Cours STIM P8 – TD 1
Génie Logiciel

Compléments sur UML

Intervenant : Anil CASSAM CHENAI

Date : 02/02/2012

Objectifs du complément

- Ce complément sera approfondi en parallèle de plusieurs TD/Cours.
- Rappels sur UML dans un cadre de gestion de projet, plutôt que dans une simple vision de modèle
- Apporter des compléments à la gestion de projet, plus adapté à un projet de développement logiciel.
- On utilise ici à un outils de modélisation UML gratuit et open-source (ici StarUML) pour illustrer le cours, mais on pourra en utiliser d'autres plus récents.
- On introduira aussi les architecture MDA qui permettent de la génération de code basé sur des modèles et qui est de plus en plus d'actualité

Documents de référence

- [1] UML Superstructure Specification V2 – OMG (object management group)
- [2] Cours UML de Rational Software (IBM)
- [3] Livre de Craig Larman : Applying UML and Patterns : an introduction to Object-Oriented Analysis

Organisation des exercices

- Présentation des concepts
- Réalisation immédiate d'un exercice d'application du concept
- A la fin d'un chapitre réalisation d'un exercice plus global mettant en jeux différents concepts et leurs articulations

Historique

- Année 70 et 80 : guerre des méthodologies
- Années 90 : unification des trois principales méthodologies sous UML 1.0 – 1.1
  - OMT (Object Modeling Technique)
  - Booch
  - OOSE (Object Oriented Sofware Engineering)
- XXX : sortie de UML 2.0
- Années 2000 : développement des méthodologies MDA

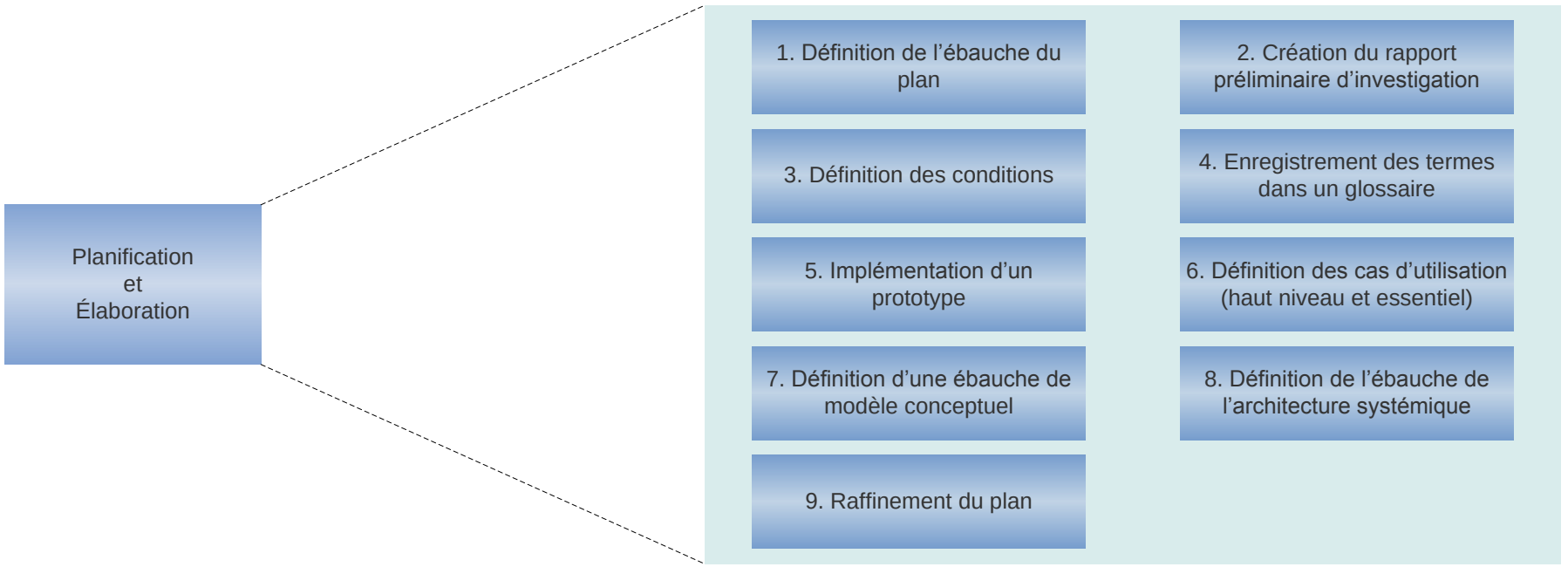
UML et Gestion de projet (1)

- Rappel des phases de gestion de projet
  - Phase d'administration
    - Analyse du CDC/Rédaction de spécification générale
    - Evaluation/Organisation/Planification
  - Phase de réalisation/pilotage
  - Phase de terminaison
- On peut utiliser UML très tôt pour structurer la phase d'administration :
  - Afin de dégrossir le projet en sous éléments
  - Les bons outils sont alors :
    - Les diagrammes de cas d'utilisation de haut niveau
    - Les diagrammes de classes utilisant un vue de niveau package par exemple

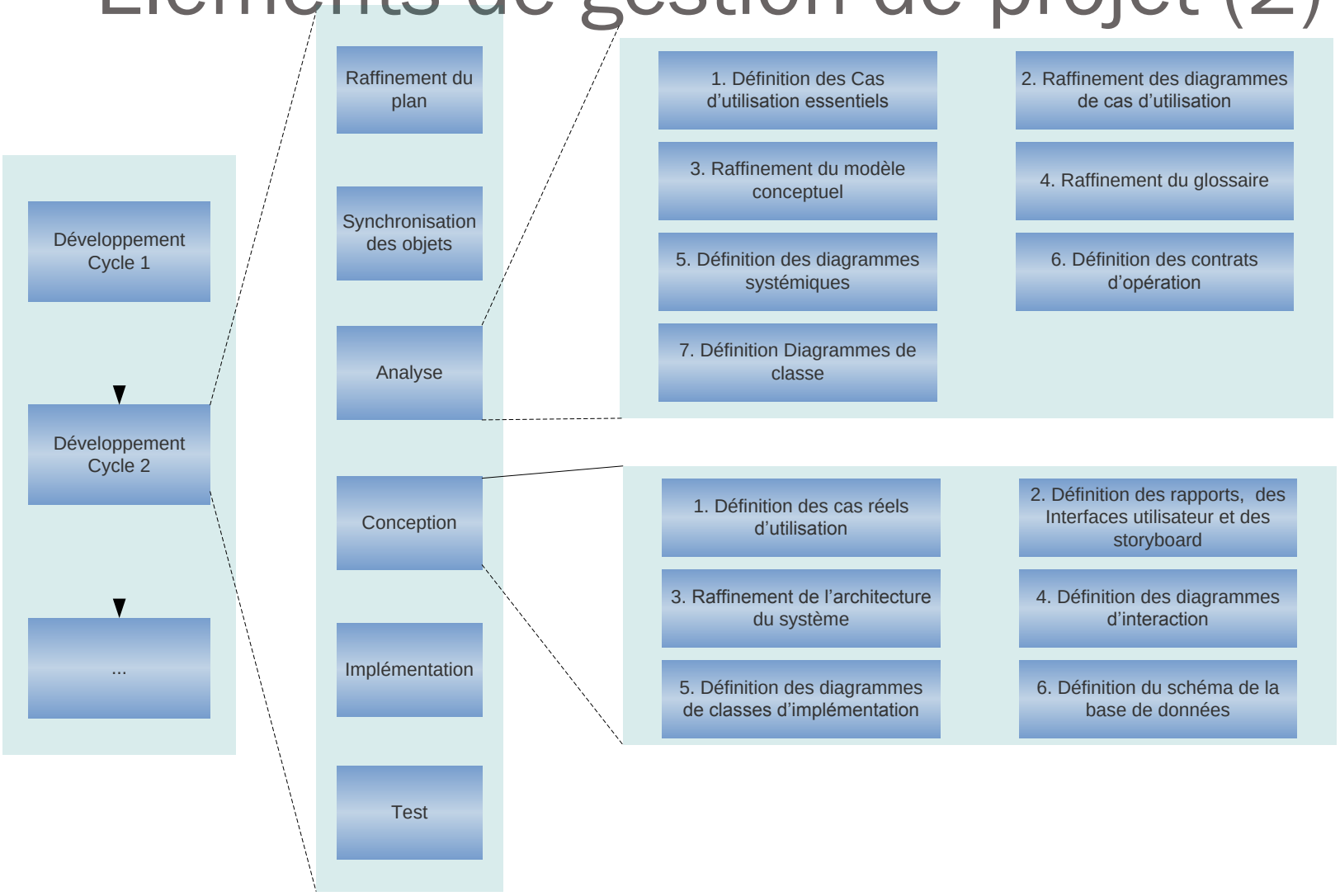
UML et Gestion de projet (2)

- Ensuite on affinera/complètera les diagrammes :
  - Dans la phase de réalisation : un première partie de la phase de réalisation correspondra à la conception UML avant de commencer le codage proprement dit. On peut avoir quelque chose comme :
    - Compléter le modèle UML
    - Générer (cas MDA) et écrire le code
    - Tester le code
    - Documenter
  - Mais aussi plus globalement lors de cycles complets Projets/Sous Projets

Éléments de gestion de projet (1) :



Éléments de gestion de projet (2)



Éléments de gestion de projet (3)

- Développement itératif
  - Affiner le modèle sur différents cycles
- « Time-boxing » des cycles
  - Entre 2 semaines à 2 mois
- L'ordre de création des artefacts (Use Case, etc n'est pas forcément linéaire et peut être fait en parallèle)

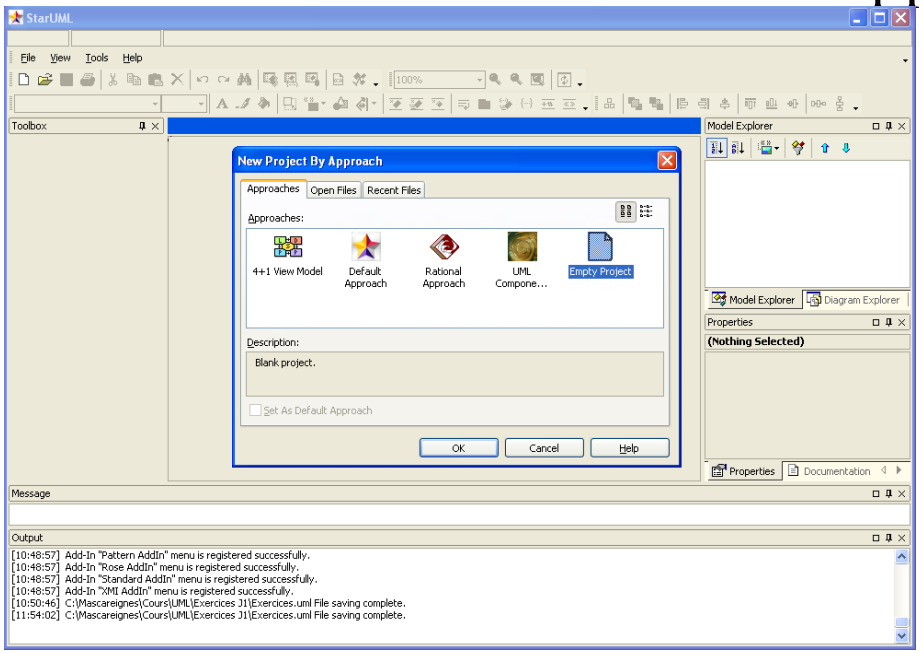
Exigences/Conception/Implémentation

Durant tout le projet, il faut séparer les **exigences** de la **conception** et de **l'implémentation**

Exigences	Conception	Implémentation
Périmètre du problème Ce qui est exigé Contexte de l'Application Hypothèses Besoins de performances	Approche générale Algorithmes Structures de données Architecture Optimisation Planification des ressources	Plates-formes Spécifications matérielles Bibliothèques logicielles Standard d'interface

Démarrer avec StarUML (1)

- Démarrage de StarUML et sélection d'une approche



- Les approches correspondent à une méthodologie d'analyse prédéfinie

Démarrer avec StarUML (2)

- Par exemple l'approche 4+1 :

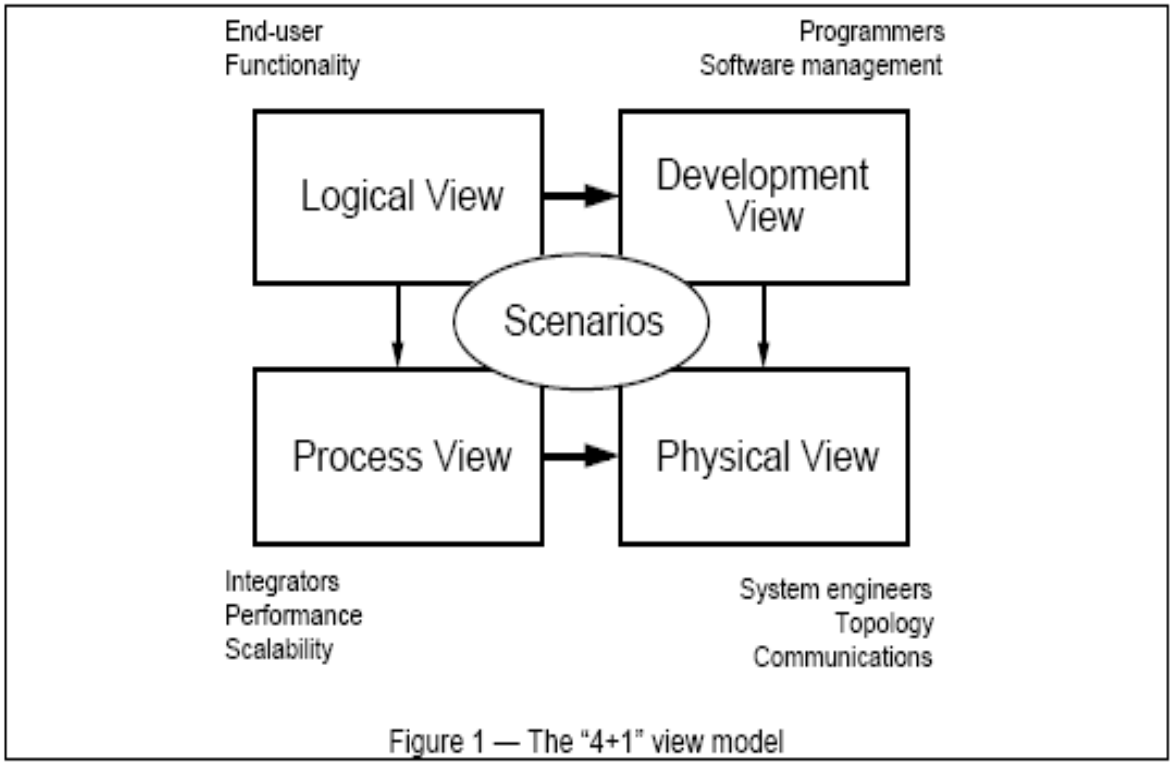
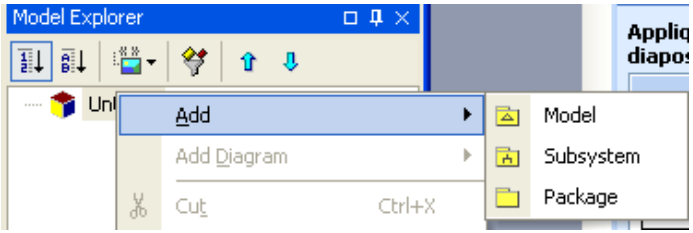
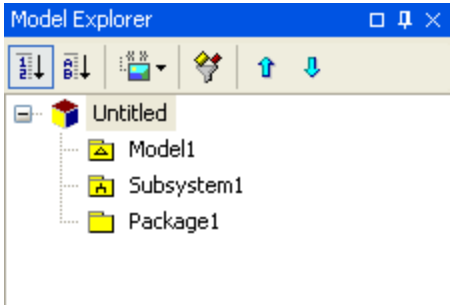


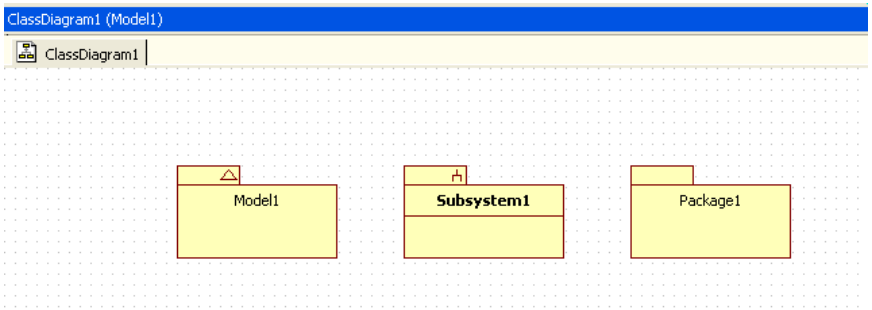
Figure 1 — The "4+1" view model

Démarrer avec StarUML (3)

- On dispose de 3 structures de rangement
  - Les modèles
  - Les sous-systèmes
  - Les packages

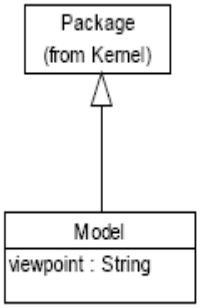






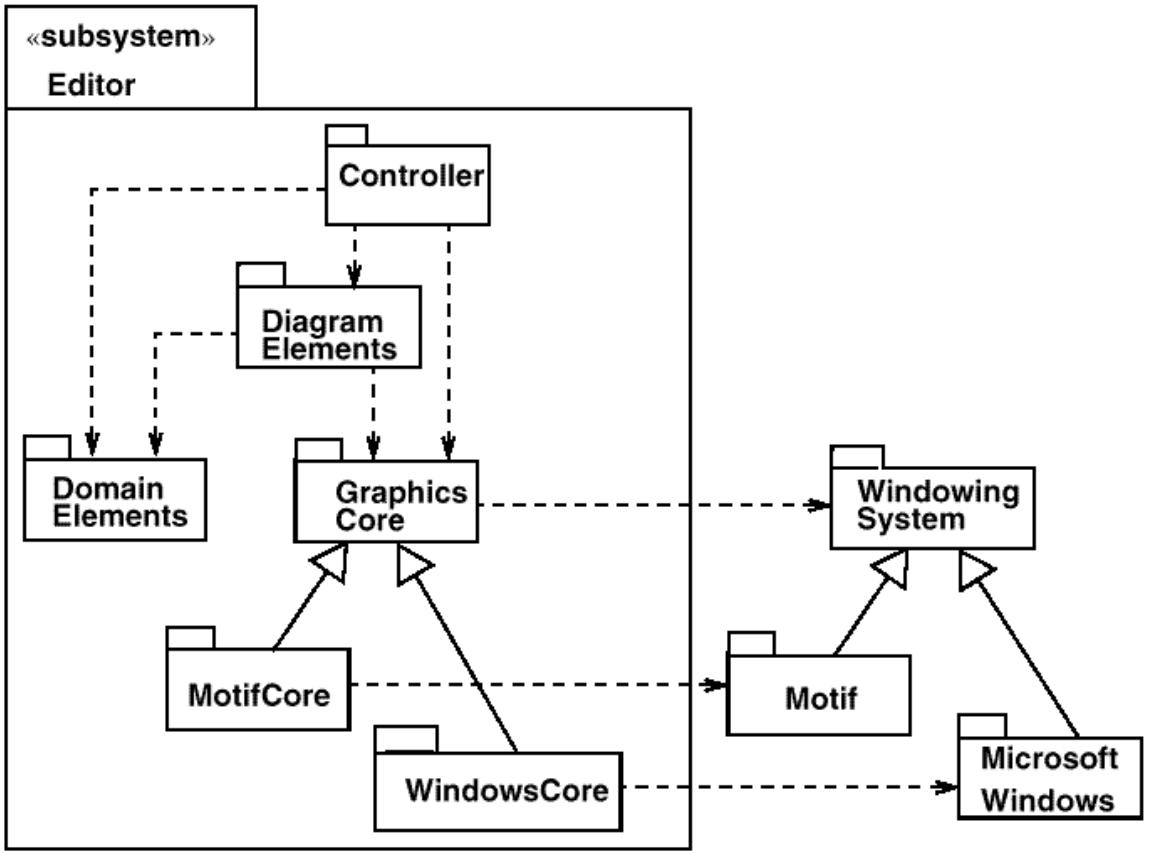
Démarrer avec StarUML (4)

- Les packages sont des dossiers de rangement des objets UML. Ils sont arborescents
- Les modèles sont en fait un type de package particulier



Les différentes dépendances de packages

Figure 4. Packages and their dependencies

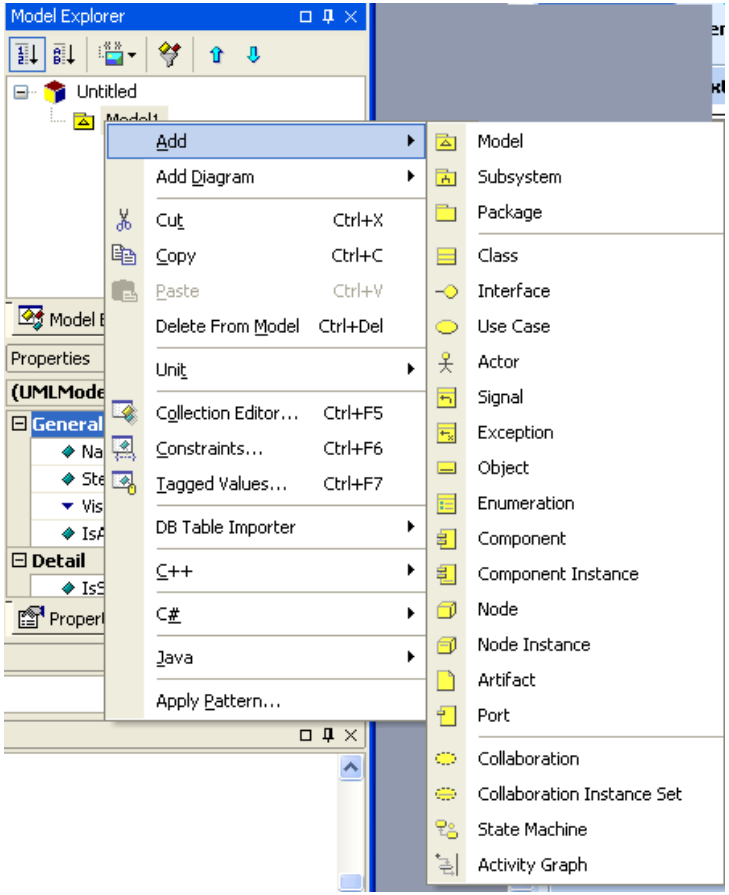


Définition du sous-système

- Un Sous-système est :
  - Ensemble de classes, d'opérations, d'évènements et de contraintes
  - Tous reliés
  - Avec des interfaces bien définies et préférablement restreintes vers l'extérieur
- Pas de trace des sous-systèmes dans UML 2 sauf via un stéréoptype. <<subsystem>>
- Donc Sous-Système=Package avec icône particulière permettant de mieux identifier leur délimitation ⇔ a utiliser le stéréotype subsystem
- Un service est un ensemble de fonctionnalités apparentées qui servent les mêmes buts. On peut utiliser des sous-systèmes pour implémenter des services

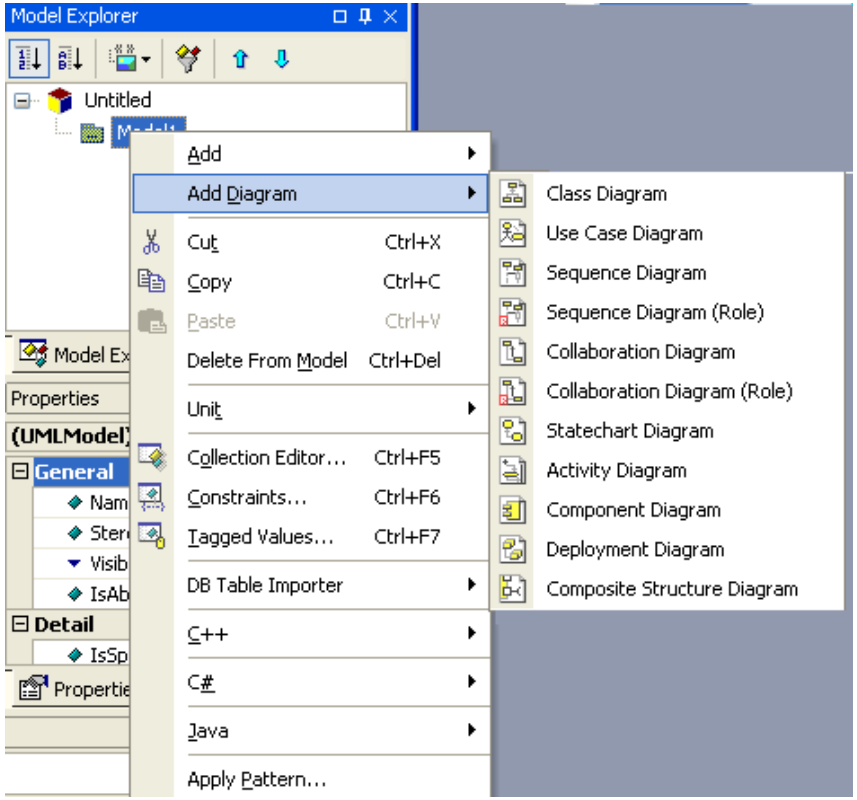
Démarrer avec StarUML (3)

- Création des objets



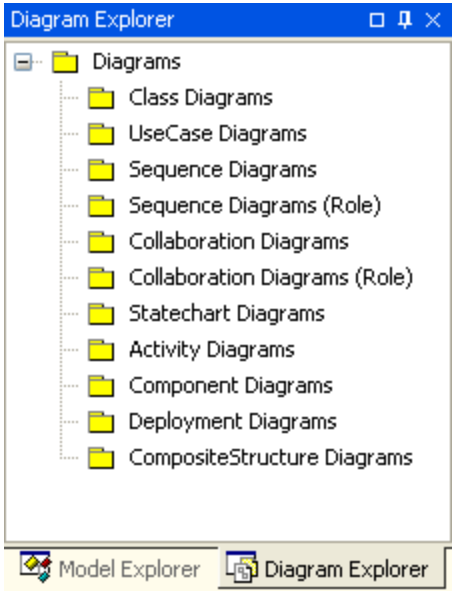
Démarrer avec StarUML (4)

- Création des diagrammes



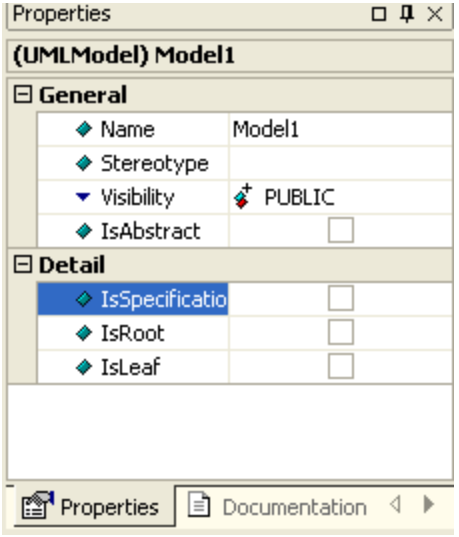
Démarrer avec StarUML (5)

- L'explorateur de diagrammes
  - Permet de voir d'un seul coup d'œil des diagrammes répartis dans l'arborescence des packages rangés par catégories



Démarrer avec StarUML (6)

- L'explorateur de propriétés
  - Permet de modifier les propriétés de l'objet de manière structurée



  - Les diagrammes ou d'autres outils permettent cependant aussi d'agir sur les propriétés des objets

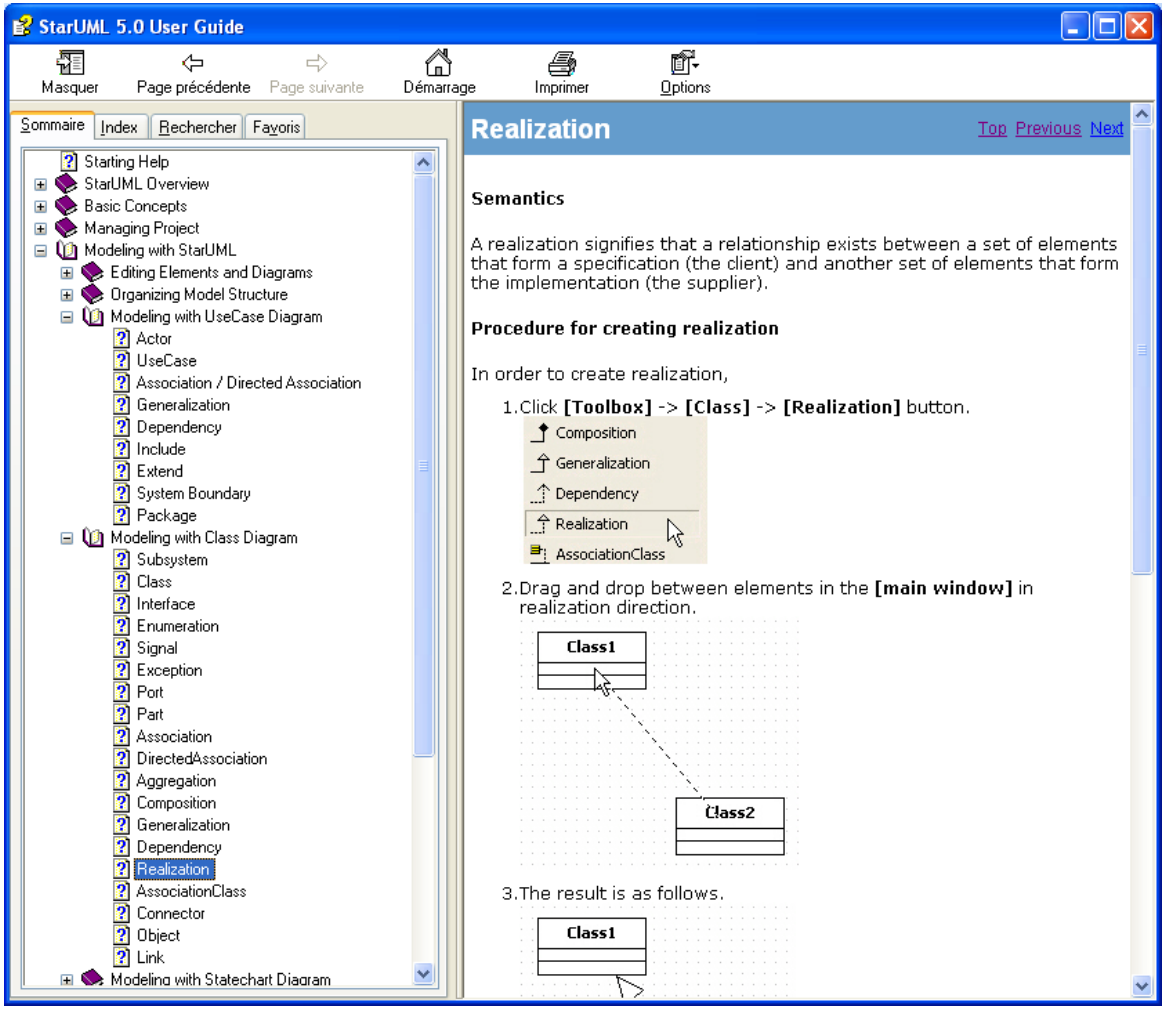
Organisation

- Nous allons :
  - Apprendre a créer et manipuler les différents objets
  - Les étudier dans le cadres des « vues » que sont les diagrammes. Il est important de comprendre qu'un objets et unique dans l'outil de modélisation, mais qu'il peut apparaître dans différentes vues
  - Apprendre les différents moyens d'agir sur les propriétés des objets avec StarUML

Manipulation de l'outil StarUML

- StarUML est un outil très riche est très structuré
- Il est fourni avec un guide utilisateur comprenant des apprentissages détaillées des différentes fonctions de l'outil en fonction du type de diagramme considéré
- Nous utiliserons ce « tutorial » pour structurer nos apprentissages après avoir présenté les concepts

Exemple de « tutorial »



Différentes vision des modèles

- Modèle de classe
  - Structure statique des objets d'un système
- Modèle d'état
  - Décrit les aspects temporels d'un objet
- Modèle d'interaction
  - Décrit la façon dont les objets collaborent pour obtenir des résultats

Mécanismes d'extension

- UML 2.0 contient des mécanismes d'extension
  - Stéréotypes : chaîne qui complètent l'information sur un élément du méta modèle UML
  - Propriétés : caractérisent un élément de modèle
  - Contraintes : spécifient des conditions qui doivent être vrai sur des éléments de modèle
  - Étiquettes (Tag) : paire nom/valeur donnant des informations complémentaires

Analyse des fonctions du système

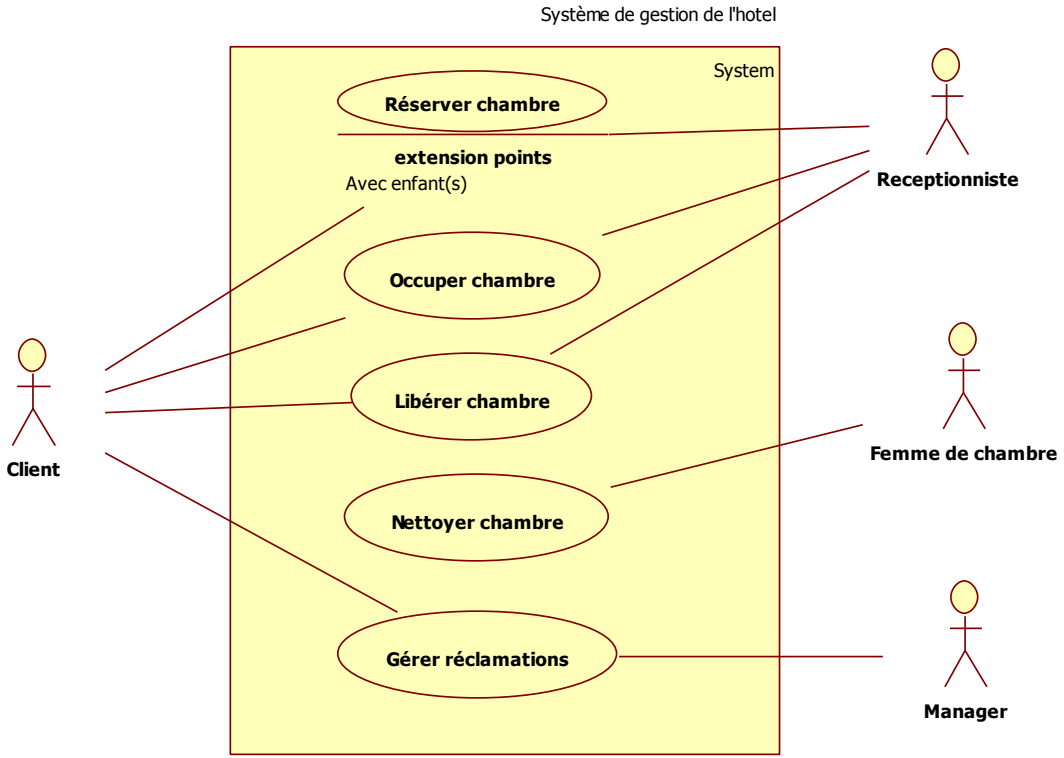
- Même si ce n'est pas de l'UML « traditionnel », l'analyse des fonctions fournies par le système et leur rangement en catégories est un des compléments essentiel de l'analyse en cas d'utilisation que nous allons voir plus loin.
- Dans une première étape, après avoir faire les interviews du client, on :
  - Liste des fonctions en leur donnant un référence
  - On créé des catégories de fonctions
  - On range cela dans un tableau auquel on fera référence dans le reste de l'analyse

Diagrammes de Use Case

- Un cas d'utilisation est une description textuelle d'une fonction que doit réaliser le futur système. Il décrit ce que le système doit faire, mais pas comment implémenter ce comportement souhaité.
- Un diagramme de cas d'utilisation est une représentation visuelle complémentaire de ces « recettes ».
- Il permet de visualiser un système et ses limites, les utilisateurs, les cas d'utilisations autour desquels ils interagissent et les liens (spécialisation, inclusion) entre ces cas d'utilisateur
- C'est un outil de communication avec des non informaticiens

Exemple

- Nous allons réaliser un modèle pour un système informatique de gestion d'un hôtel.



Exemples de cas d'utilisation textuel

- Réserver chambre
  - Le réceptionniste reçoit un appel ou reçoit un client à la réception
  - Il prend le nom, la date, la durée sur séjour et le nombre de personne
  - Il propose un type de chambre disponible et son prix
  - Si le client accepte, il bloque la ou les chambres et le matériel complémentaire (lit bébé)

Deuxième exemple

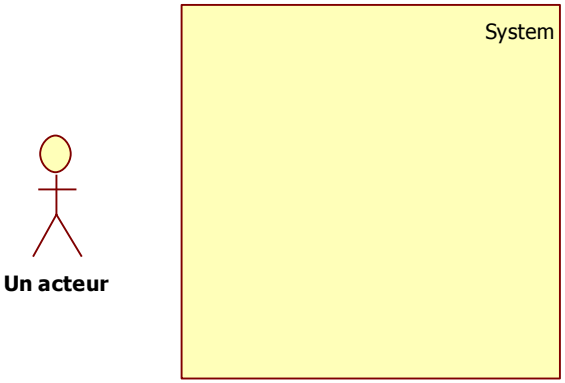
- Occuper chambre :
  - Le client se rend à la réception.
  - Il demande à avoir les clés de sa chambre.
  - Le réceptionniste vérifie :
    - que la chambre a bien été réservée
    - que le nombre de personnes correspond bien
    - que les équipements supplémentaires (lit bébé) sont bien disponibles
  - Si tout se passe bien il donne initialise un badge magnétique, le donne au client avec son numéro de chambre et lui indique le chemin vers celle-ci
  - Le client se rend à sa chambre et vérifie :
    - qu'il peut bien y accéder
    - que la chambre lui convient bien. Dans le cas contraire il passe en phase réclamation

Niveau de détail des Use Case

- Ce cas d'utilisation est un cas d'utilisation de haut niveau. On débute sans surcharger le cas d'utilisation avec des détails
- Lors de cycles de modélisation, on va enrichir les cas d'utilisation et traiter les variantes
- Mettre des références aux fonctions dans la fiche d'un use case.
  - Quand un use case utilise une fonctionnalité, l'indiquer en la mettant en référence dans la fiche du UseCase
  - Un fonctionnalité va donc être implémentée par cycle, en travaillant sur les use cases qui l'utilisent
  - Détecter les fonctionnalités sans cas d'utilisation … ce qui traduit un manque dans les spécifications

Acteurs et système

- Un acteur n'est pas une partie du système. Il correspond à un rôle.



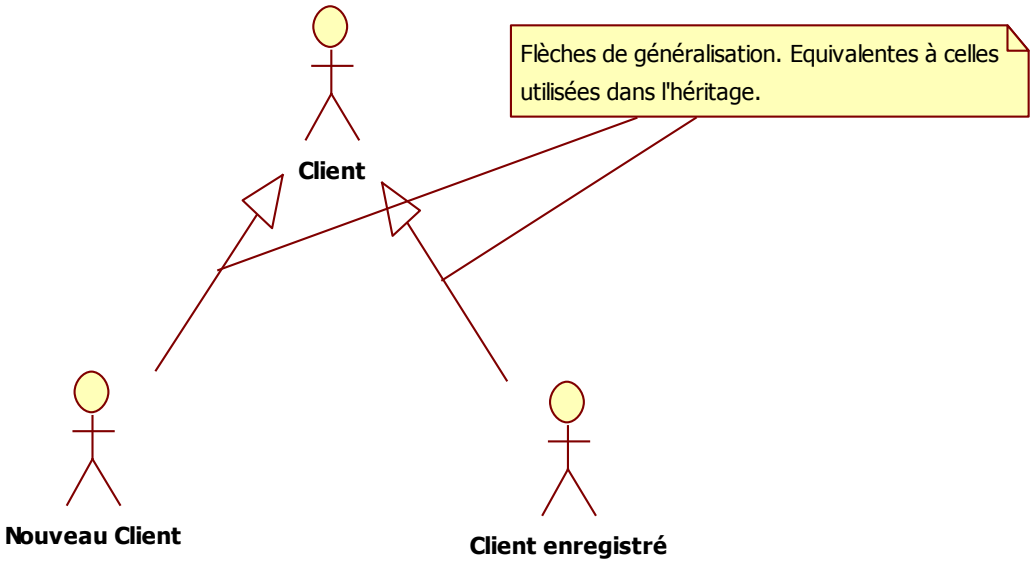
- Un utilisateur peut avoir plusieurs rôles et donc correspondre à un acteur.
- Un acteur peut être un humain,une machine ou un autre système.
- Le système peut être un système informatique mais aussi un système plus vaste (exemple l'entreprise) dans quel cas certains acteurs disparaissent dans le système.

Vision utilisateur du Use Case

- Un cas d'utilisation
  - décrit une interaction d'un (ou de plusieurs) acteur(s) avec le système. Une interaction est une séquence d'actions.
  - doit donner un effet tangible pour l'utilisateur.

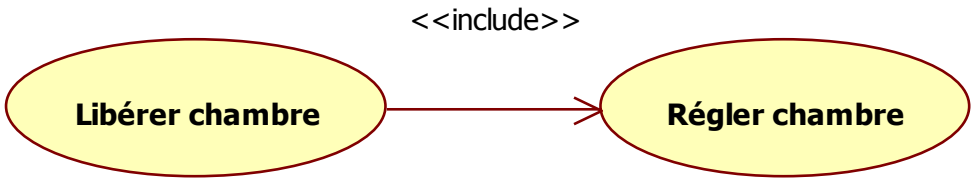
Usage de généralisations

- Permet de clarifier les dépendances entres rôles et packages par exemple (annonce l'héritages des classes)
- Exemple sur les acteurs :



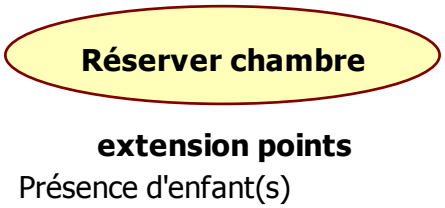
Utilisation et Spécialisation des Use Case

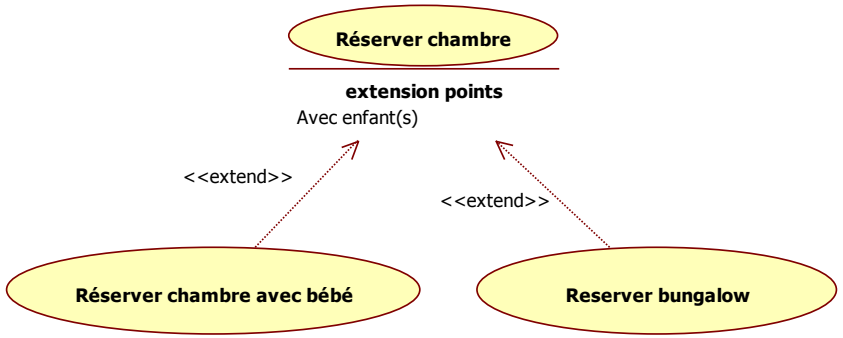
- Un cas d'utilisation peut faire référence à d'autres cas d'utilisation
  - On a vu <<extends>> pour la spécialisation
  - On a <<include>> pour l'utilisation



Points d'extension

- Un point d'extension est une location dans un cas d'utilisation où des séquences d'un autre use-case peuvent être insérés.
- Chaque point d'extension doit avoir un nom unique à l'intérieur du cas d'utilisation.



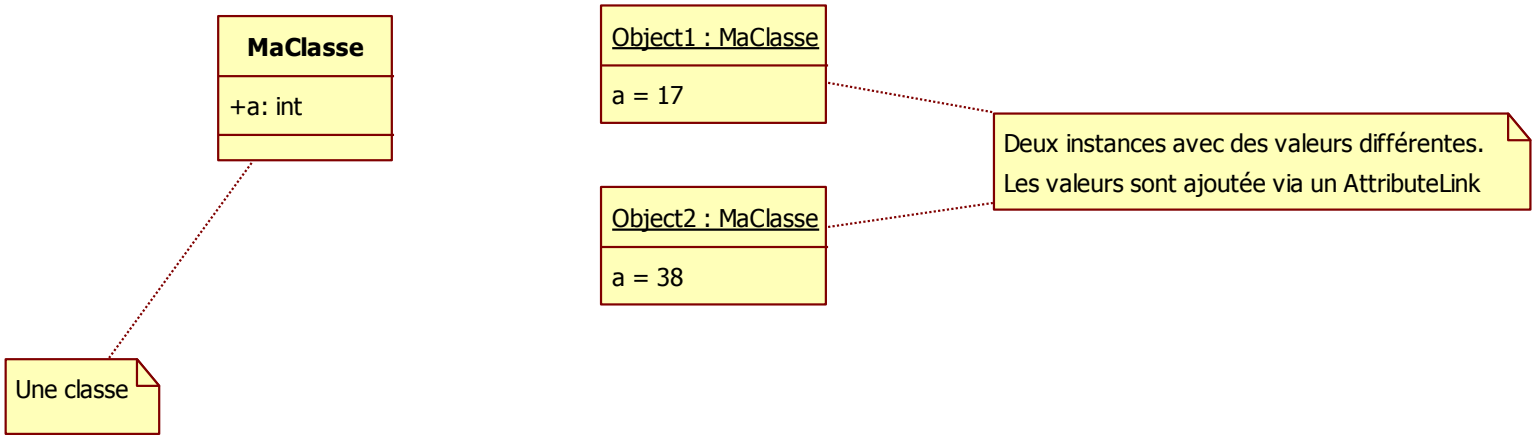


Diagrammes de classes

- Permettent de modéliser
  - Les classes
  - Leurs relations
  - Leurs attributs et opérations
  - Les contraintes
- Ils correspondent à une statique du modèles
- Au début modèle conceptuel pour modéliser les concepts du problème
- Ensuite dans un deuxième temps reprise sous forme de diagrammes de conception (Design)

Classes et objets

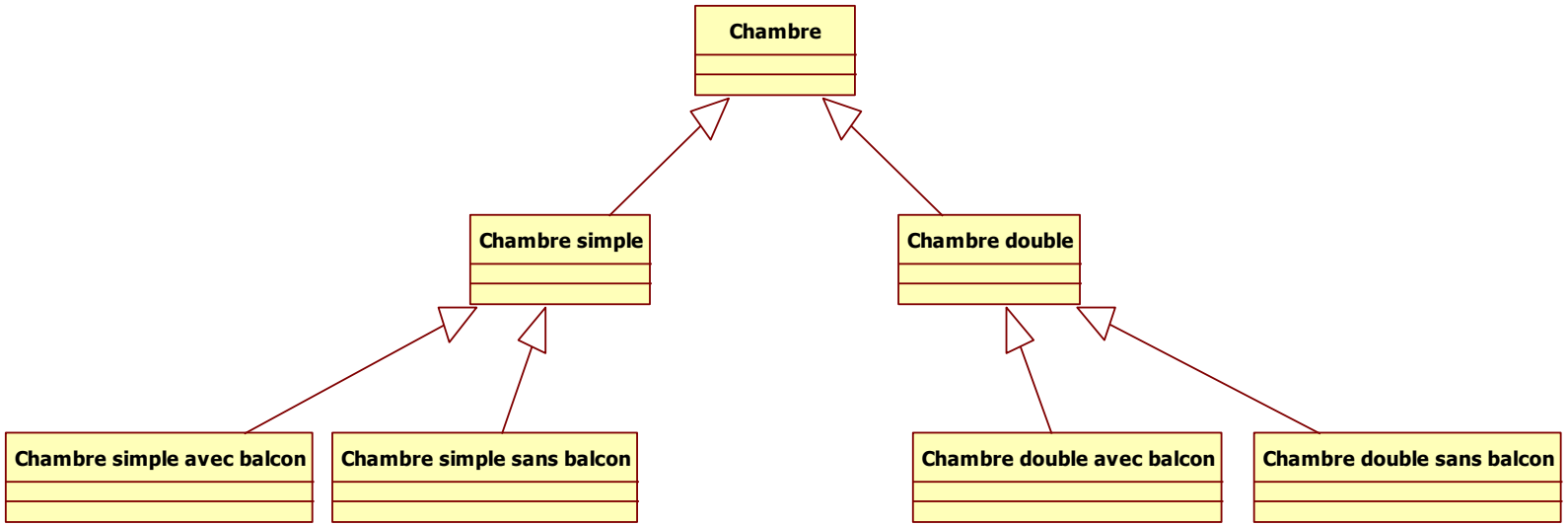
- Les diagramme de classes permettent de manipuler des classes ou des instances de classe (appelées aussi des objets)



- Les instances se reconnaissent facilement au fait que leur nom est souligné et que « : » précède le nom de la classe, que l'instance soit nommée ou non.

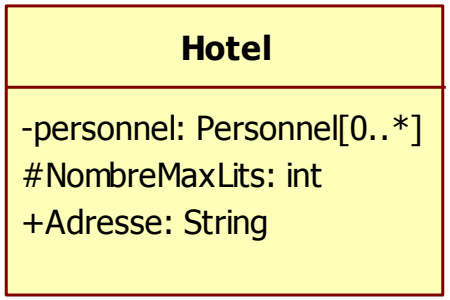
Héritage

- Les diagrammes de classes permettent de définir facilement l'héritage entre deux classes



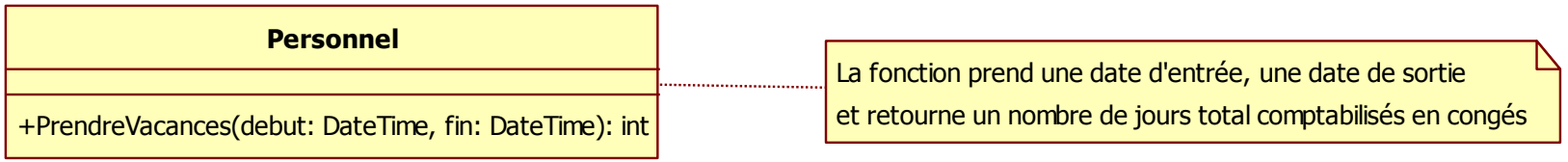
Attributs d'une classe

- Une classe hôtel
  - Avec un tableau de visibilité privée(signe -) de personnes
  - Avec un nombre total de lit qui est un entier de type protégé (signe #) avec une valeur initiale de 40
  - Avec une adresse qui est une chaîne de caractères publique (signe +)



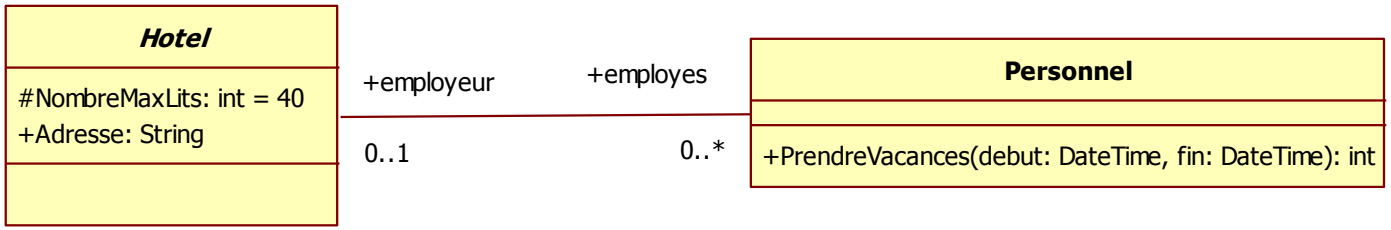
Opérations d'une classe

- En plus des attributs les classes peuvent disposer d'opérations
- Les opérations contiennent optionnellement des valeurs en entrée, en sortie, en entrée sortie ainsi qu'une valeur de retour



Les associations

- Plutôt que d'utiliser un tableau, on met des associations entre objets
  - Avec des noms de rôles aux extrémités
  - Et une cardinalité



Exercices

- Exercice 1 :Créer un diagramme de classe
  - Mettre un attribut a:string dans cette classe
  - Créer deux instances de cette classe
  - Leur ajouter deux AttributeLink afin que la première instance ait pour valeur de a « hello » et la deuxième « world ».

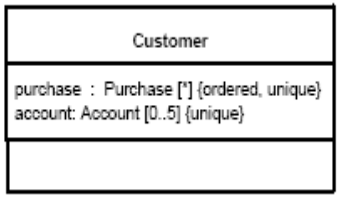
Exercices

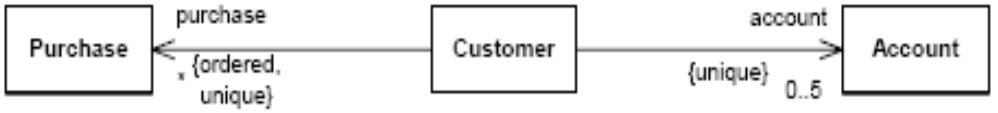
- Exercice 2
  - Créer un diagramme de classe sous StarUML
  - Un ajouter une classe MaClasse au diagramme
  - Ajouter un objet au même diagramme Objet1 instanciant MaClasse

- Exercice 3
  - Créer une classe bâtiment qui contient un champ numéro de bâtiment. Modifier les visibilités et types d'attribut.
  - Créer un objet chambre.
  - Créer une association entre le bâtiments et des chambres

Multiplicité

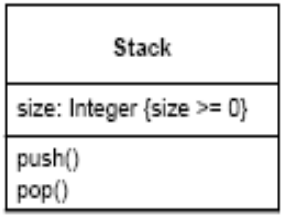
- Permet de préciser le nombre d'éléments contribuant à une relation
- On peut avoir deux façons les représenter
  - Au niveau d'un attribut
  - Au niveau d'une extrémité d'association

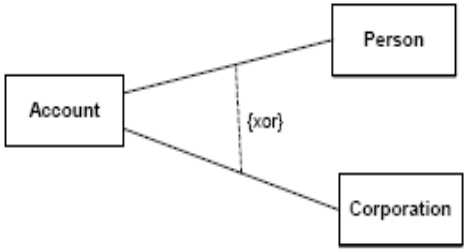


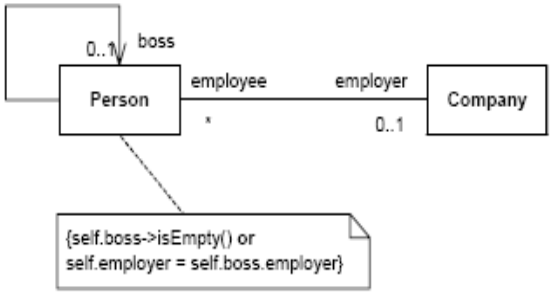


Contraintes (1)

- On a trois façons de noter les contraintes sous UML
  - Attachées à un attribut
  - A coté d'une ligne pointillée liant deux relations ayant une contrainte
  - Comme une note

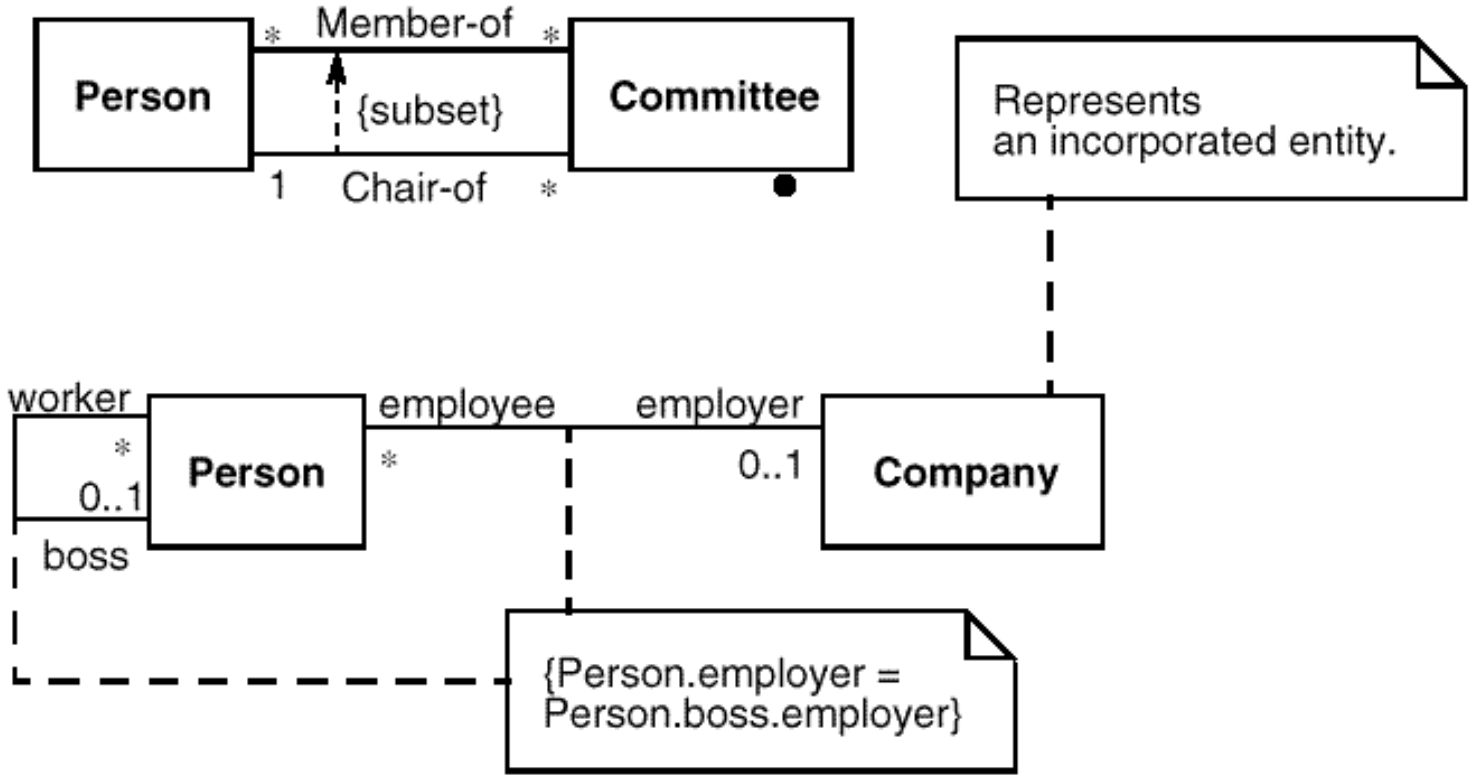






Contraintes (2)

Figure 5. Constraints



Grammaire (BNF) de la multiplicité

- <multiplicity> ::= <multiplicity-range> ['{' <order-designator> [',' <uniqueness-designator>] '}']
- <multiplicity-range> ::= [<lower> '..'] <upper>
- <lower> ::= <integer> | <value-specification>
- <upper> ::= '*' | <value-specification>
- <order-designator> ::= 'ordered' | 'unordered'
- <uniqueness-designator> ::= 'unique' | 'nonunique'

Exercices

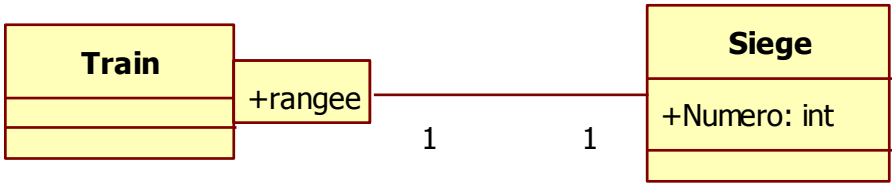
- Créer une contrainte de chaque type listé plus haut sous StarUML
  - Une contrainte attachée à un attribut (utiliser l'éditeur de collection Ctrl+F5)
  - Une contrainte rattachée à deux relations
  - Une contrainte signalée par une simple note
- Quel est l'avantage d'une contrainte attachée à un attribut par rapport à une simple note ?

Agrégation et composition

- Une association peut être une agrégation quand les objets associés ont une liaison forte avec la source. On utilise une représentation avec un losange vide au niveau de la source
- Une agrégation est une composition si quand on détruit la source on détruit tous les objets liés. On utilise une notation avec un losange plein

Les qualificateurs

- On peut préciser et réduire la cardinalité d'une association en utilisant un qualificateur



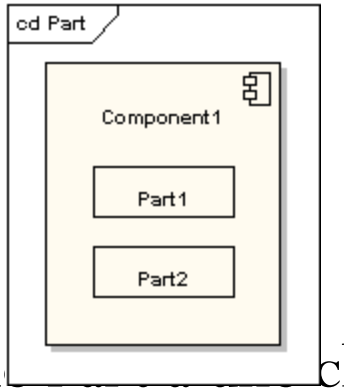
- Partitionne ensemble des instances et suggère les clés primaire du Mappage OR
- Démo : ajout d'un qualificateur à une relation

Structure composites

- Les diagrammes de classes peuvent détailler la structure interne des classes et les éléments accessibles depuis l'extérieur.
- On dispose pour cela
  - Des Ports
  - Des Part
  - Des Interfaces

Les Parts de classes

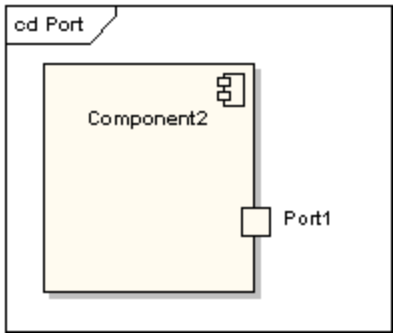
- Cette structure permet de décrire une sous partie d'un composant ou classe



- Exercice : rajouter un[illegible]classe dans un diagramme de classe et la connecter à une autre classe

Les Ports

- Les Ports décrive des sous éléments de classes accessibles depuis l'extérieur



- Exercice : rajouter un Port à une classe dans un diagramme de classe et la connecter à une autre classe

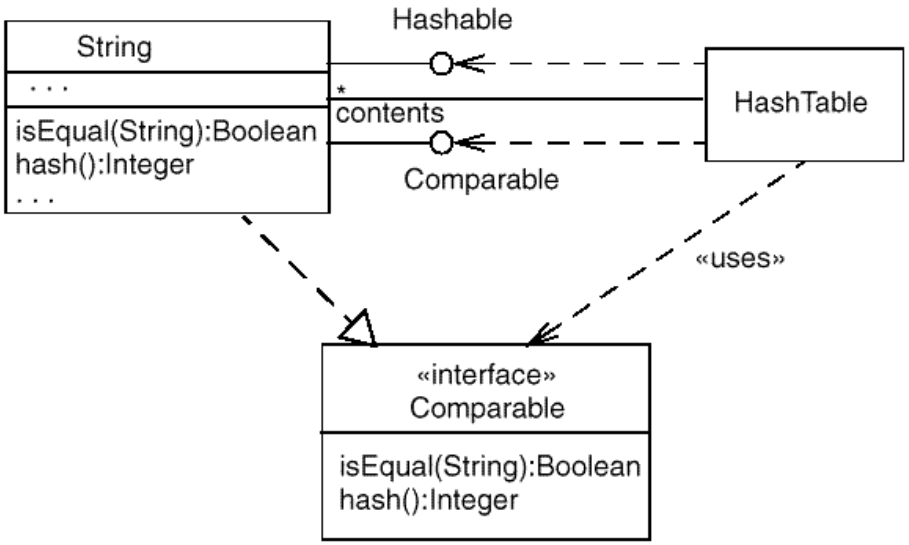
Interfaces (1)

- Les interfaces et leurs arborescences peuvent être obtenues par un stéréotype <<interface>>
- Un objet peut soit :
  - Avoir une relation de dépendance avec une interface (utiliser l'interface)
  - Être une réalisation d'une interface (implémenter l'interface)

Interfaces (2)

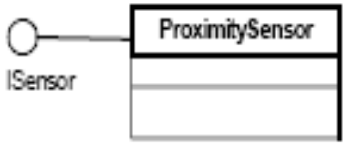
- Deux notations « mappées »
  - La notation cerclé
  - Ou l'usage du stéréotype <<interface>> sur une classe

Figure 13. Interface notation on class diagram



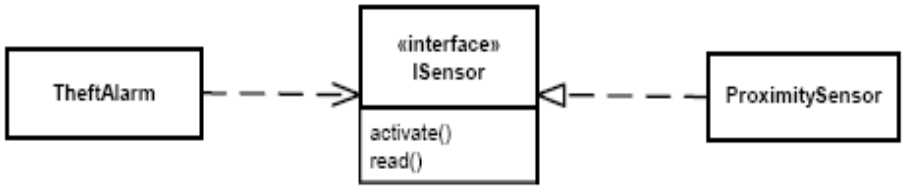
Interfaces (2)

- Réalisation



- Dépendance

- Autre notation possible :



Exercice

- Dans un diagramme de classes de StarUML
  - Créer une classe et deux interfaces
  - Créer une réalisation de la classe par rapport à la première interface
  - Créer une dépendance de la classe par rapport à la deuxième interface
- Dans quel cas de notation sommes nous ?

Les diagrammes dynamiques (1)

- Les diagrammes de classe sont une extension objets de méthodologies comme Merise
- Il leur manque tous les aspects dynamiques
  - Différents états des objets
  - Transition entre états suite à des stimulis
  - Réponse a des évènements et échange de messages entre objets

Les diagrammes dynamiques (1)

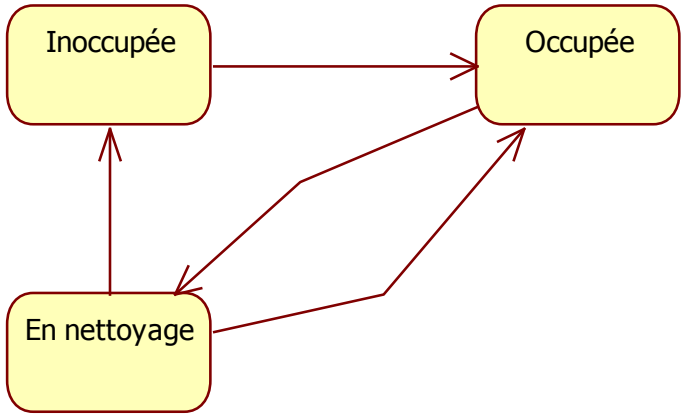
- Les différents diagrammes dynamiques sont :
  - Diagrammes d'états
  - Diagrammes d'interactions
    - Diagrammes de séquences
    - Diagrammes de collaboration
  - Diagrammes d'activité
- Notes : Sous StarUML les diagrammes d'interaction (séquence et collaboration) sont regroupés dans des instances d'ensemble de collaboration (CollaborationInstanceSet)

Diagrammes d'états (1)

- Ces diagrammes se focalisent sur une classe donnée
- Un classe peut avoir différents états caractérisés par différentes valeurs de ses paramètres (vision ensembliste, partition des classes)
- Des transitions relient ces états

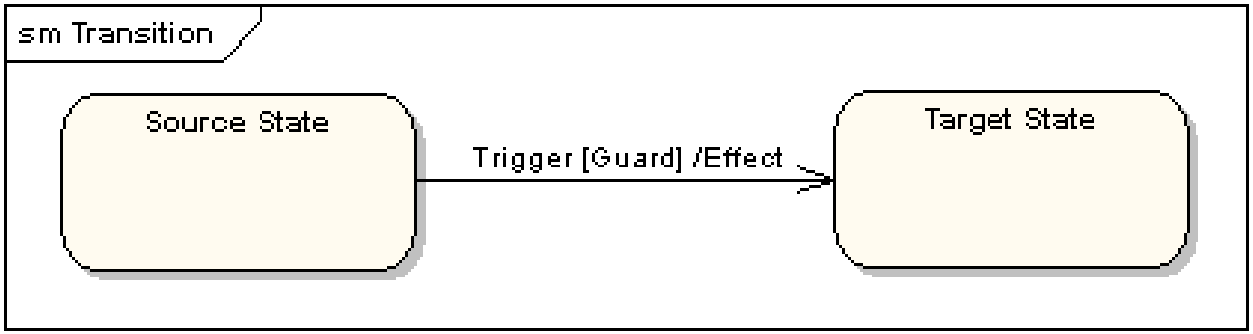
Diagrammes d'états (2)

- Exemple - Cas de l'objet chambre :
  - On identifie trois états : inoccupée, occupée et en nettoyage
  - On identifie les transitions entre ces états (flèches orientées)



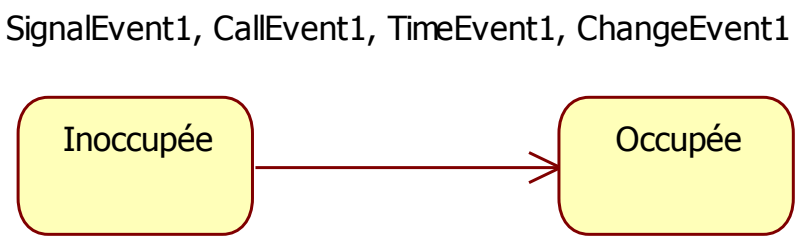
Diagrammes d'états (3)

- Les transitions :
  - Les transition permettent la migration entre états.
  - Elles résultent d'un évènement.
  - Elles portent le nom et les paramètres de l'évènement associé



Diagrammes d'états (4)

- Quatre catégories d'évènements peuvent déclencher de transitions
  - Un évènement de type signal : SignalEvent (par exemple le callback d'un appel asynchrone)
  - Un appel direct de méthode : CallEvent
  - Le déclenchement d'un timer : TimerEvent
  - Le changement d'une valeur : ChangeEvent

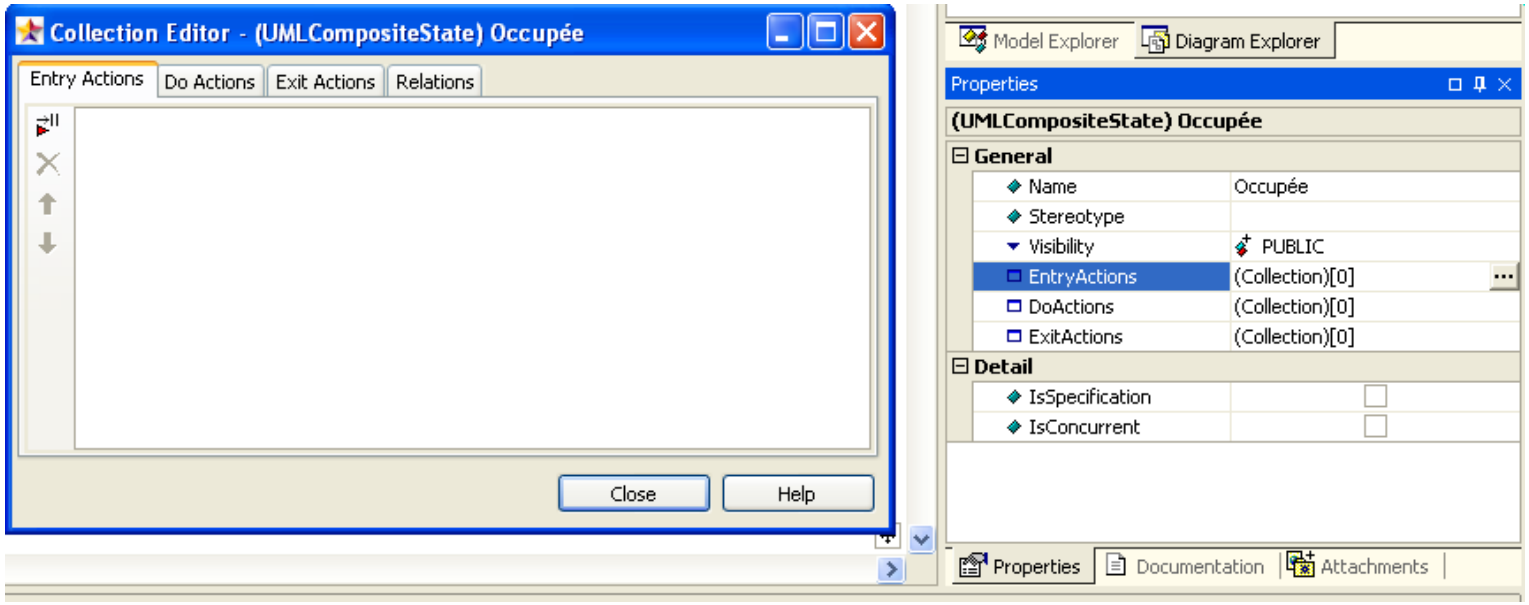


Diagrammes d'états (5)

- Les transitions, lors de leur déclenchement peuvent réaliser des actions (courtes dans le temps) appelées effects.
- Elle peuvent aussi n'avoir lieu que sous certaines conditions (GuardConditions)

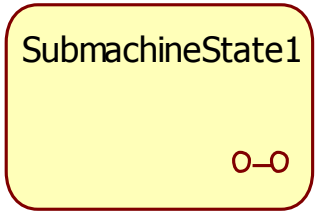
Diagrammes d'états (6)

- Les états peuvent eux aussi déclencher des actions courtes :
  - Entry action, Do Actions, Edit Actions



Diagrammes d'états (7)

- Les états peuvent être hiérarchiques et contenir des sous-états.
- Les diagrammes d'états d'un sous-états peuvent être référencés dans le diagramme principal via :



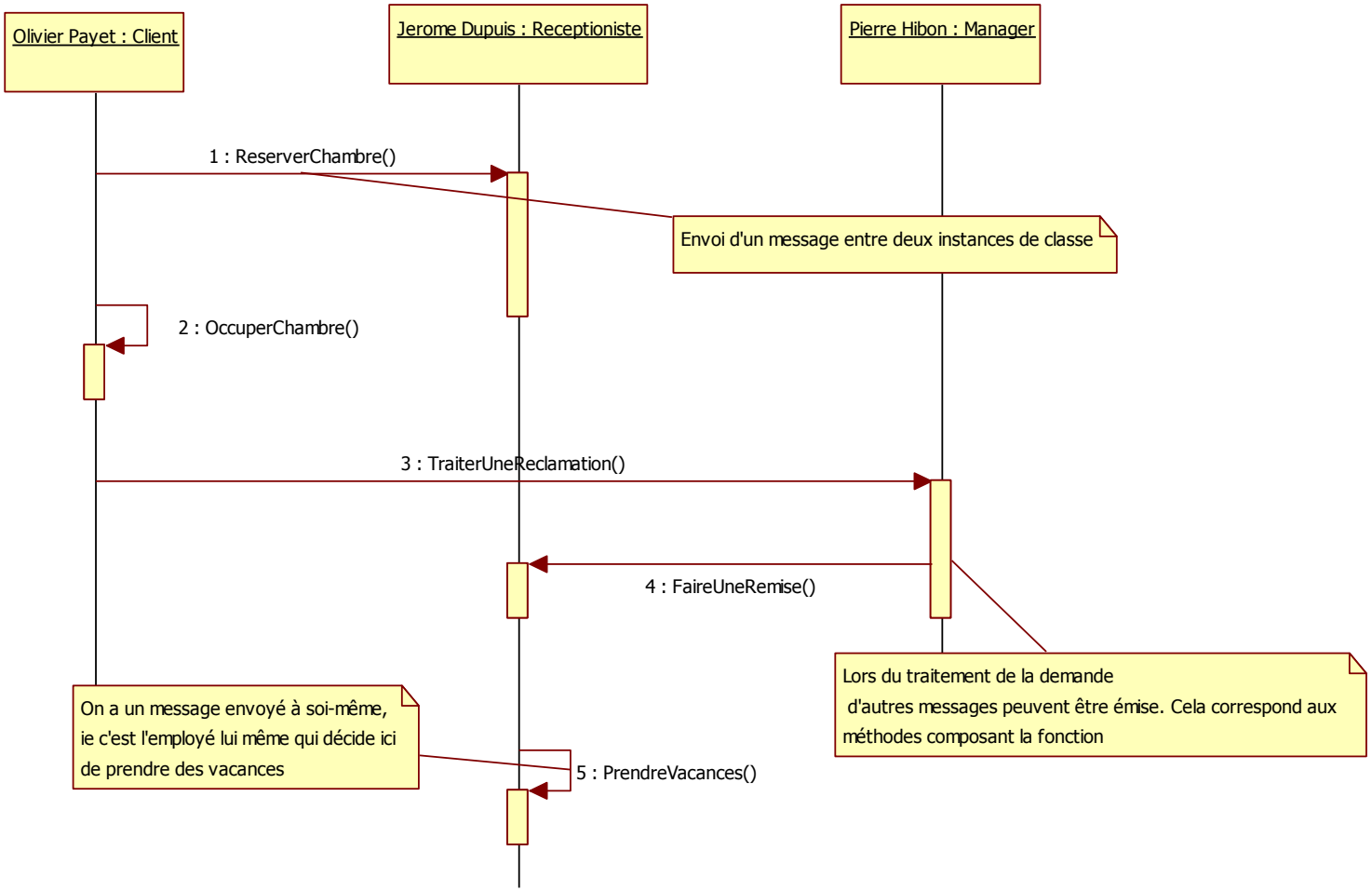
Exercices

- 1) Reproduire le diagramme de changements d'états pour une chambre
- 2) Créer un diagramme d'état pour un membre du personnel, en considérant le cas d'une personne non encore embauchée.

Solution

Diagrammes de séquence (1)

- Décrit une dynamique d'échange entre instance de classes

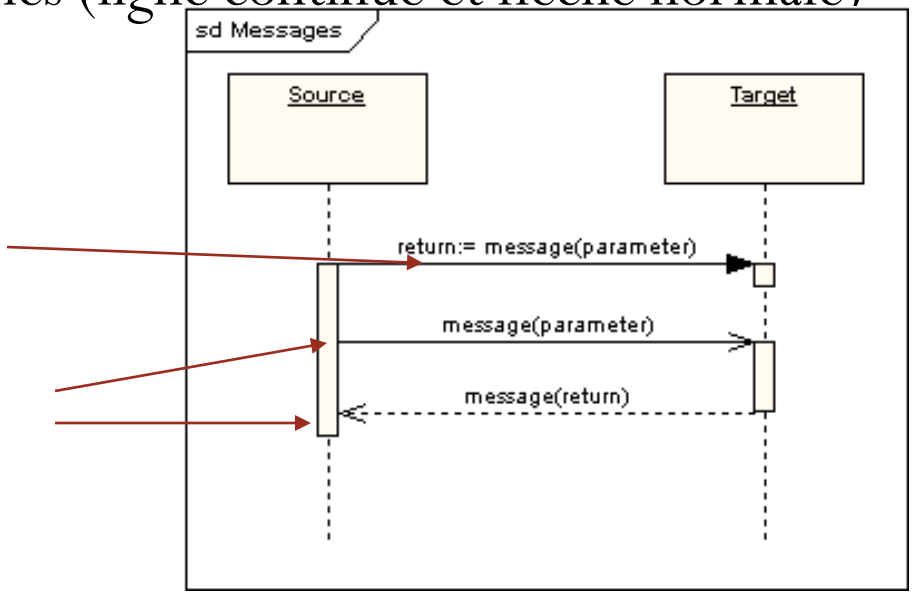


Diagrammes de séquence (2)

- Les diagrammes de séquence permettent de capturer un enchaînement d'évènements et l'envoi de messages entre objets
- Il permet de visualiser la responsabilité : qui est responsable de déclencher une méthode (prendre des vacances par exemple)
- Il ne peuvent pas représenter des procédures complexes
  - Conditions et branchements

Diagrammes de séquence (3)

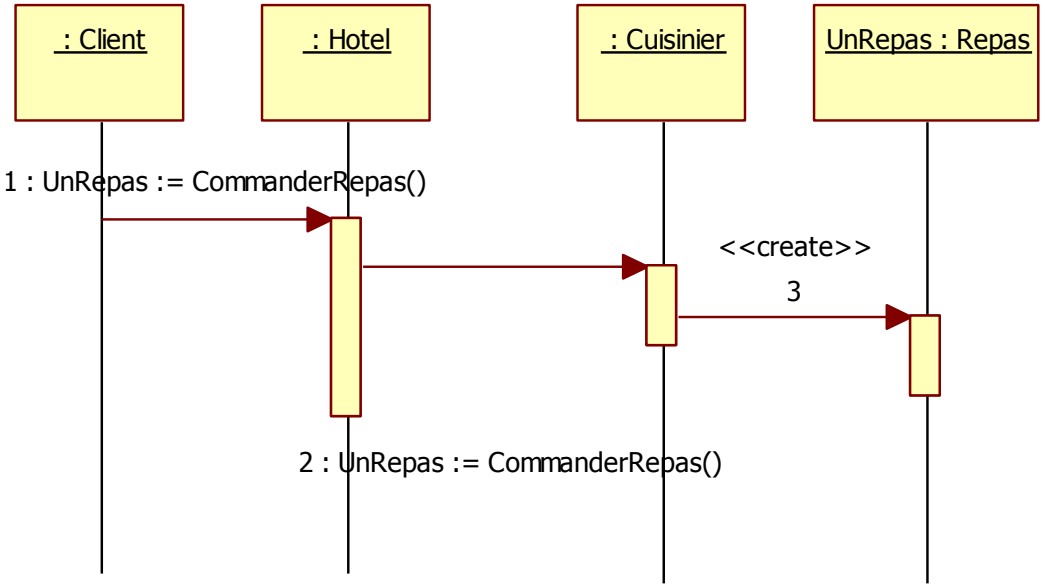
- On peut envoyer des message synchrones (ligne continue et flèche solide) et asynchrones (ligne continue et flèche normale)



- Le retour d'un message asynchrone est représenté avec des pointillés.

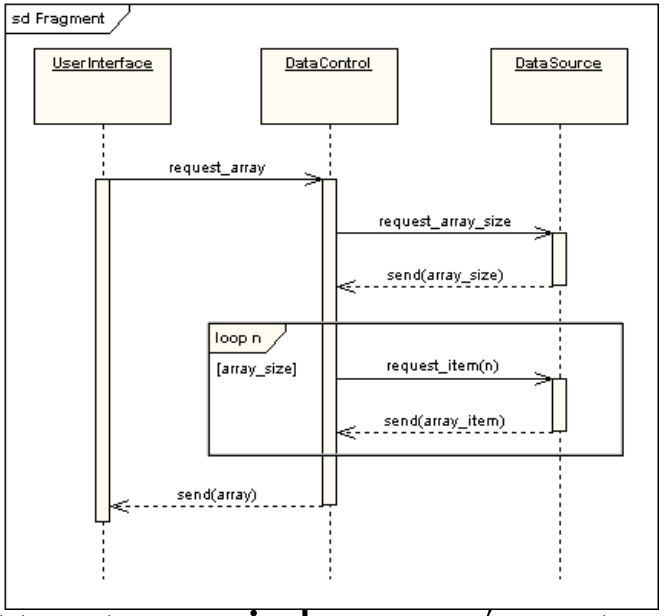
Diagrammes de séquence (4)

- Création et destruction
  - On dispose aussi de deux autres types de messages : create et destroy pour l'instanciation et la libération d'un objet



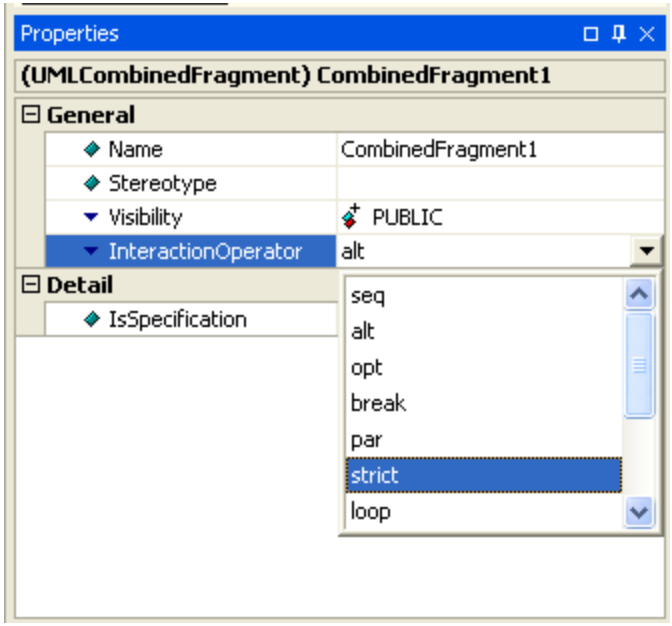
Diagrammes de séquence (5)

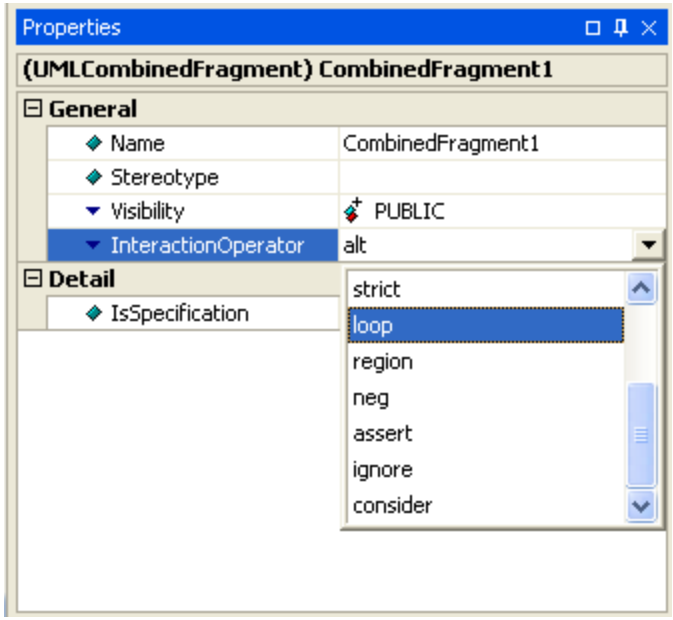
- On peut représenter des boucles en utilisant des fragments combinés (loop)



- Les fragments combinés permettent aussi de représenter des alternatives (alt), des régions (region), des assertions (assert),…

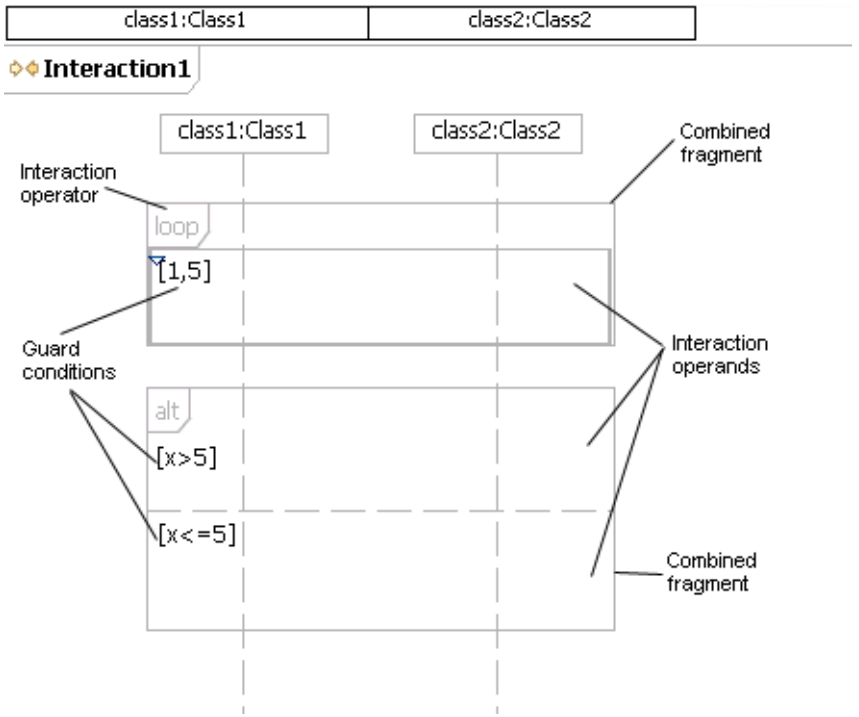
- Les différents types de fragments combinés sous StarUML





Diagrammes de séquence (6)

- Les opérandes d'interaction permettent de créer des catégories de traitements dans les fragments
- Par exemple :



Diagrammes de séquence (7)

- En fait les fragments permettent se s'affranchir de certaines limites des diagrammes de séquence en rajoutant
  - Les boucles
  - La logique conditionnelle
  - Break
  - Fonctionnement parallèle
  - …
- Mais ce sont les diagrammes de collaboration qui sont le plus adapté au cas des procédures complexes.

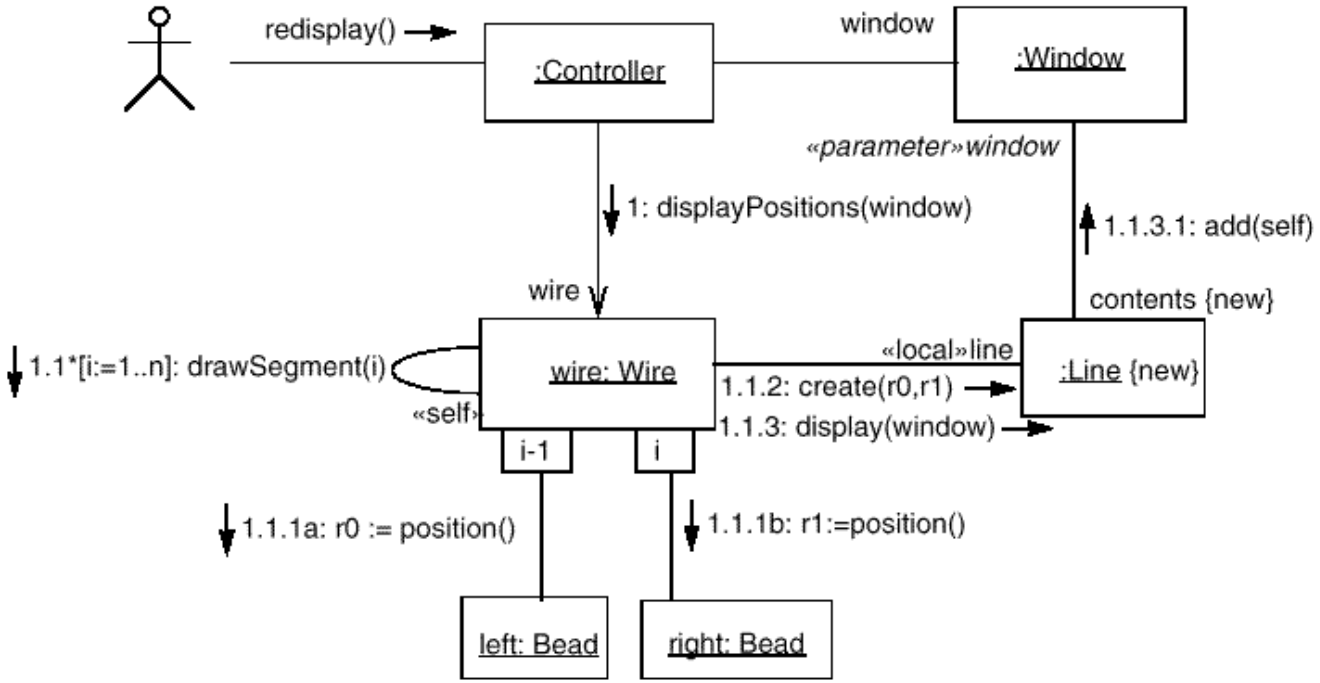
Diagrammes de collaboration (1)

- Appelés aussi de manière plus moderne diagrammes de communication
- Les diagrammes de collaboration se focalisent en priorité sur la relations entre objets (via les liens). Les stimulis sont alors secondaires
- On dessine ainsi différentes instances d'objets et on les relie par des liens sur lesquels on pourra mettre un ou plusieurs stimulis

Diagrammes de collaboration (2)

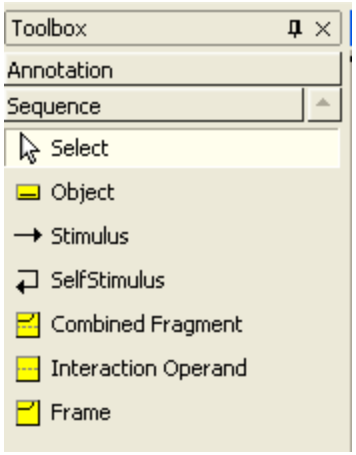
- Exemple de diagramme de collaboration

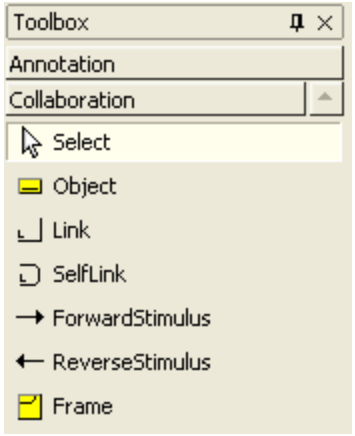
Figure 37. Collaboration diagram



Diagrammes de collaboration (3)

- Comparaison palettes séquence et de collaboration dans StarUML



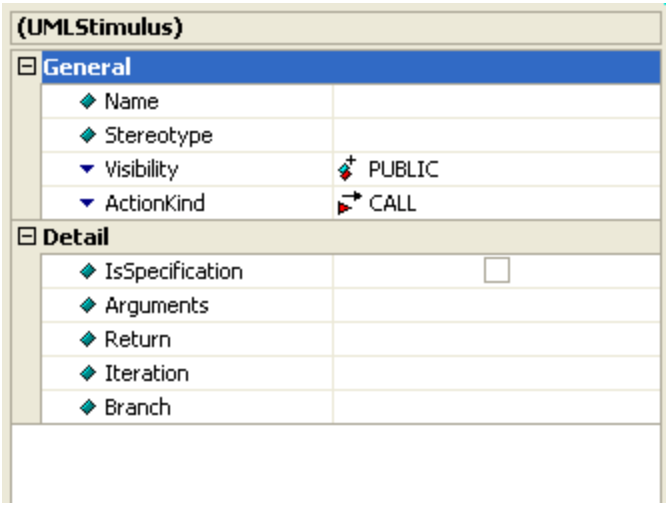


Diagrammes de collaboration (4)

- Les stimulis correspondent à l'appel d'une méthode sur l'objet cible.
- Il peuvent être adaptés pour :
  - Passer des paramètres qui sont des instances d'objets présent dans le diagramme
  - Retourne des valeurs nommées, utilisables elles aussi dans d'autres stimulis
  - Être répétées un certain nombre de fois
  - S'appliquer sous certaines conditions

Diagrammes de collaboration (5)

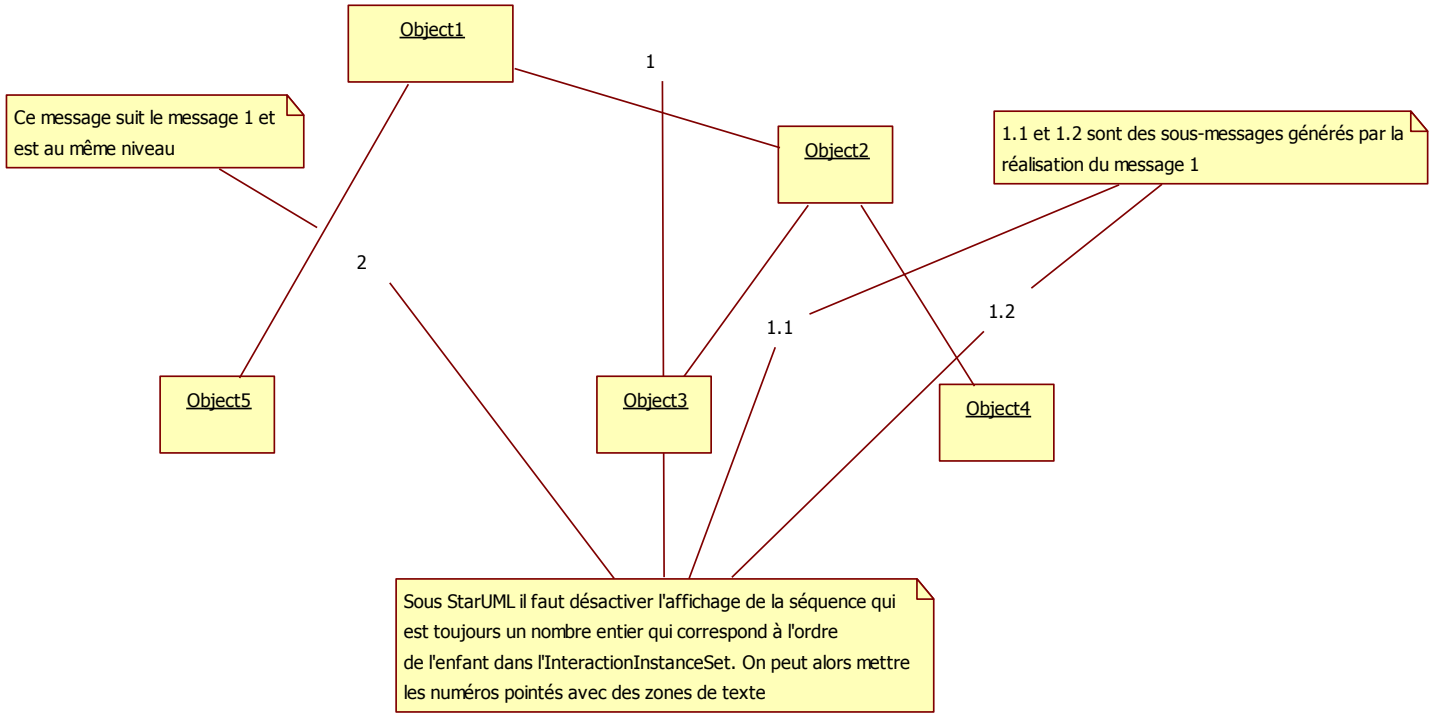
- Propriétés d'un stimuli



- Un stimuli avec comme nom monStimuli, valeur de retour s, condition (branch) a>0, itération de 1à 5 et arguments (a,3) s'affiche comme :
- *[1..5][a>0] s := monStimuli(a,3)

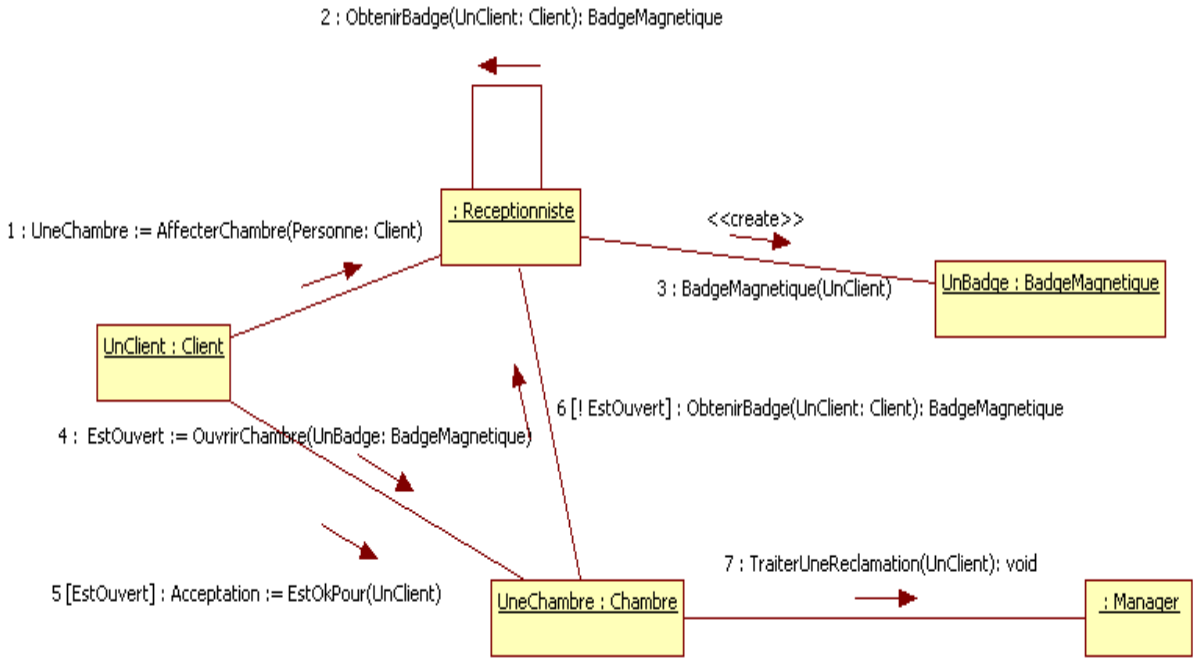
Diagrammes de collaboration (6)

- Décomposition des messages en messages secondaires etc.



Diagrammes de collaboration (7)

- Transposition du Use Case : Occuper chambre



Diagrammes de collaboration (8)

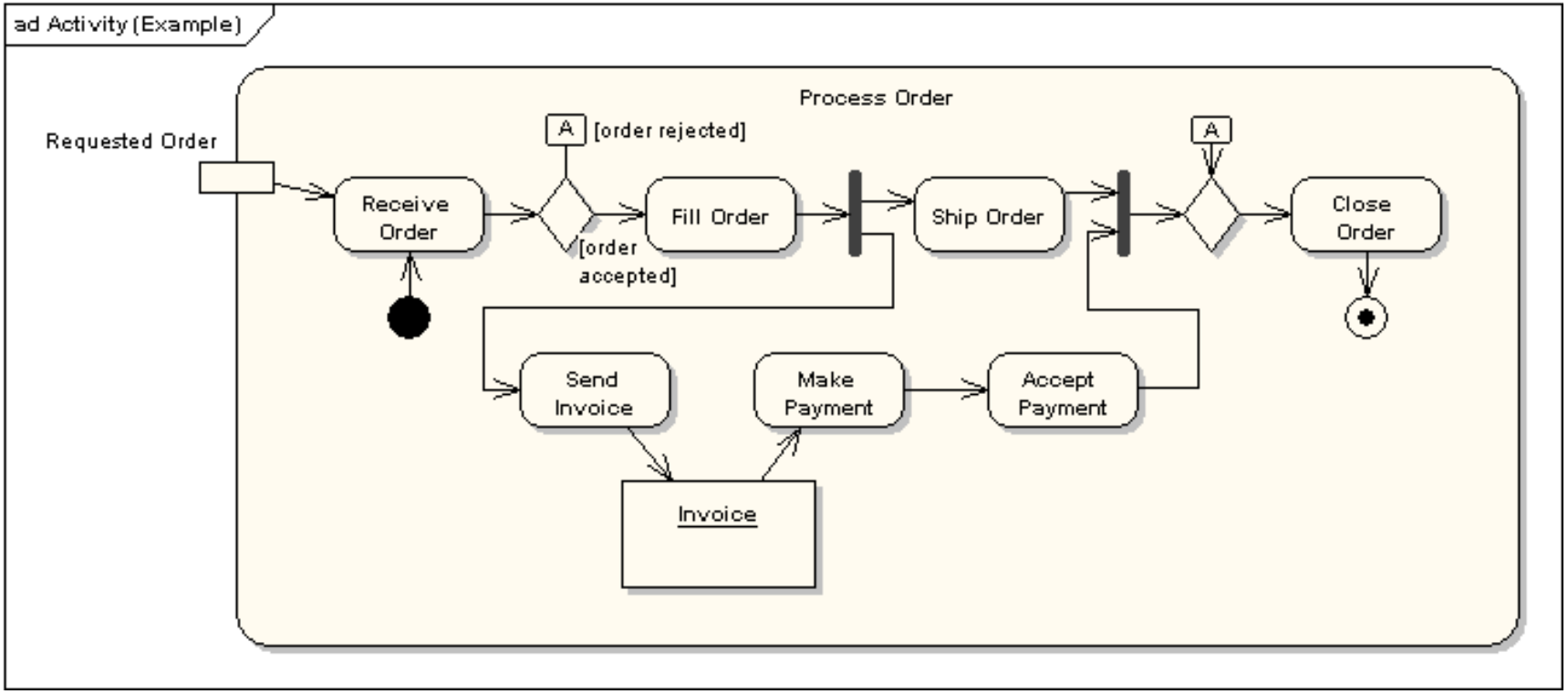
- Exercice :
  - Reproduire le diagramme précédent dans starUML
  - Note : pour faire apparaître le nom et le type d'un stimuli jouer sur la propriété MessageSignature du diagramme

Diagrammes d'activités (1)

- Les diagrammes d'activités permettent de décrire un workflow, i.e. un enchaînement de tâches
- Il sont proches des diagrammes d'états mais ils ont une composante temporelle où les diagrammes d'états décrivent des transitions au sein d'une « ontologie »

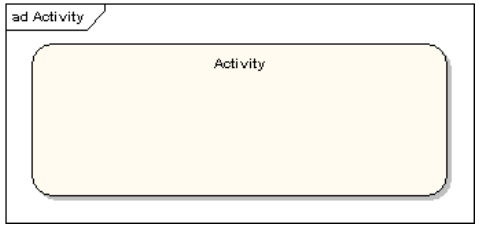
Diagrammes d'activités (2)

- Un exemple de diagramme d'activité

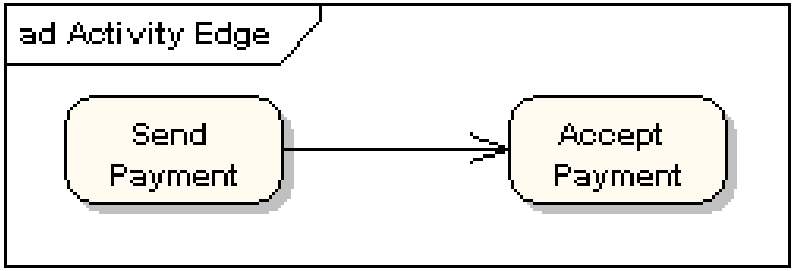


Diagrammes d'activités (3)

- Les activités sont représentées par des carrés au cotés arrondis



- Elles sont reliées entre elles par des transitions qui correspondent à une succession temporelle.



Diagrammes d'activités (4)

- Nœuds finaux :
  - Il y a deux types de nœud finaux.
  - Un nœud final lié à un parcours :

  - Un nœud final général du workflow. Lorsqu'il est atteint le workflow entier s'arrête

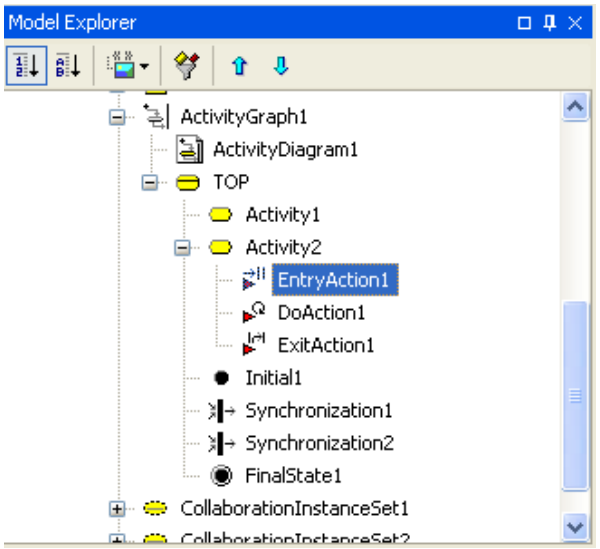
Diagrammes d'activités (4)

- Une activité est une opération « longue » qui se décompose en tâches
  - Une ou plusieurs tâches d'entrées (courtes)
  - Une ou plusieurs tâches interne

(DoAction qui sont longues)

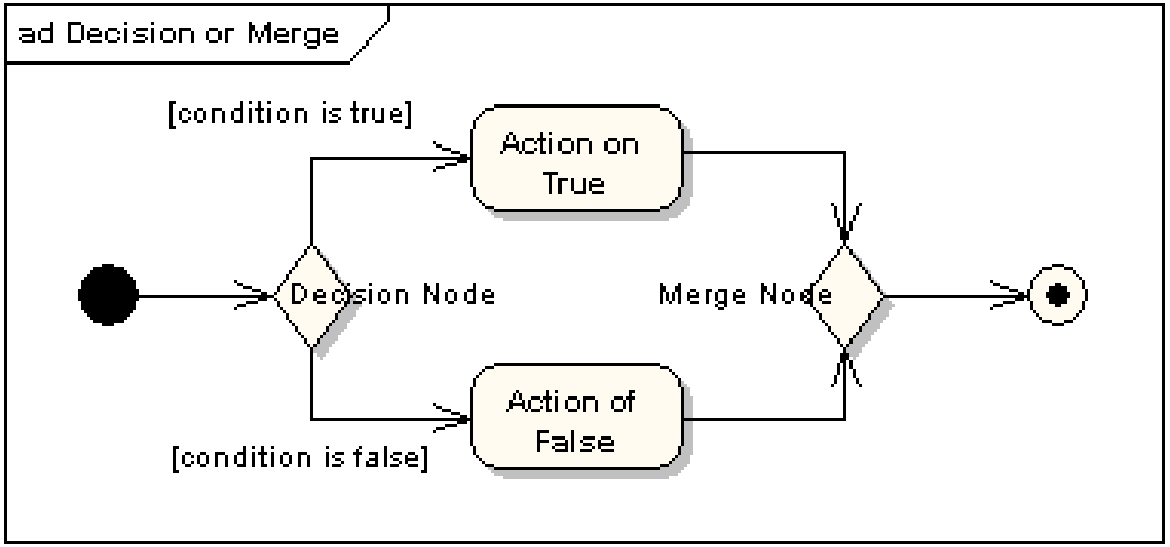
  - Une ou plusieurs tâches

de sortie (courtes)



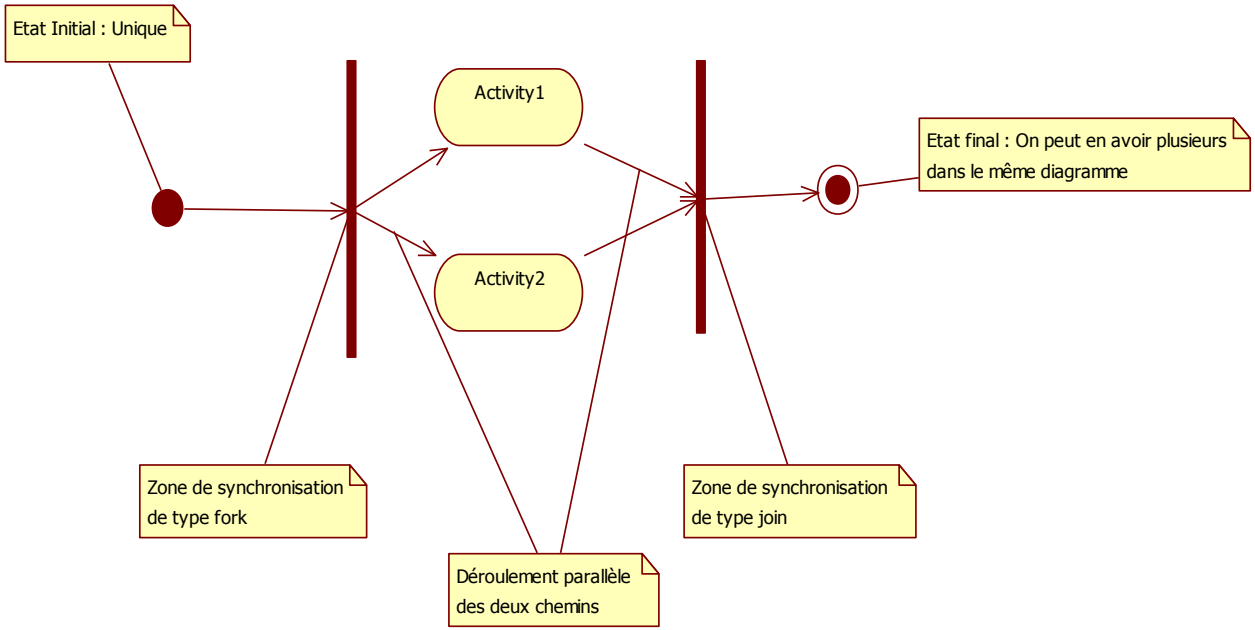
Diagrammes d'activités (5)

- Nœud de décision (choix de branche suivant condition de garde) et nœud de regroupement (merge node)



Diagrammes d'activités (5)

- Fork et Join : permettent un déroulement parallèle et une synchronisation

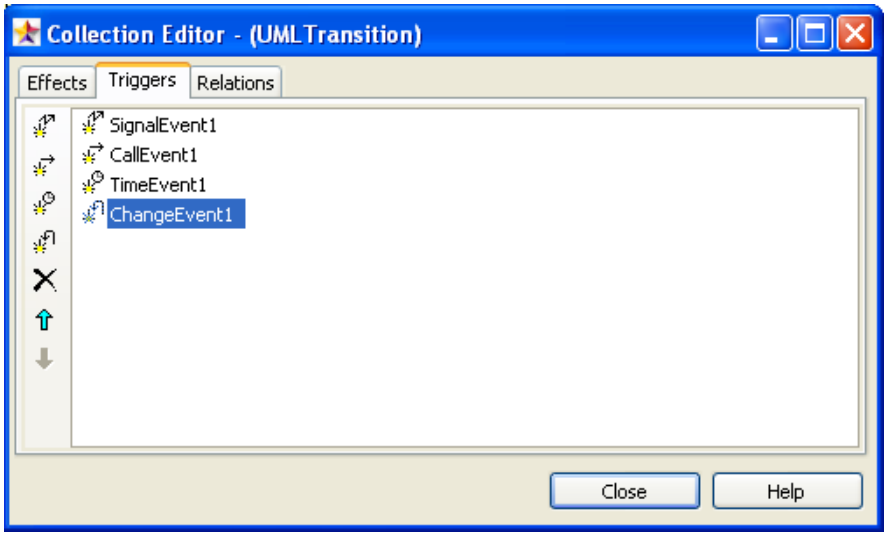


Diagrammes d'activités (4)

- Contraintes sur les transitions : une transition peut être munie d'une contrainte qui empêche son parcours si elle n'est pas à vrai (propriété GuardCondition).
- Une transition peut aussi déclencher une action (courte) appelée « effect » avant de passer à sa cible à condition que la condition de garde soit vérifiée.

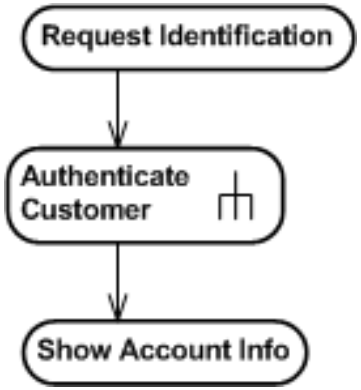
Diagrammes d'activités (5)

- Les transitions peuvent être déclenchées par différents déclencheurs (triggers)
  - Des signaux
  - Un appel de fonction
  - Un évènement de type timer
  - Un évènement de modification



Diagrammes d'activités (6)

- On peut décomposer une diagramme d'activité sous forme hiérarchique en utilisant la notion de sous-activité



- Une sous-activité fait appel à un autre diagramme d'activité dans un diagramme donné. On a une logique de réutilisation de diagrammes

Diagrammes d'activités (7)

- On peut indiquer des déclenchement asynchrones avec les icônes d'envoi (send) et de reception (accept) de signaux