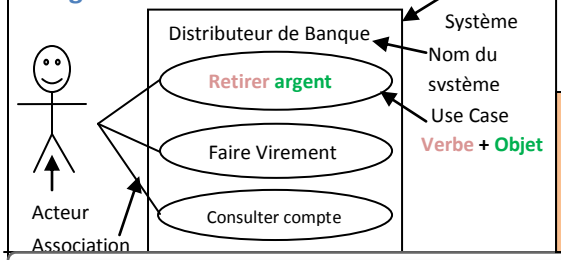
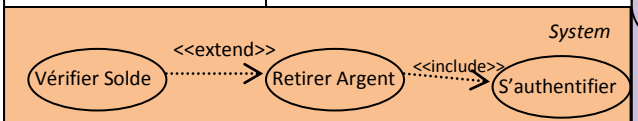
Diagramme de cas d'utilisation



Extend : Un cas A étend un cas B lorsque que A peut être appelé au cours de l'exécution de B

Include : Un cas A inclut un cas B si A inclut systématiquement B dans son comportement: Utile pr décomposer un cas complexe en sous-cas simple



Point d'extension : Lorsque l'extension intervient lors d'un cas précis

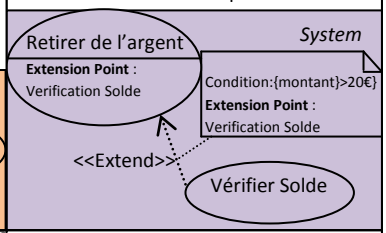
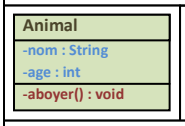


Diagramme de Classe

Représentation schématique



Element Sup : **Nom** de la classe
Element Central : **Propriétés** de classe
Element Inf : **Méthode(s)** de classe

1..n Owned By 1 — **Association** : Relie deux classes entres elles avec éventuellement des **Cardinalités** et le type **de relation**

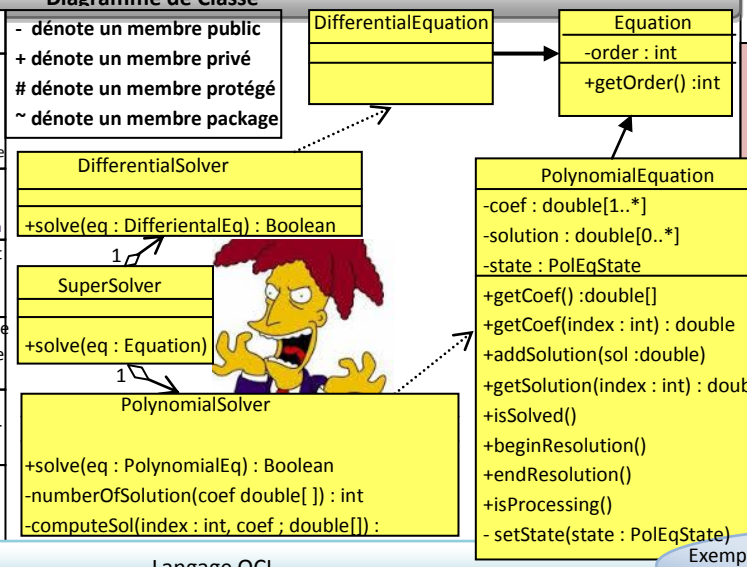
Composition : Ou une classe ne peut exister par elle-même mais doit être membre d'une autre classe

B — A — **Dépendance** : Quand **une classe** en utilise **une autre** comme membre ou paramètre il y a relation de dépendance

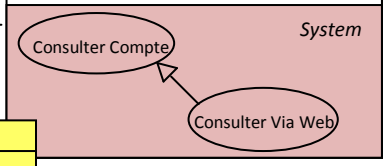
Agrégations : indiquent une relation de contenant/contenu. Elle décrite par une relation "peut possèder".

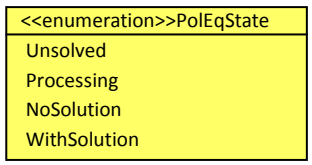
B — A — Généralisation ou Héritage : **Classe** qui possède en + toutes les propriétés et méthode(s) d'une **classe parente**

- dénote un membre public
+ dénote un membre privé
# dénote un membre protégé
~ dénote un membre package



Generalize : A est une généralisation de B lorsque A est un cas particulier de B





context PolynomialEquation
inv : coef->size() = ordre +1
inv : 0 <= solution ->size() <= ordre
context PolynomialEquation::state
init : PolEqState ::Unsolved

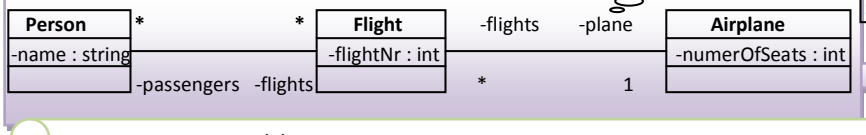
Langage OCL

Exemple : **Context** *Compte::débiter* **(montant : Integer)**
pre : (montant > 0) **and** (montant<5000)

Définition : **Object Constraint Langage** Ensemble de contrainte dans un UML
- Contrainte : Expression Booléenne True or False Syntaxte: {contrainte}
- types de contraintes : **Invariant, Pré-condition, Post-condition etc.**
- Langage déclaratif : On ne peut faire que des requête, on ne décrit pas le comportement à adopter si une contrainte n'est pas respecté
- Langage Sans effet de bord: Les instances ne sont pas modifiés par les contraintes
- Deux manière d'écrire une contrainte OCL: entre accolade dans une note ou avec le mot clé "Context" dans un fichier séparé.

Utilisation :
- Description d'invariance sur les classes et les types de variables
- Pré & Post-conditions sur les opérations
- Règles de dérivation des attributs

Nom de l'élément utilisé en OCL + Cardinalités



Elément : Classe, attribut, opération ...
pour faire référence à un élément d'une classe utiliser les "::" ex :
context compte ← *La classe compte*
context compte::getSolde() ← *La méthode getSolde de la classe compte*
Stéréotype : inv (invariance), pre(pré-conditions), post(post-condition)
Contrainte : 3 notations différentes,

Nommage par défaut : mot-clé *self*
`context Person`
`inv : self.age >= 18`

Nommage omit :
`context Person`
`inv : age >= 18`

Nommage explicit :
`context p : Person`
`inv : p.age >= 18`

Accéder aux propriétés d'une classe : utilisation du '.' Pour naviguer
- à un attribut : self.flightNr
- à une opération : self.availableSeats()
- à l'autre côté de l'association : self.plane

Stéréotypage :
pré-condition : doit être vraie avant l'appel de l'opération
post-condition : doit être vraie après l'appel de l'opération
Pour post-condition on peut utiliser 'result' (indique la valeur retournée par l'opération) et '@pre' (indique la valeur d'un attribut avant l'appel de l'opération)
context Compte ::Debiter(montant :integer)
pre : montant >0
post : solde = solde@pre – montant
context Compte ::getSolde() :Integer
post : **result** = solde

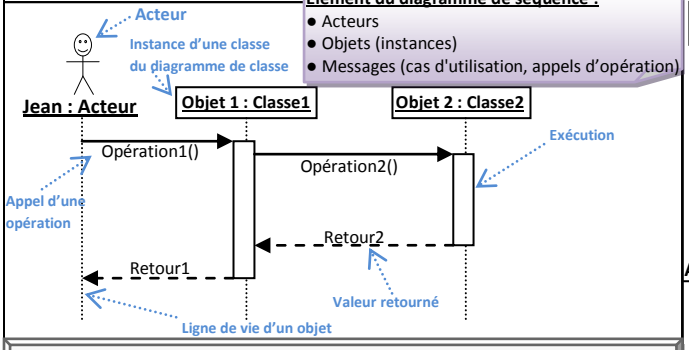
Body, Ce type de contrainte permet de définir directement le résultat d'une opération. le résultat de l'appel de l'opération *getSolde* doit être égal à l'attribut *solde*
➔ **context** Compte::getSolde() : Real
body : solde
Def et let..in sont des types de contraintes qui permettent de déclarer et de définir la valeur d'attributs. Permet également de déclarer et de définir la valeur retournée par une opération interne à la contrainte.
context Personne
inv : let argent=compte.solde->sum() **in** age>=18 implies argent>0

context Personne
def : argent : int = compte.solde->sum()
context Personne
inv : age>=18 implies argent>0

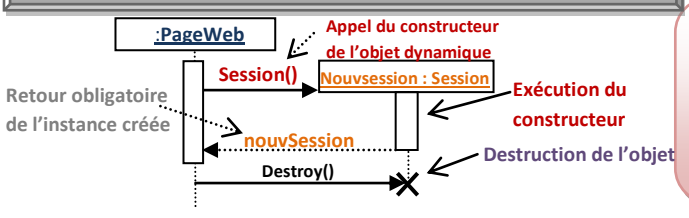
Init permet de préciser la valeur initiale d'un attribut
context Personne::marié : Boolean
init : false
La contrainte **Derive** impose une contrainte perpétuelle.
context Personne::age : Integer
derive : date_de_naissance - Date::current()

Définition: Les diagrammes de séquence montrent les interactions entre objets selon un point de vue temporel. Représente les interactions entre objets en insistant sur la chronologie des envois de messages.

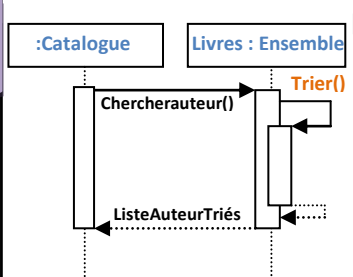
Elément du diagramme de séquence :
- Acteurs
- Objets (instances)
- Messages (cas d'utilisation, appels d'opération)



Création/Destruction d'objet



Message réflexif



Appel d'une opération interne : trier()
Signifie que l'opérateur éxécute l'opération ssi la Condition de Garde (CG) est vraie

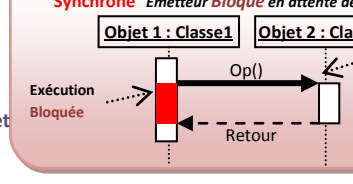
Opérateur d'interaction

Exemple d'opérateur d'interaction au verso

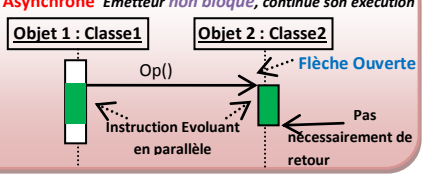
Alt [a][b][c]...	Exécute **soit a soit b soit c** ....
Opt	Exécute une opération SSI la **CG** est vrai
break	Exécute l'opé X et ignore ts les X+n
Loop(min,max)	Pour i de min à max faire l'opération
Par	**X sous parties s'exécute simultanément**
critical	Coupe toute exécution parralèle
ignore	ignore la sous partie
consider	
assert ≠	Seul ses séquences sont valides
neg ≠	Représente une séquence invalide
seq	Strict en beaucoup plus compliqué
Strict	**X sous parties s'exécute de haut en bas**

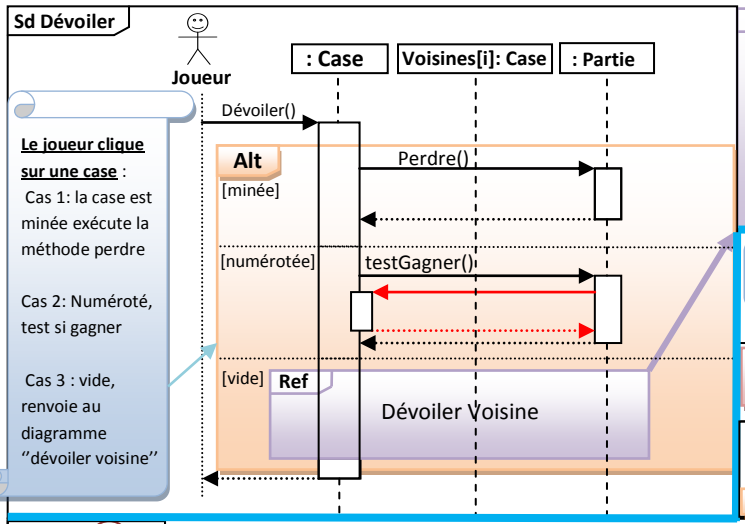
Type de message

Synchrone *Emetteur Bloqué en attente de retour*



Asynchrone *Emetteur non bloqué, continue son exécution*





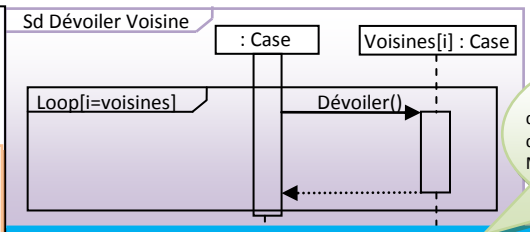
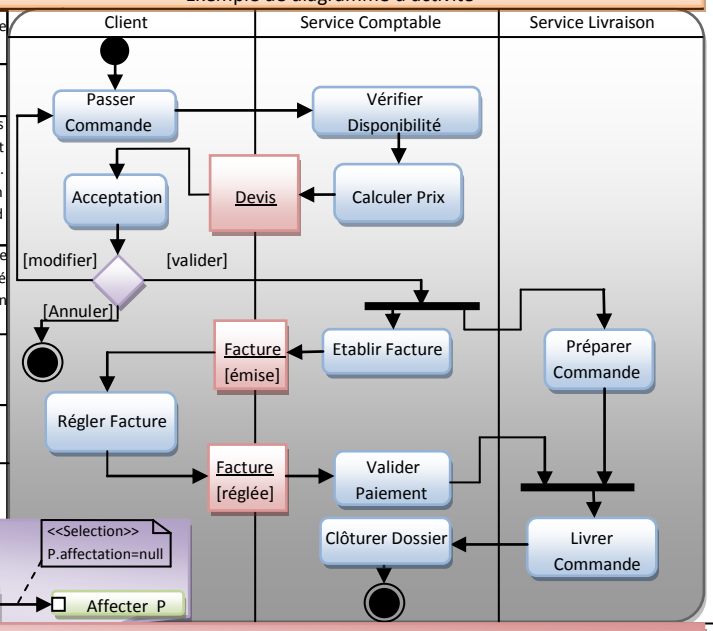


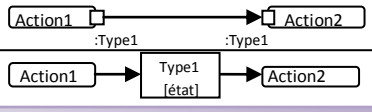
Diagramme d'activité :
Décrit le déroulement d'un cas d'utilisation ; Décrit le comportement d'une méthode dans une classe
Modéliser le cheminement de flots de contrôle et de flots de données

Nœud d'action	**Un nœud d'action** est un nœud d'activité exécutable qui constitue l'unité fondamentale de fonctionnalité exécutable dans une activité. L'exécution d'une action représente une transformation ou un calcul quelconque dans le système modélisé. Les actions sont généralement liées à des opérations qui sont directement invoquées. Un nœud d'action doit avoir au moins un arc entrant.
Nœud d'objet [état]{contrainte}	**Un nœud d'objet** permet de définir un flot d'objet (*i.e.* un flot de données) dans un diagramme d'activités. Ce nœud représente l'existence d'un objet généré par une action dans une activité et utilisé par d'autres actions
	Un nœud initial est un nœud de contrôle à partir duquel le flot débute lorsque l'activité enveloppante est invoquée. Une activité peut avoir plusieurs nœuds initiaux. Un nœud initial possède un arc sortant et pas d'arc entrant.

Exemple de diagramme d'activité



	Lorsqu'un flot d'exécution atteint un **nœud de fin de flot**, le flot en question est terminé, mais cette fin de flot n'a aucune incidence sur les autres flots actifs de l'activité enveloppante.
	Transition Elles sont déclenchées dès que l'activité source est terminée et provoquent automatiquement et immédiatement le début de la prochaine activité à déclencher (l'activité cible).
Décision [else] [condition]	**Un nœud de décision** est un nœud de contrôle qui permet de faire un choix entre plusieurs flots sortants. Il possède un arc entrant et plusieurs arcs sortants. L'utilisation d'une garde `[else]` est recommandée après un nœud de décision car elle garantit un modèle bien formé. **Un nœud de fusion** est un nœud de contrôle qui rassemble plusieurs flots alternatifs entrants en un seul flot sortant. Il n'est pas utilisé pour synchroniser des flots concurrents (c'est le rôle du nœud d'union) mais pour accepter un flot parmi plusieurs.
	Lorsque l'un des arcs d'un **nœud de fin** d'activité est activé, l'exécution de l'activité enveloppante s'achève et tout nœud ou flot actif au sein de l'activité enveloppante est abandonné. Si l'activité a été invoquée par un appel synchrone, un message (*reply*) contenant les valeurs sortantes est transmis en retour à l'appelant.
Bifurcation	**Un nœud de bifurcation**, également appelé nœud de débranchement est un nœud de contrôle qui sépare un flot en plusieurs flots concurrents (exécution en parallèle). **Un nœud d'union**, ou nœud de jointure est un nœud de contrôle qui synchronise des flots multiples
e :int s:int f :int	**Pins** : (entrée /Sortie) Utilisé pour supprimer les nœuds d'objets



Un nœud de Stockage : Est un nœud tampon central particulier qui assure la persistance des données :
Les données de sortie sont dupliquées et les données originales restent dans le nœud de stockage
Si une donnée d'entrée contient le même objet déjà dans le nœud de stockage, l'objet est écrasé

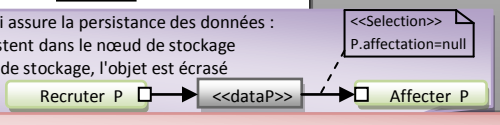


Diagramme d'Etat-transition (State-Machine)

Un diagramme d'Etat Transition rassemble et organise les Etat d'une classe données. Un diagramme d'Etat ne peut être associé qu'à une seule classe

Etat Simple
Transitions Internes
: Etat d'un objet durant sa durée de vie en attente d'un évènement.

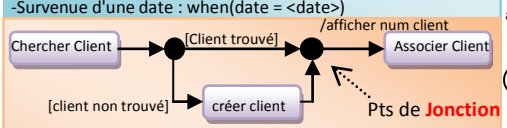
Etat Initial : Pseudo état qui indique l'état de départ lorsque que l'Etat Enveloppant ou L'objet est invoqué/créé.

Etat Final : Pseudo état qui indique que le diagramme d'état-transition ou l'état enveloppant est terminé.

Etat Composite
Etat d'un objet enveloppant des sous-états. Par définition un diagramme d'état est un état composite.

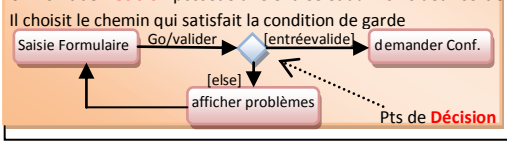
Evénement

Quand un événement est reçu, une transition peut être déclenchée et faire basculer l'objet dans un nouvel état.
Types d'événement
Evénement d'appel (call) :
-Réception de l'appel d'une opération par un objet
-Méthodes déclarées au diagramme de classes
Evénement de type signal :
-La réception d'un signal envoyé par un objet
-Communication asynchrone à sens unique entre deux objets
Evénement de changement :
-Changement de valeur de faux à vrai d'une expression - Booléenne : when (expression).
Evénement temporel :
-Epuisement d'un délai (timeout) : after (durée).
-Survenue d'une date : when(date = <date>)

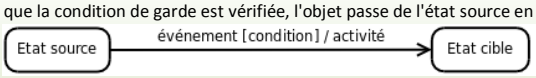


Un Point de **Jonction** Simplifie le schéma et le rend plus lisible

Un Point de **Décision** possède une entrée et au moins deux sortie Il choisit le chemin qui satisfait la condition de garde



Transition
Une transition représente pour l'objet, le passage instantané d'un état à un autre état.
Une transition se produit suite à un évènement
Une transition peut être conditionnée à l'aide d'une garde : expression booléenne exprimée en langage naturel
Si un événement déclencheur se produit et que la condition de garde est vérifiée, l'objet passe de l'état source en état cible et exécute une activité (effet de transition)

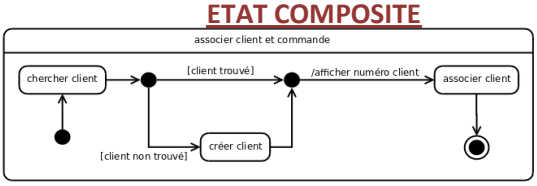


Transition Interne

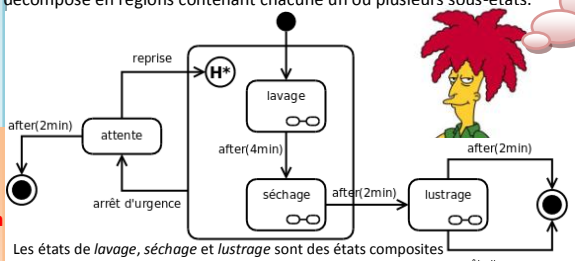
Saisie mot de passe
entry /ne plus afficher les entrées
on aide / afficher l'aide
exit / réafficher les entrées
do / gérer la saisie de l'utilisateur

entry – *entry* permet de spécifier une activité qui s'accomplit quand on entre dans l'état.
exit – *exit* permet de spécifier une activité qui s'accomplit quand on sort de l'état.
do – l'activité commence dès que l'activité *entry* est terminée. Lorsque cette activité est terminée, une transition d'achèvement peut être déclenchée. Si une transition se déclenche pendant que l'activité *do* est en cours, cette dernière est interrompue et l'activité *exit* s'exécute.
include – permet d'invoquer un sous-diagramme d'états-transitions.

ETAT COMPOSITE



Un état simple ne possède pas de sous-structure mais uniquement, le cas échéant, un jeu de transitions internes. Un état composite est un état décomposé en régions contenant chacune un ou plusieurs sous-états.



Les états de *lavage*, *séchage* et *lustrage* sont des états composites définis sur trois autres diagrammes d'états. En phase de lavage ou de séchage, le client peut appuyer sur le bouton d'arrêt d'urgence. S'il appuie sur ce bouton, la machine se met en attente. Il a alors deux minutes pour reprendre le lavage ou le lustrage, exactement où le programme à été interrompue, c'est à dire au niveau du dernier sous-état actif des états de lavage ou de lustrage (état historique profond). Si l'état avait été un état historique plat, c'est toute la séquence de lavage ou de lustrage qui aurait recommencée. En phase de lustrage, le client peut aussi interrompre la machine. Mais dans ce cas, la machine s'arrêtera définitivement.

Un **état historique plat**, est un pseudo-état qui mémorise le dernier sous-état actif d'un état composite. Une transition ayant pour cible l'état historique est équivalente à une transition qui a pour cible le dernier état visité de l'état englobant.
Un ***état historique profond*** permet d'atteindre le dernier état visité dans la région, quel que soit sont niveau d'imbrication, alors que le l'état historique plat limite l'accès aux états de son niveau d'imbrication.

Ce que disent les ingénieurs en informatique... et ce qu'il faut comprendre:
- "Nous allons inscrire ce projet au planning": On s'en occupera si on a rien d'autre à faire
- "C'est un programme complètement nouveau!": C'est pas du tout compatible avec l'ancienne version
- "Ce programme ne nécessite aucune maintenance": C'est impossible à déboguer
- "Ce programme ne nécessite que peu de maintenance": C'est quasiment impossible à déboguer
- "Nous respecterons les standards": On a toujours fait comme ça et ce n'est pas aujourd'hui qu'on va changer
- "Nous tenons à respecter les standards": Vous n'allez pas remettre en cause tout ce qu'on vient de faire
- "La nouvelle version de ce programme est 100% compatible avec la précédente": On n'a touché à rien
- "Différentes approches ont été tentées": On essaie encore de deviner ce qui se passe.
- "On approche d'une solution": On s'est réunis pour prendre un café...