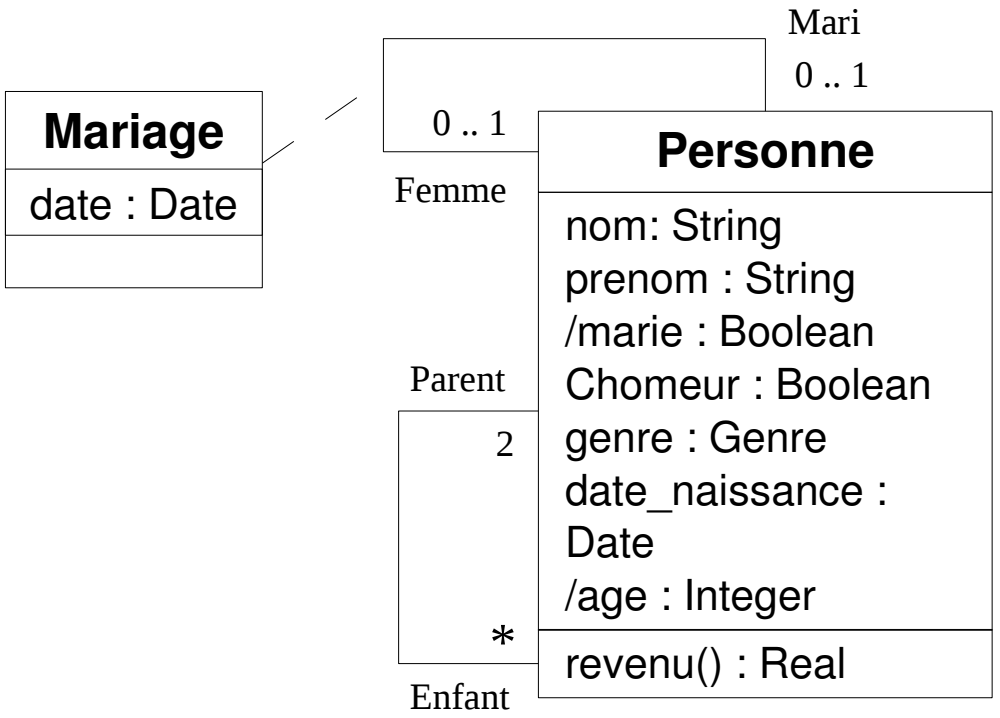
Contraintes

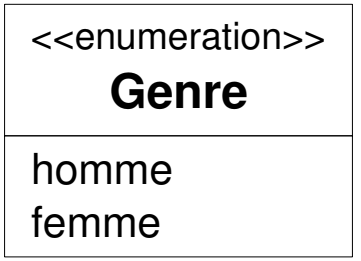
```
context Personne
inv : parent->size()<=2
```

```
context Personne
inv : parent->size()=2 implies
    ( parent->exists(genre=Genre::homme) and
      parent->exists(genre=Genre::femme) )
```

OCL

Contraintes

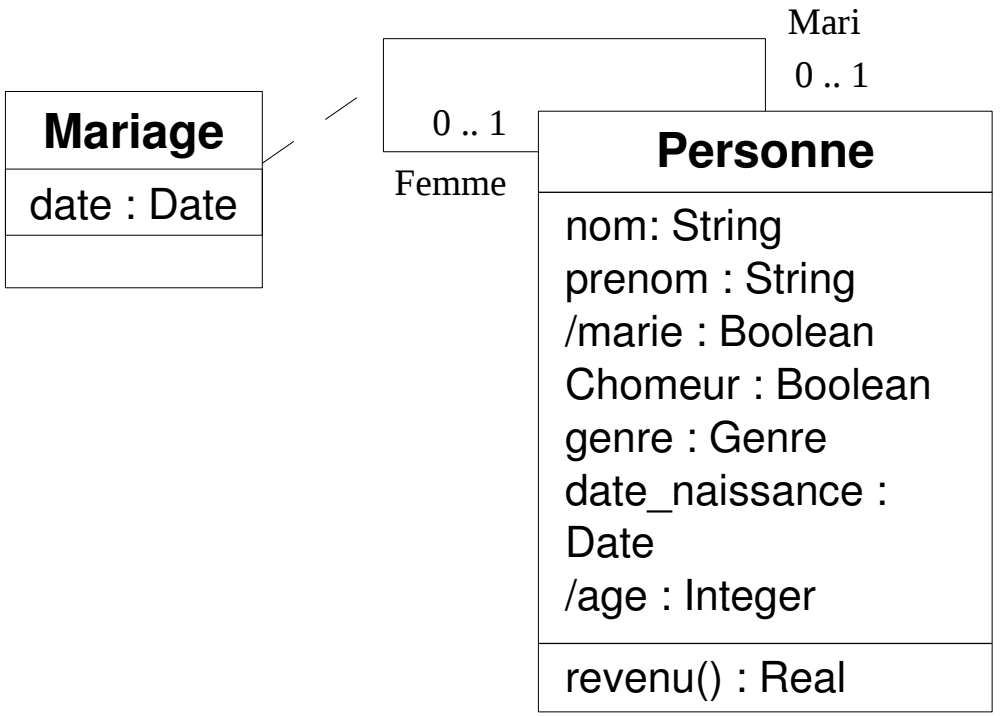





```
context Personne
inv : enfant->notEmpty() implies
   enfant->forAll( p : Personne | p.parents->includes(self))

context Personne
inv : parent->notEmpty() implies
   parent->forAll ( p : Personne | p.enfant->includes (self))
```

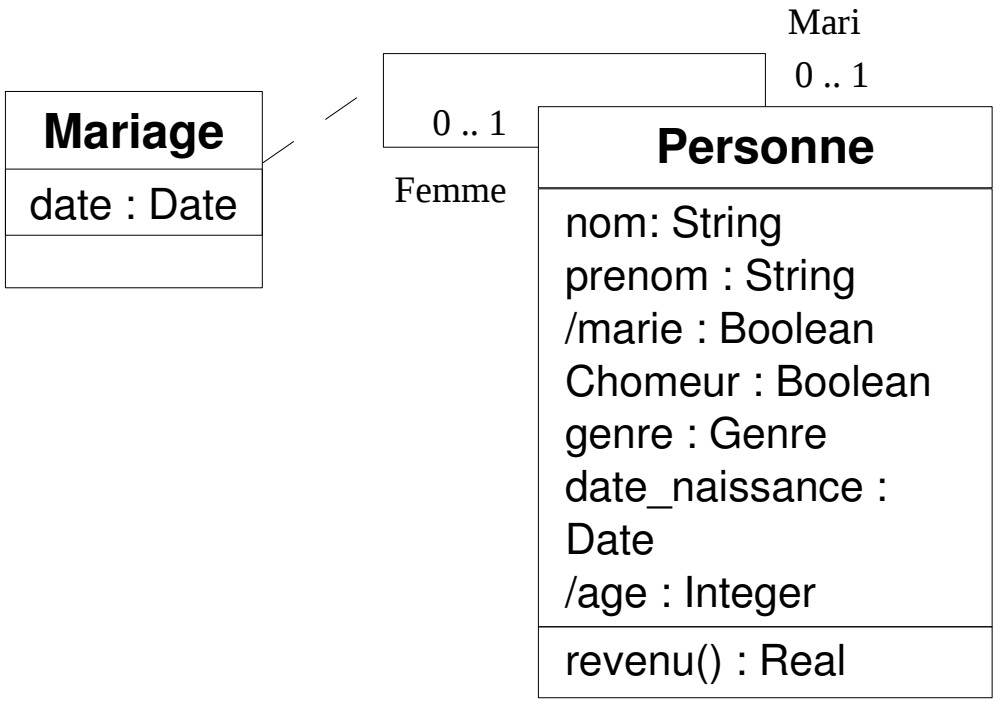
Contraintes

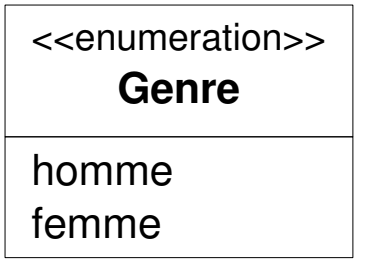



```
context Personne::marié
derive : self.femme->notEmpty() or self.mari->notEmpty()
```

OCL

Contraintes

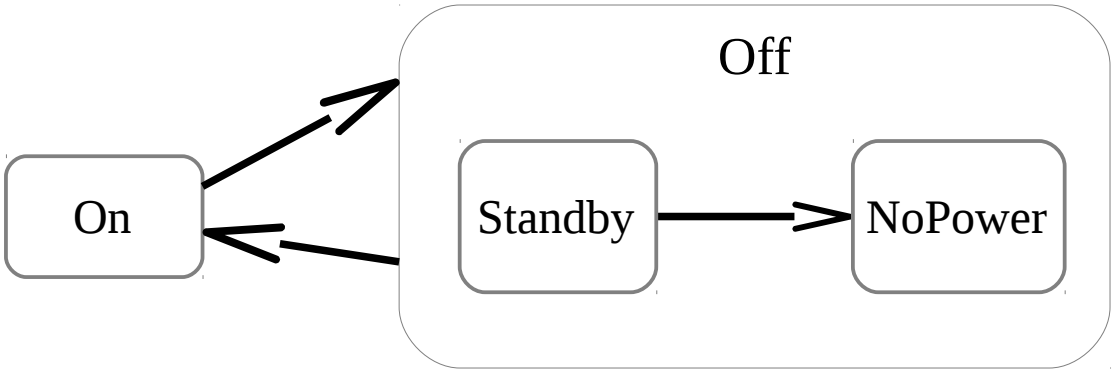





```
context Personne
inv : self.marié implies
    self.genre = Genre::homme implies (
      self.femme->size()=1 and
      self.femme.genre = Genre::femme)
    and self.genre = Genre::femme implies (
      self.mari->size()=1 and
      self.mari.genre = Genre::homme)
    and self.age >=18
```

OCL

Contraintes




```
object.oclInState(On)
Object.oclInState(Off)
Object.oclInState(Off::Standby)
Object.oclInState(Off::NoPower)
```

Conclusion

Langage formel avec grammaire élémentaire

Evite les ambiguïtés du langage naturel

Exploité dans quelques outils de génération de code

Références

OMG http://www.omg.org/spec/OCL