

# ***ENSTA : cours IN204***

## ***Introduction à JAVA et UML***

Olivier Sigaud  
LIP6/ Anim atLab  
[olivier.sigaud@lip6.fr](mailto:olivier.sigaud@lip6.fr)  
01.44.27.88.53

# *Plan du cours 9*

- UML : cas d'utilisation
- UML : diagrammes de séquence
- UML : diagrammes états-transitions
- UML : démarche d'emploi

# *Cas d'utilisation et diagrammes de séquence*

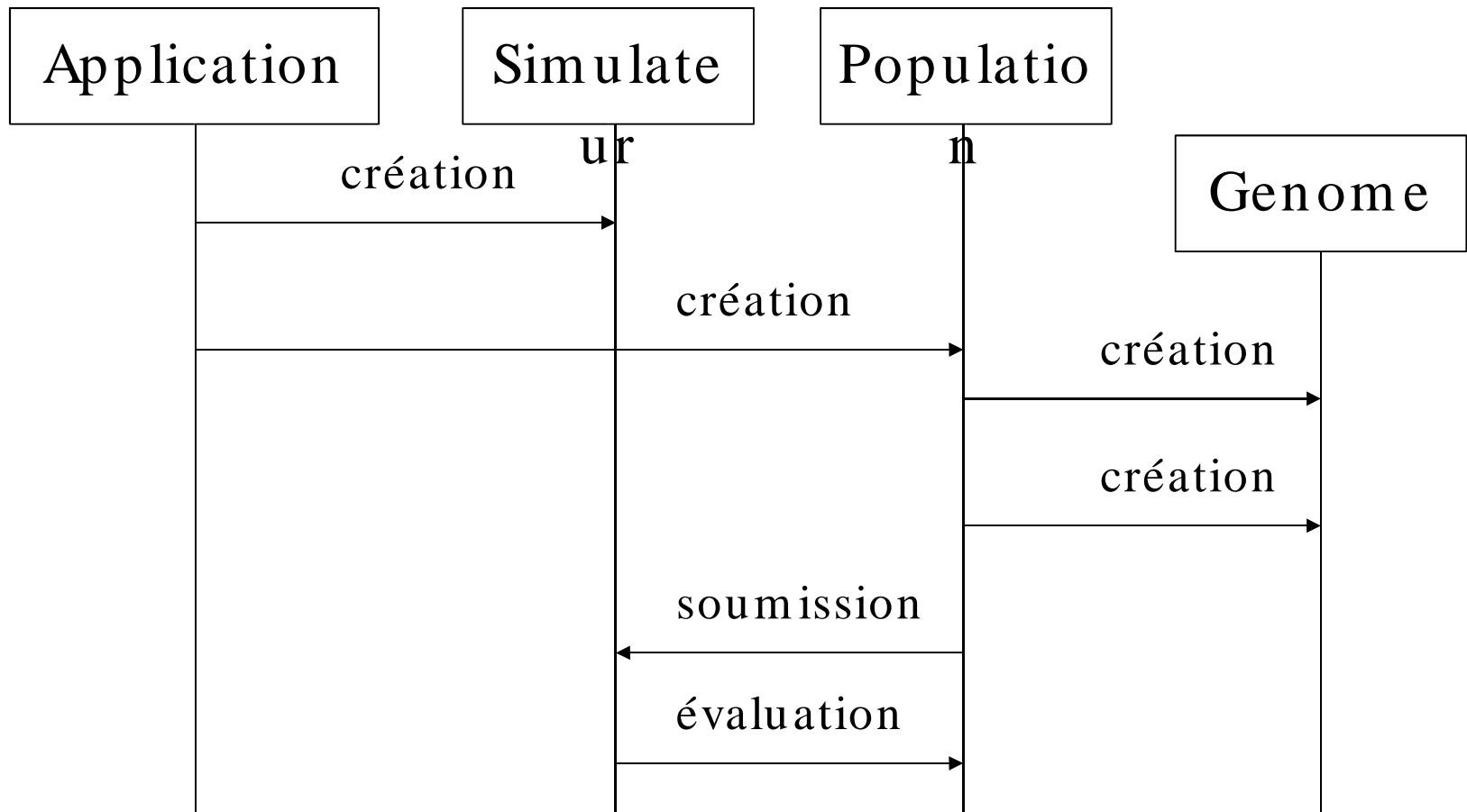
# *Des cas d'utilisation aux diagrammes de séquence (1)*

- Un **diagramme de séquence** décrit un **scénario** d'**interaction** entre objets du système et **acteurs externes**
- Un **scénario** peut être vu comme une des instances d'un **cas d'utilisation**
- La description doit être suffisamment générale et exhaustive pour identifier tous les **algorithmes**

# *Des cas d'utilisation aux diagrammes de séquence (2)*

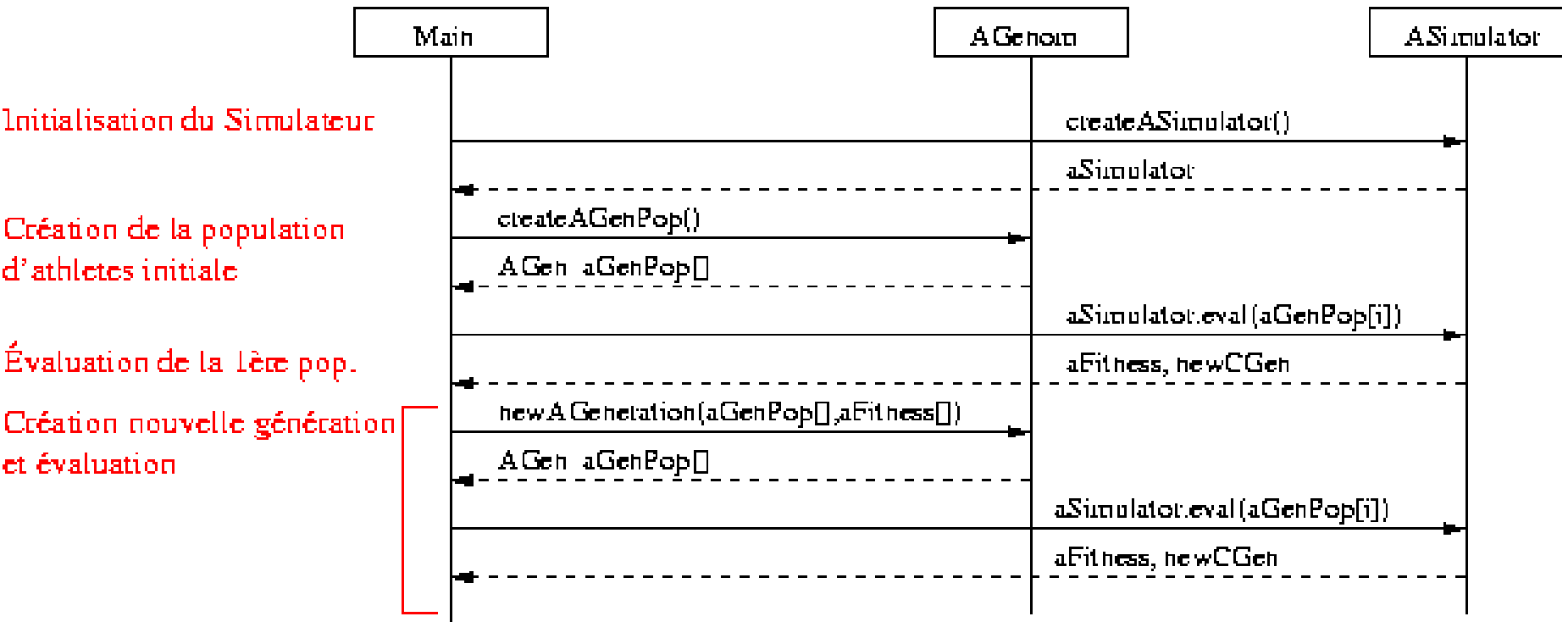
- **Problème** : la combinatoire des interactions peut être immense.
  - décrire quelques scénarios **dans les cas nominaux**
  - décrire les scénarios **aux limites**
  - décrire les scénarios **d'anomalie**
- Un analyste programmeur doit pouvoir coder en lisant des diagrammes de séquences

# *Le diagramme de séquence*



Permet d'identifier les interactions entre objets concernés

# Le diagramme de séquence



Identifier les classes et méthodes d'interface concernées



# *Usage du diagramme de séquence*

- En tant que document d'analyse : sert à représenter de plus en plus précisément la dynamique du système
- En tant que document de conception : sert à figer les interactions entre les sous-parties du logiciel
- En tant que document d'implémentation : sert à décrire certains algorithmes (la partie interaction)
- Peut conduire à des re-découpages des vues statiques

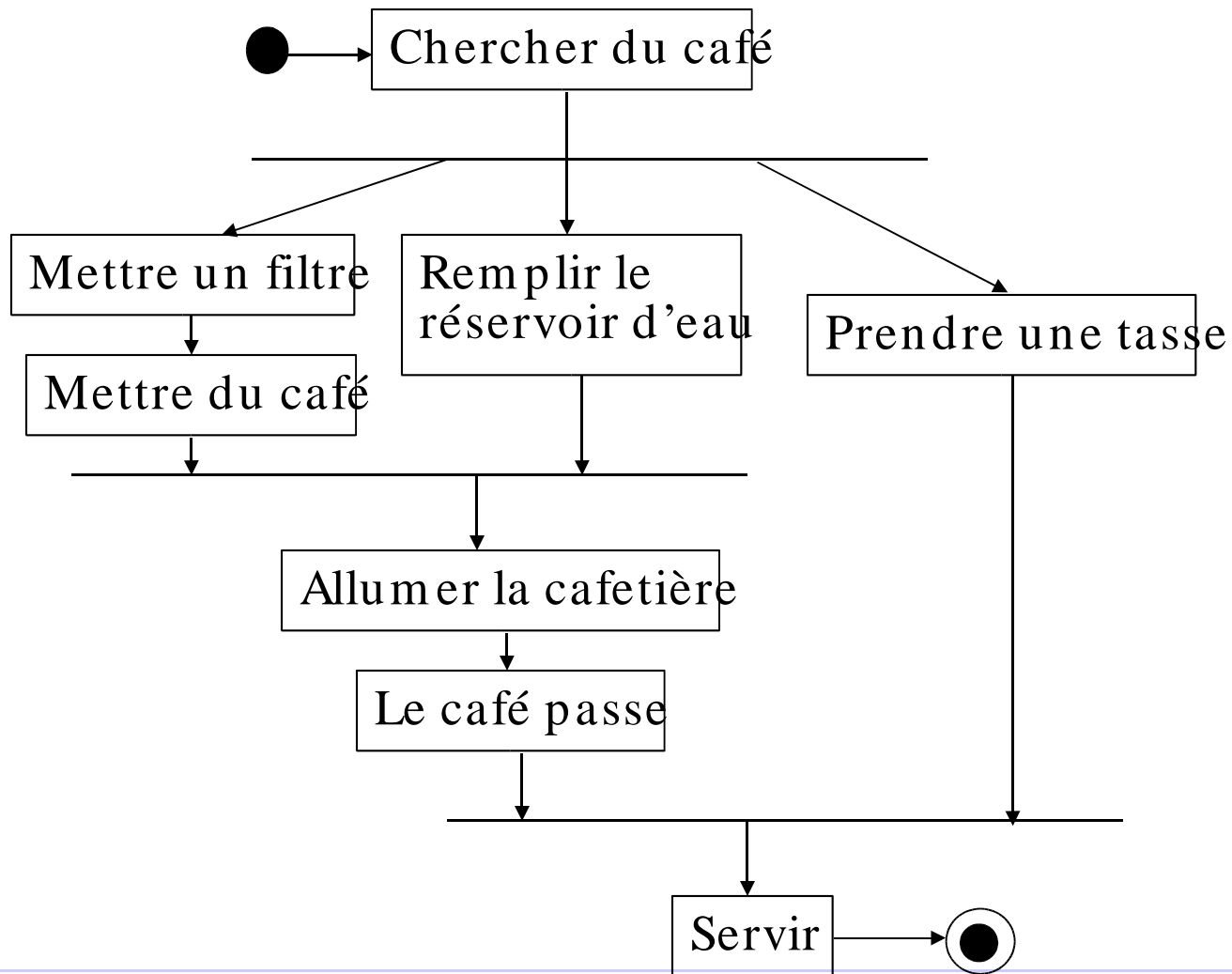
Essentiel pour le passage de l'analyse à la conception

# *Interaction classes-séquences*

- Les objets concrets représentés dans un scénario sont des acteurs ou des objets (instances de classes)
- Construire le scénario peut faire apparaître de nouvelles classes (notamment dans les transmissions de paramètres)
- En conception, cela fait surtout apparaître les méthodes et éventuellement des attributs
- Réaliser les diagrammes de séquence permet donc de lister les **méthodes** dont on aura besoin
- Les AGL en tirent parti : quand on rajoute un appel de méthode dans un diagramme de séquence, la méthode est ajoutée dans le diagramme de classes
- Toujours en conception, penser à détailler **tous les paramètres**
- **Attention, les classes abstraites n'apparaissent pas -> ne pas se contenter des diagrammes de séquence pour concevoir une architecture générique !**

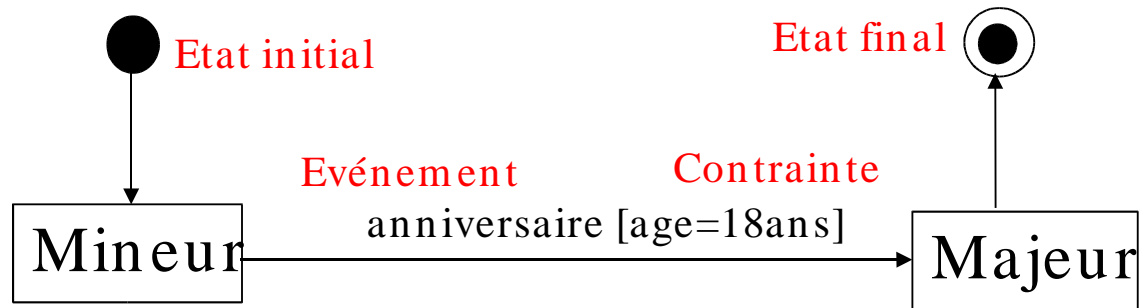
# *Automates*

# Diagramme d'activité



# *Le diagramme états-transitions*

Source : Statecharts (David Harel)



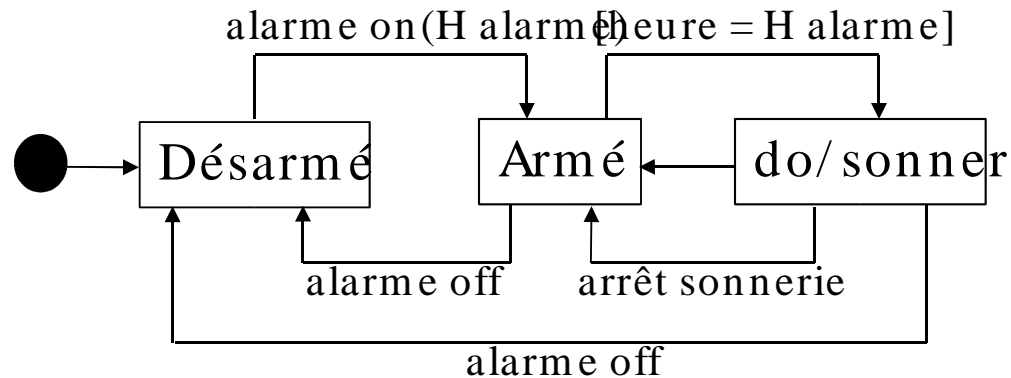
Permet de représenter sous forme d'**automates** les objets dotés d'une dynamique com

Très utile pour les **applications interactives**

Délicat à utiliser => Ne pas en abuser

# Exemple : réveil matin

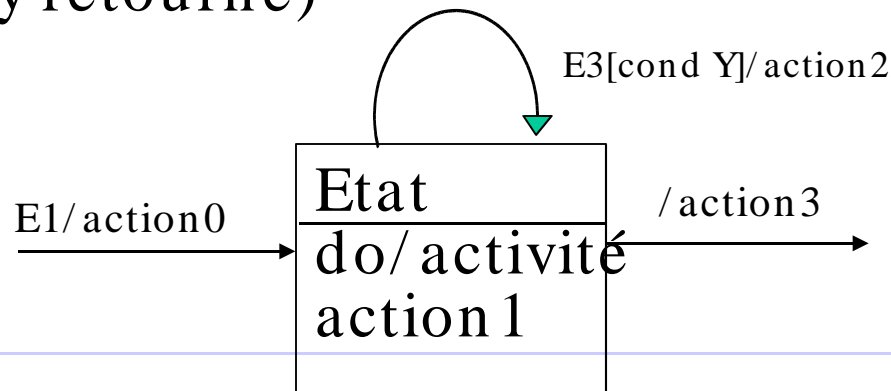
- **3 boutons** : alarme on/ off, arrêt sonnerie, réglage alarme



- **Transition automatique** à la fin de la sonnerie
- **Constat** : notation très compacte pour décrire toute la dynamique

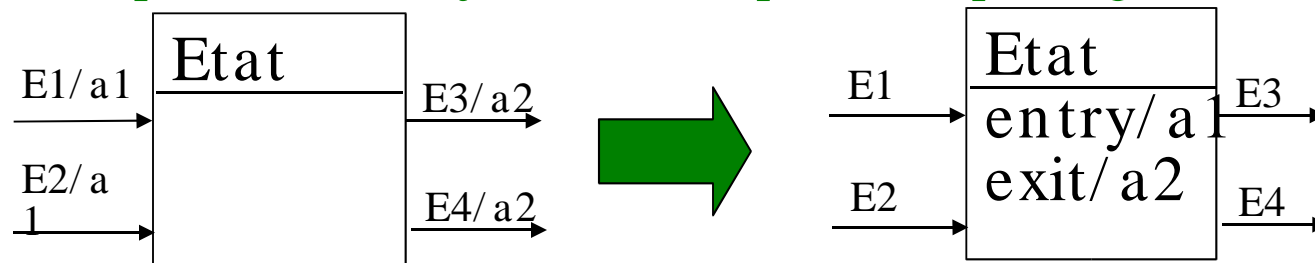
# Actions et activités

- **Envoi d'événement** : déclenchement d'une transition.
- **Action** : opération instantanée déclenchée par une transition.
- **Activité** : opération durable, interruptible, associée à un état.
- **Action interne** : action qui n'entraîne pas de transition
- **Transition automatique** : transition sans événement, déclenchée à la fin de l'activité.
- **Transition propre** : transition vers le même état (on en sort et on y retourne)



# Action en entrée et en sortie

- **Action en entrée** : action exécutée à chaque entrée dans l'état
- Exemple : redessiner le contenu de la fenêtre quand la souris y entre
- **Action en sortie** : action exécutée à chaque sortie de l'état
- Exemple : mettre à jour un compteur de passage à la sortie



- Intérêt de factoriser les actions de sortie dans le cas hiérarchique

# Ordonnancement

- **Entrée :**

- action sur la transition d'entrée (a0)
- action d'entrée (a1)
- activité (activité)

- **Dans l'état :**

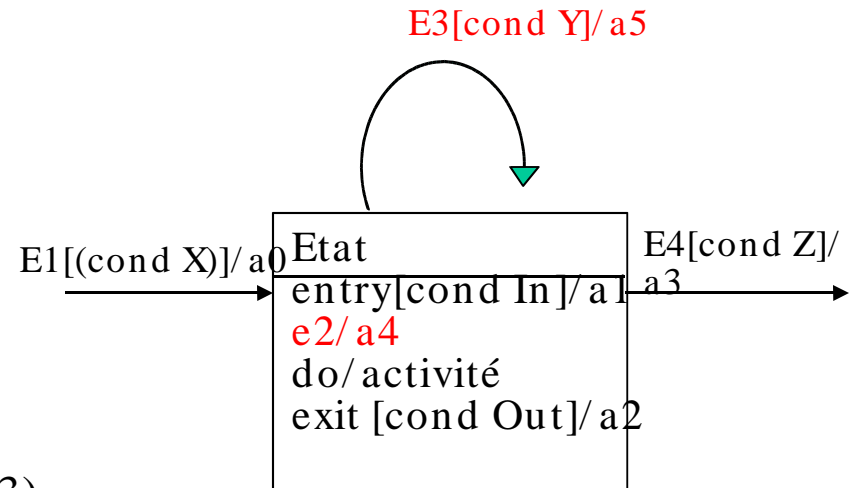
- interruption de l'activité (activité)
- exécution de l'action interne (a4)
- reprise de l'activité

- **En sortie :**

- arrêt de l'activité (activité)
- action de sortie (a2)
- action sur la transition de sortie (a3)

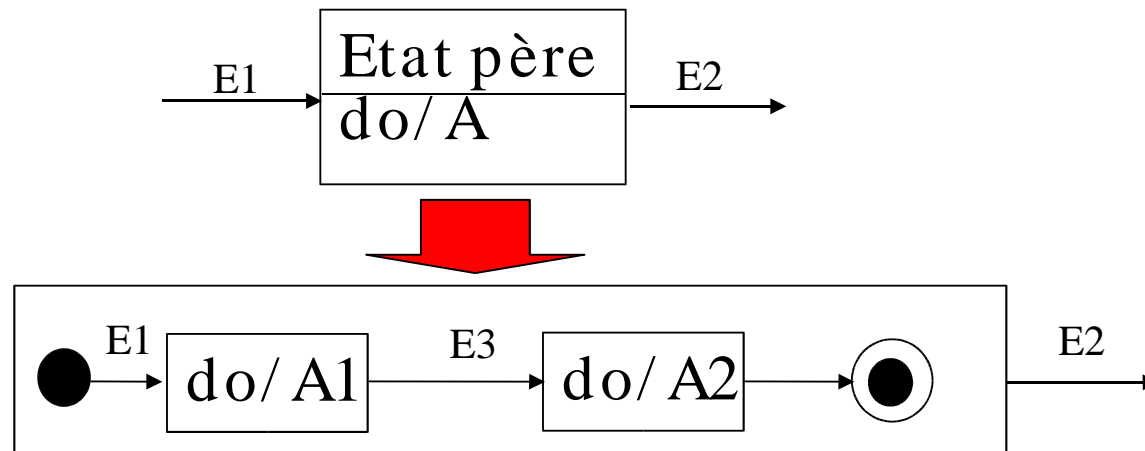
- **Transition propre :**

- ordre de sortie (activité, a2), (a5), puis ordre d'entrée (a1, activité)

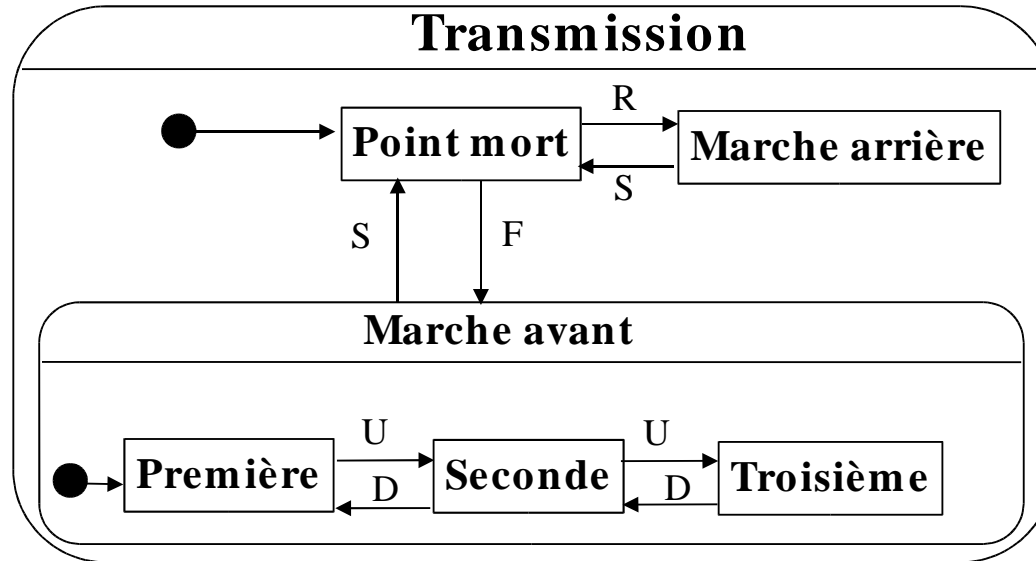


# Automates hiérarchiques

- Les diagrammes d'états à plat **deviennent rapidement illisibles**, il faut structurer hiérarchiquement.
  - **Raffiner** : décomposer en sous-états
  - **Synthétiser** : regrouper dans un même état



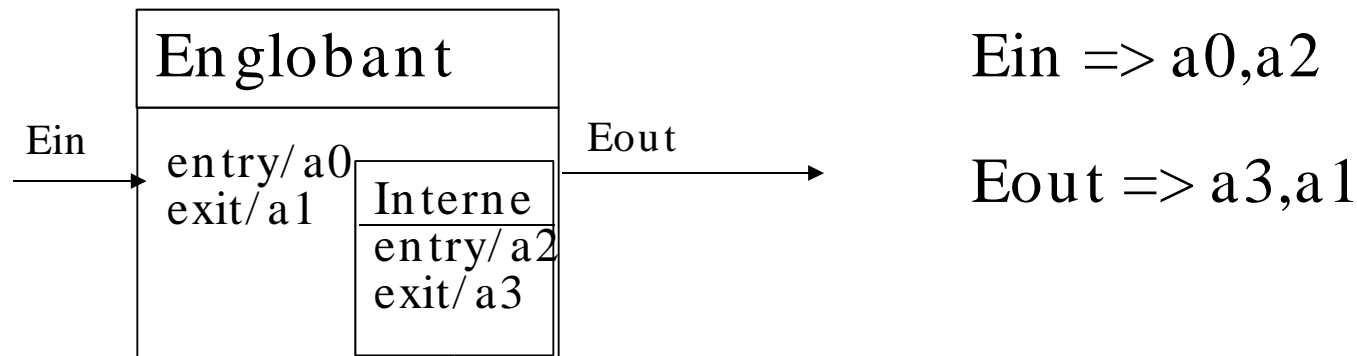
# Automates hiérarchiques



- Un **sous-état** hérite de son **sur-état** des actions internes, des transitions de sortie, il n'hérite pas des transitions en entrée ni des activités.

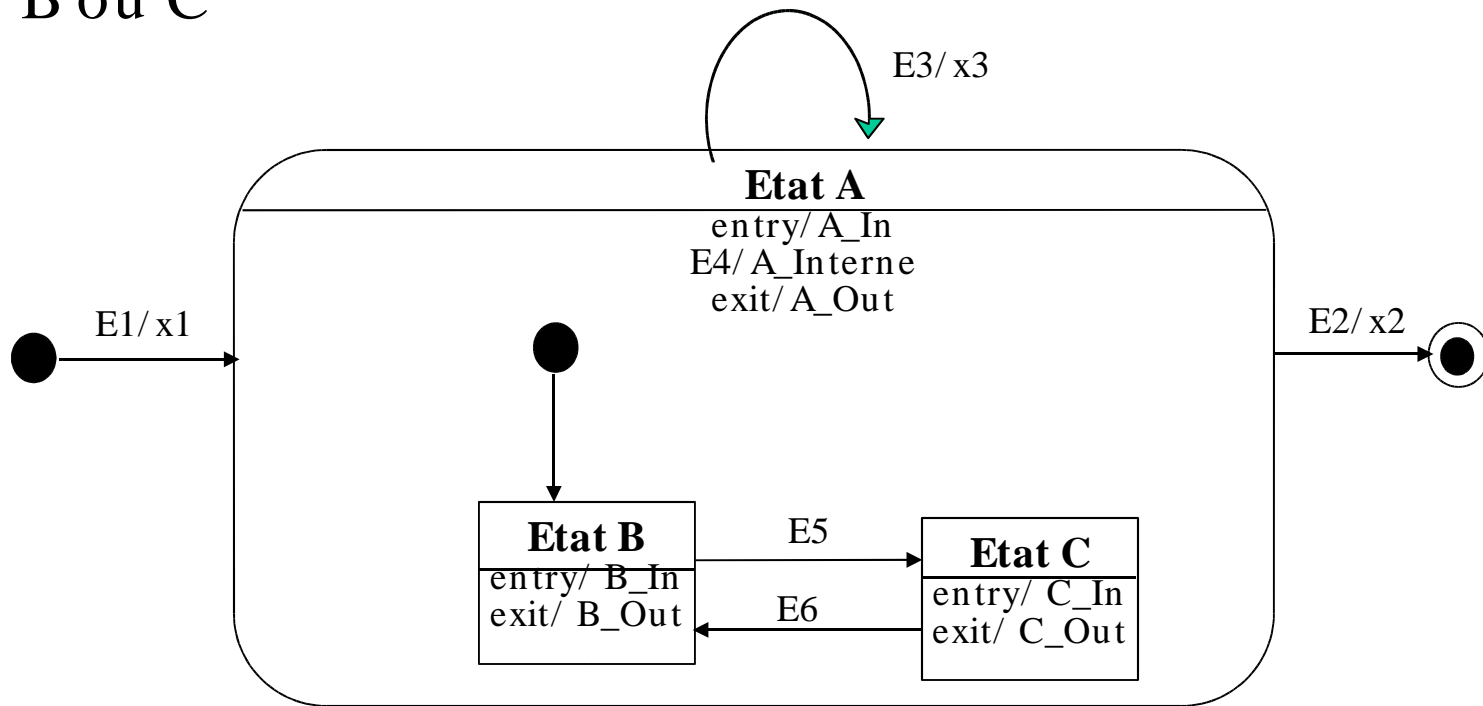
# Hiérarchie et ordonnancement

- Quand on entre dans l'état englobant, on entre dans l'état interne **initial**
- On commence par entrer dans l'état le plus englobant pour **aller vers le plus interne**
- On commence par **sortir de l'état le plus interne** pour sortir d'un état englobant

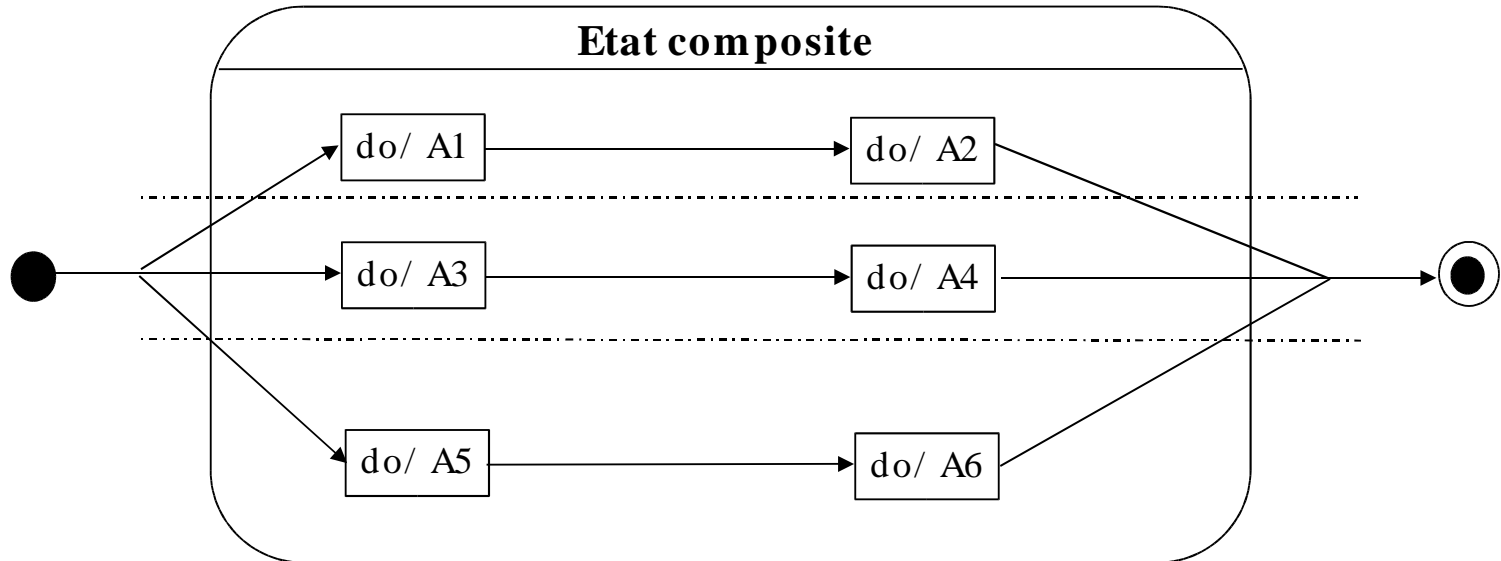


# Exercice

- Indiquez les séquences d'actions pour chaque événement reçu selon que l'automate est dans l'état B ou C



# Parallélisme et synchronisation



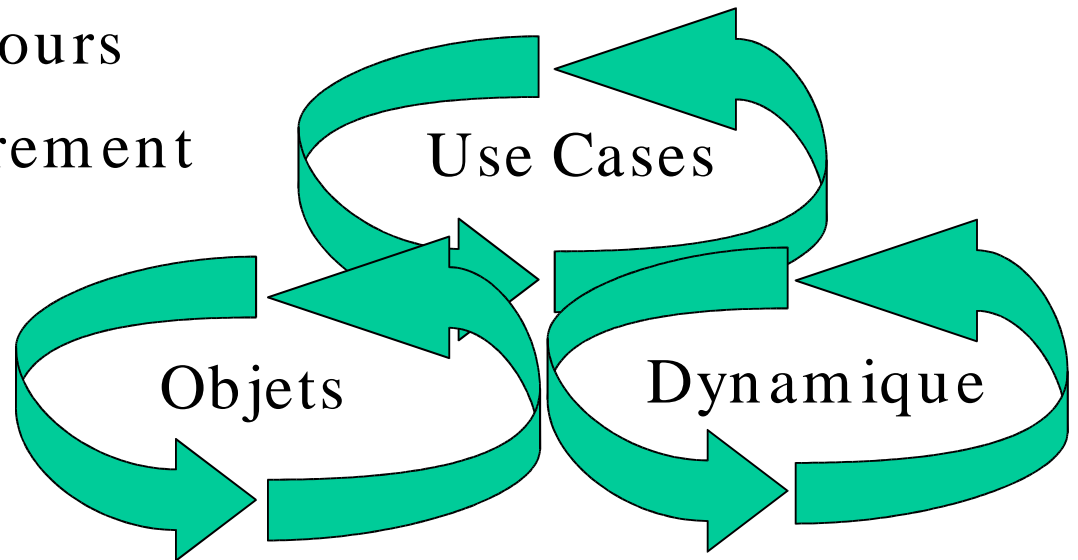
Activation de A1, A3 et A5  
en même temps

Transition quand A2, A4 et  
A6  
sont toutes finies

# *Démarche*

# *Démarche UML*

- Définir des vues sur le logiciel
- Utiliser la redondance entre les vues
- Travailler les différentes vues en parallèle
- Vérifier la cohérence à chaque étape
- Nombreux allers-retours
- Faire valider régulièrement



# *Interactions*

- Le découpage en packages doit être fait tôt, mais **nécessite des réaménagements**
- Diagramme des cas => différents cas (et retours)
- use case => diagrammes des classes (et retours)
- use case => diagrammes de séquences/ajouts de classes
- identifier les automates nécessaires
- diagrammes de séquences => automates (synthèse)

# Processus basique de conception

- Partir d'un modèle en analyse
  - partager le travail
  - détailler le contenu, les échanges
    - compléter les diagrammes de classes
    - compléter les diagrammes de séquence
  - identifier les doublons, les problèmes
  - modifier le partage (itérations)

## Figurer les interfaces

- Classes et méthodes d'interface
- Paramètres échangés, valeurs de retour
- Passage à l'implémentation

# *Formalisation du partage*

- Pour chaque entité identifiée lors de l'analyse, définir l'interface avec les équipes environnantes (contrat)
- Réaliser les diagrammes de séquences, nommer les méthodes, les paramètres, les valeurs de retour...
- Objectif : chaque équipe devient capable de travailler de façon autonome

# *Les interfaces comme contrat*

- Définir **toutes** les classes utiles (vu précédemment)
- Identifier les traitements qui impliquent plusieurs classes
- **Faire des diagrammes de séquences pour mettre en évidence les interactions**
- Définir la signature des méthodes, ce qu'elles font
- Identifier puis figer les interfaces **en figeant les signatures**
- Cela définit le contrat engageant chaque équipe

# *Conception détaillée locale*

- Pour chaque entité identifiée lors de l'analyse, conduire la conception détaillée en interne de chaque équipe
- Réaliser les diagrammes de classes, de séquences et éventuels automates détaillés
- Penser la généricité, la réutilisabilité
- Objectif : chaque équipe sait de façon détaillée **comment** elle va faire
- Engendrer les squelettes de code par équipe

# *Mise en commun de la conception*

- Pour chaque entité identifiée lors de l'analyse, validation définitive des interfaces avec les autres équipes
- Recherche des incohérences, redondances, conflits
- Compilation globale des squelettes engendrés par chaque équipe
- Objectif : validation globale de la conception

Tout le monde peut se mettre à coder

# *Conclusion sur UML*

- UML : un langage très riche
- La méthodologie n'est pas figée
- Les outils sont embryonnaires
- L'évolution va très vite (UML 2.0)
  - vers la vérification de cohérence (ToDoList)
  - vers la production automatisée de code
- suivre l'actualité sur le web
- Rien ne remplace l'expérience pour utiliser UML efficacement

# *Dynamique de l'élève*

