R3.04 cours n°2

I.Borne

- Les diagrammes d'états (modèle dynamique)
- Rappels sur les packages

Introduction

- Jusqu'à maintenant nous avons vu :
  - Comment décrire les spécifications d'un système en utilisant des cas d'utilisation
  - Comment modéliser la structure statique d'un système en utilisant un modèle de classes
  - Comment modéliser l'interaction entre les objets pour satisfaire les spécifications en utilisant des diagrammes d'interaction
- Nous n'avons pas vu comment modéliser la décision d'un objet sur ce qu'il doit faire quand il reçoit un message.
  - Le comportement d'un objet peut être affecté par les valeurs de ses attributs.
  - Quelles sont les dépendances entre l'*état* d'un objet et ses réactions aux messages et autres événements ?

Diagramme d'états (modèle dynamique)

- Il est lié au concept de machine à états finis.
- Il concerne le cycle de vie d'un objet générique d'une classe particulière au fil de ses interactions avec le reste du monde.
- Le diagramme d'états permet de modéliser le comportement d'un objet.
- Dans un diagramme d'états des événements sont reliés à des états en spécifiant la séquence d'états provoquée par une séquence d'événements.

Qu'est-ce qu'un état ?

- Un état représente une situation durant la vie d'un objet pendant laquelle :
  - il satisfait une certaine condition
  - il exécute une certaine activité
  - ou bien il attend un certain événement.
- Un état a une durée finie

Les événements en UML (1)

- La réception d'un signal envoyé par un autre objet ou par un acteur

(ex : une exception), en général asynchrone.

- L'appel d'une opération *(call event)* sur l'objet récepteur, en général synchrone.

nom-événement '(' liste-paramètres ')'
nom-paramètre ':' type-paramètre

Les événements en UML(2)

- Le passage du temps *(time event)*, modélisé avec `after` suivi d'une expression représentant une durée, décomptée à partir de l'entrée dans l'état courant.

after (une-durée)

ex : after(10 secondes)
after (10 secondes après l'entrée dans l'état A)

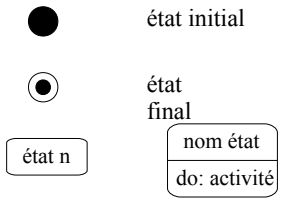
- Un changement dans la satisfaction d'une condition, modélisé par `when` suivi d'une expression booléenne.

when (condition-booléenne)

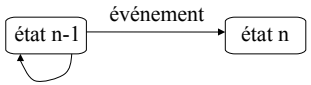
ex: when (date=1 janvier 2008)

Diagramme d'états (syntaxe graphique)

- États



- Transitions : permettent de passer d'un état à un autre ou de rester dans le même état



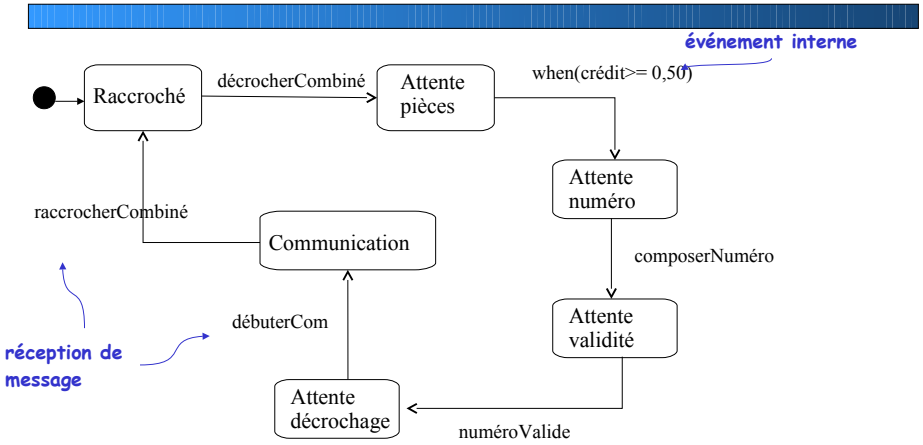
Un premier exemple : État nominal d'un publiphone

Il s'agit de modéliser les états d'un téléphone public.
Le fonctionnement du téléphone est simple :

- On décroche le combiné, on insère des pièces pour avoir la tonalité
- on compose le numéro et on attend que cela sonne chez votre correspondant
- le correspondant décroche et la communication est établie.
- à la fin on raccroche

Maintenant nous nous intéressons aux différents états qui vont caractériser ce téléphone et aux événements simples correspondants aux actions de l'utilisateur qui font passer d'un état à un autre.
Le diagramme nominal illustre cette première étape.

État nominal d'un publiphone



Comment trouver les états ?

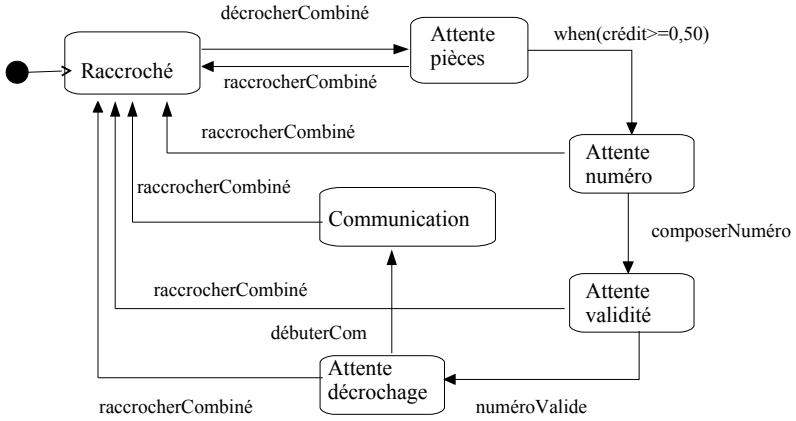
- Intuition, expertise métier : vocabulaire
- Étude des attributs et des associations de classe : valeur seuil, comportement induit par l'existence ou l'absence de certains liens.
- Démarche systématique : classe par classe, chercher le diagramme d'interaction le plus représentatif du comportement des instances et associer un état à chaque intervalle entre événements émis ou reçus.

Élaboration incrémentale des diagrammes d'états

- Comportement nominal : séquence d'états de sa naissance à sa mort avec les transitions associées.
- Ajouter progressivement les transitions correspondant aux comportements alternatifs.
- Intégrer les comportements d'erreur.
- Compléter les actions sur les transitions et les activités dans les états.
- Structurer en sous-états si le diagramme est devenu complexe.

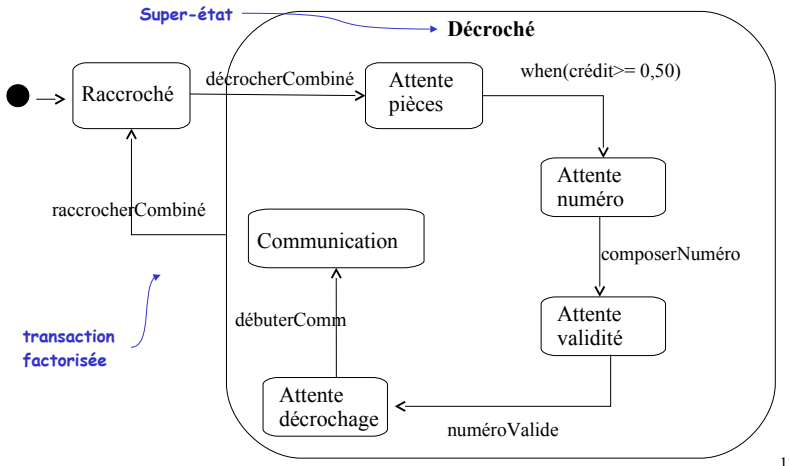
L'appelant peut raccrocher à tout moment

Ajout des transitions « raccrocherCombiné »

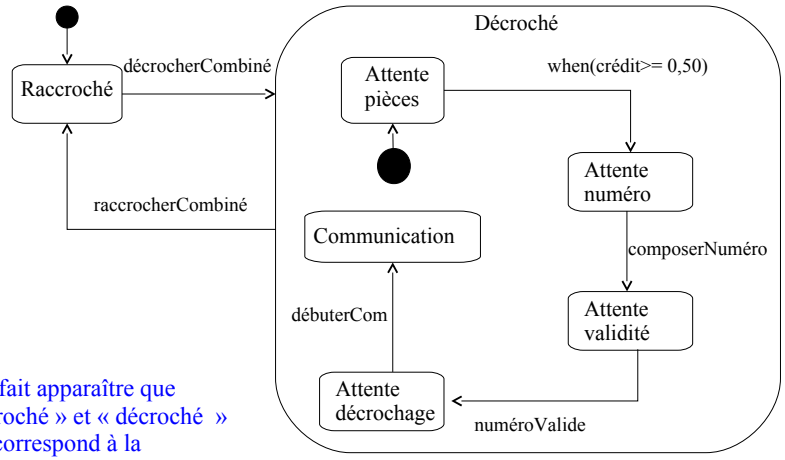


Insertion d'un super-état pour améliorer la lisibilité

On peut insérer un super état pour factoriser des transitions.
Ici on indique qu'en sortie de chaque état du super-état on a une transition vers l'état raccroché



Sous-état initial



- 1er niveau ne fait apparaître que les états « raccroché » et « décroché »
- 2ème niveau correspond à la décomposition de l'état « décroché »

Explications du sous-état initial

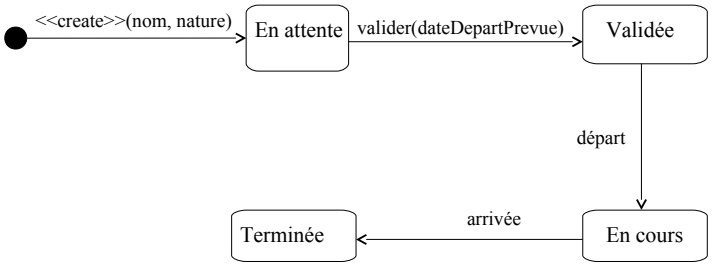
- Au lieu d'avoir directement une transition entre « raccroché » et « attente pièces » nous obtenons une première transition ente « raccroché » et « décroché »
- puis le symbole graphique de l'état initial intérieur de « décroché », afin d'indiquer explicitement le sous-état initial.

- Cette manière permet de découper le diagramme d'états en deux niveaux :

1er niveau ne faisant apparaître que les états « raccroché » et « décroché »
2ème niveau correspond à la décomposition du « décroché »

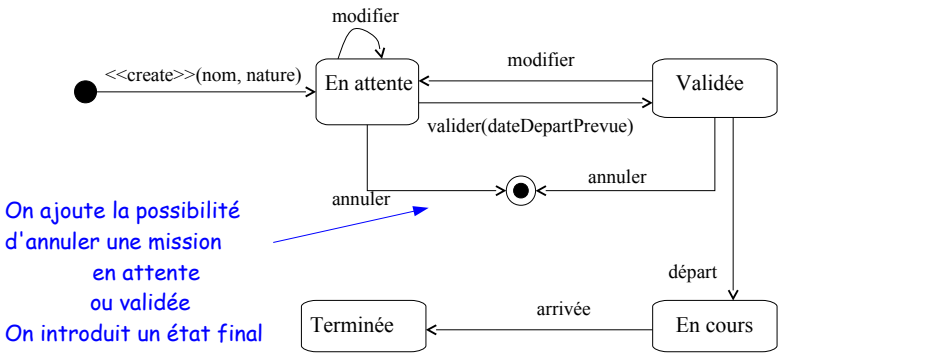
Deuxième exemple : la classe Mission

Transitions nominales



En attente : de la création d'une instance de mission à sa validation
Validée : les affectations se sont bien passées et le répartiteur valide la mission qui attend alors son départ effectif.

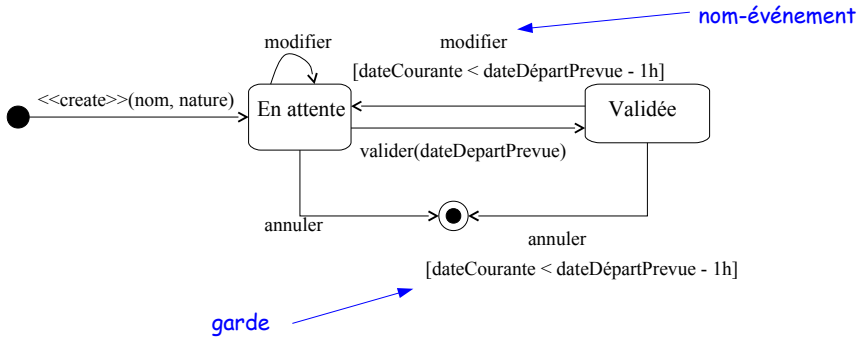
Classe Mission : transitions alternatives



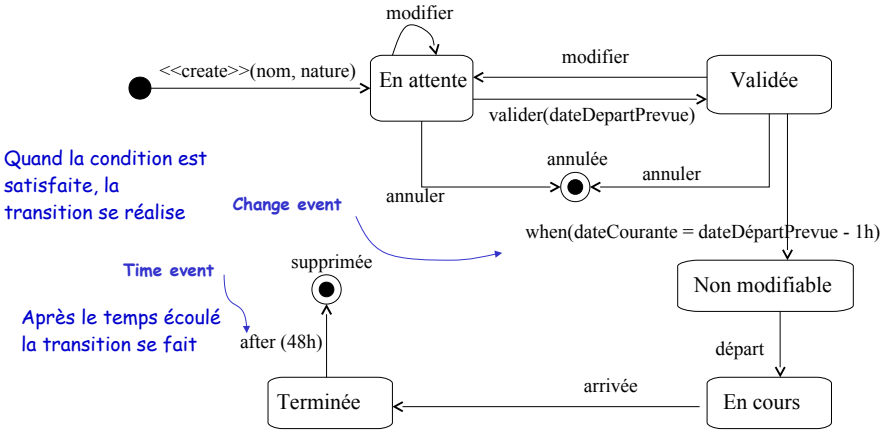
Classe Mission : condition de garde

Syntaxe complète d'une transition :

nom-événement '('paramètres ')' '[' garde ']' '/' activité



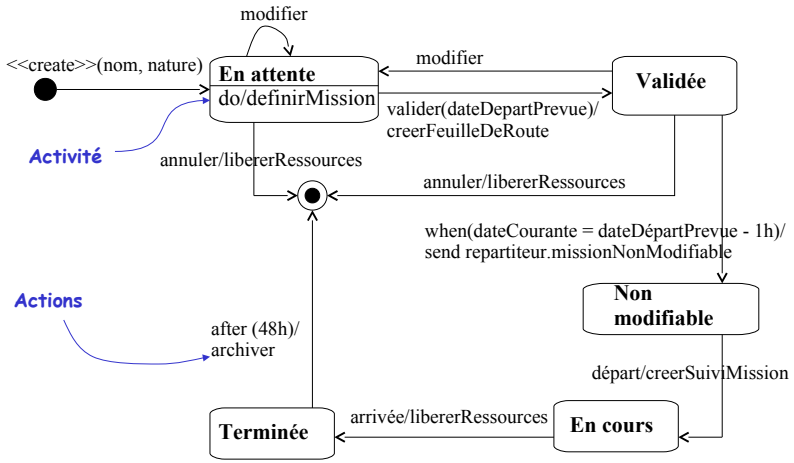
Classe Mission : 1er diagramme d'états



Explication du diagramme précédent

- Pour éviter de dupliquer la condition [dateCourante< datePrevue - 1h], il est également possible de la remplacer par un état intermédiaire entre les états *Validée* et *EnCours*.
- Si nous voulons exprimer le fait qu'au bout de 48h une mission terminée est archivée et supprimée de la mémoire vive du système, que doit-on faire ?
  - On utilise la notion de time event (passage du temps) avec le mot-clé *after*.
- Il est possible d'indiquer plusieurs état finaux avec des noms différents afin d'augmenter la lisibilité du diagramme, ou bien pour distinguer des manières différentes de détruire un objet.

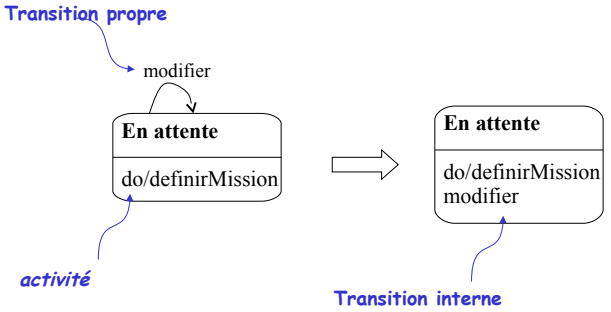
Classe Mission : ajout d'actions et d'activités



Transition propre et transition interne

Une transition propre ou une transition interne ne fait pas changer d'état

Dans l'exemple les deux notations sont équivalentes.



Transition sans déclencheur

- La transition est déclenchée automatiquement quand l'état source a terminé son activité

Ici quand l'activité de l'état « EnRecherche » est terminé l'objet passe directement à l'état « Prêt »

Précisions sur les actions

Les actions propres peuvent être documentées à l'intérieur de l'état :

- Pour une action qui sera exécutée à chaque entrée dans l'état, on utilise l'événement spécial `entry`

```
entry / action
```

- Pour une action qui sera exécutée à chaque sortie de l'état, on utilise l'événement spécial `exit`

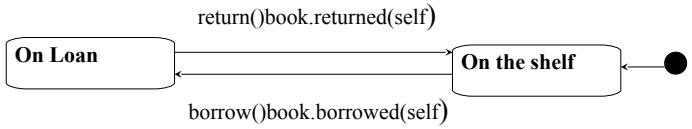
```
exit / action
```

- Pour une action récurrente qui sera exécutée dans l'état on utilise l'événement `do`

```
do / action
```

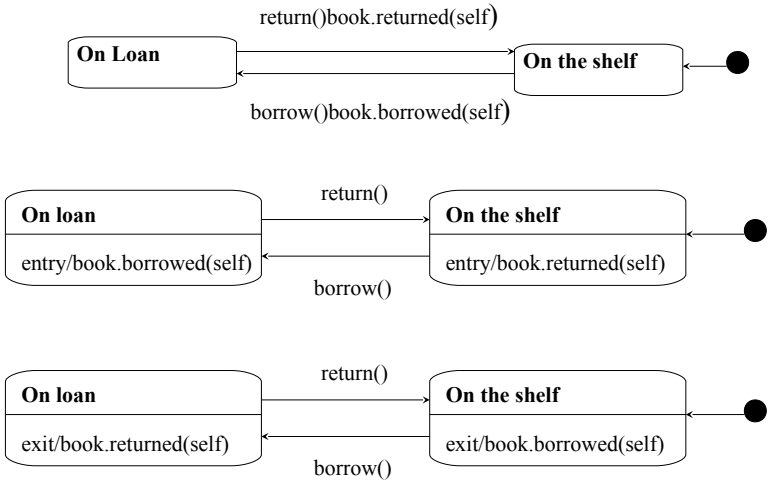
Les clauses `entry` et `exit`

On veut exprimer qu'un livre a deux états : en prêt ou sur l'étagère de la bibliothèque



Dans la page suivante on voit comment utiliser « entry » et « exit » pour exprimer la même chose

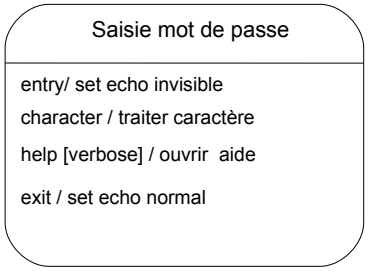
Les clauses `entry` et `exit`



État et transition interne

Syntaxe d'une transition interne :

nom-événement '(' liste-paramètres ')' '[' garde ']' '/' activité-à-réaliser



Diagrammes d'états : événements et signaux

- Qui déclenche une transition d'état ?
- Les événements peuvent comporter
  - des signaux (asynchrones)
  - des appels (synchrones)
  - l'écoulement du temps (asynchrone)
  - ou un changement d'état (asynchrone)
- La modélisation des événements permet la modélisation des processus et des « threads ».

Dimension dynamique

- Pour un système parfaitement statique ?
  - Un système parfaitement statique n'est pas intéressant car rien ne se produit jamais.
- Tous les systèmes comportent-ils une dimension dynamique ?
  - A priori oui.
- Par quoi est déclenchée la dynamique ?
  - La dynamique est déclenchée par des faits qui se produisent à l'extérieur et à l'intérieur.

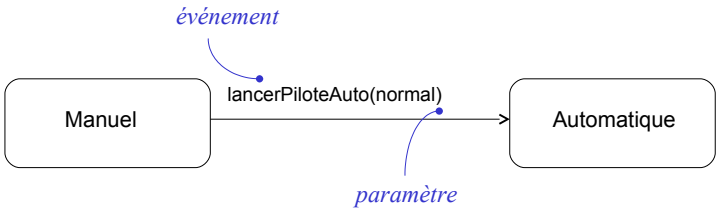
Sortes d'événements

- Les événements peuvent être externes ou internes.
- Les événements externes se produisent entre le système et les acteurs.
- Les événements internes surviennent parmi les objets qui existent à l'intérieur du système.

Événement appel

- Un événement *appel* représente le déclenchement d'une opération.
- Lorsqu'un objet invoque une opération sur un autre objet, le contrôle passe de l'expéditeur au destinataire.

Dans un automate à états finis, la transition est déclenchée par l'événement, l'opération est exécutée, le destinataire transite vers un autre état et le contrôle revient à l'expéditeur.

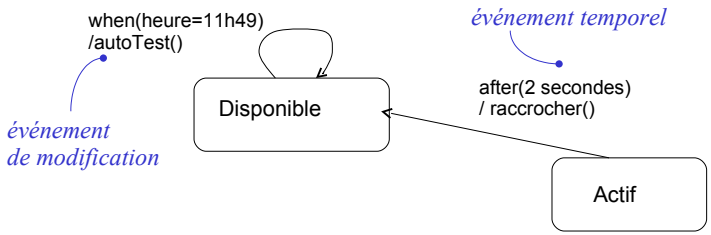


Événement temporel et de modification

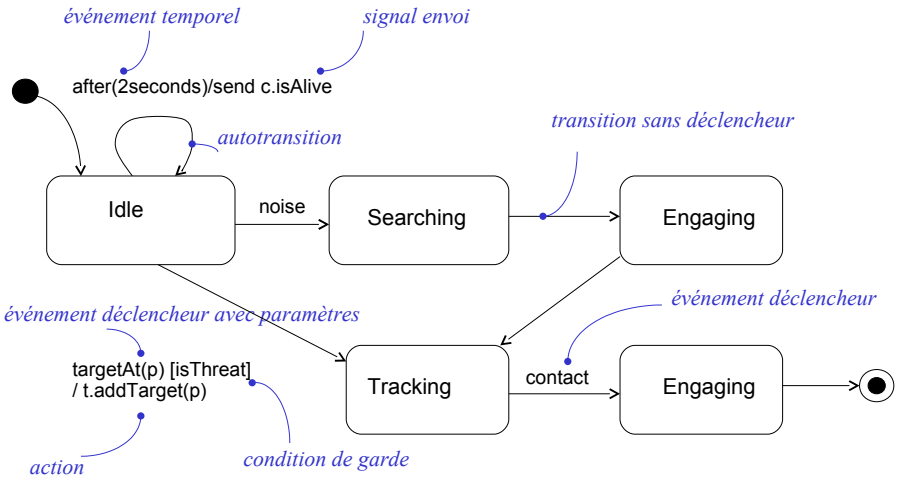
Un événement temporel représente l'écoulement du temps : on utilise le mot clé `after` suivi d'une expression qui correspond à une durée.

`after(2 secondes)` depuis la fin de l'état Disponible

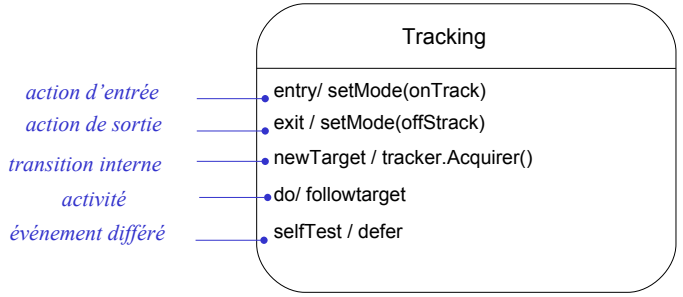
Un événement de modification représente un changement d'état ou le fait de satisfaire une condition : on utilise le mot clé `when` suivi d'une expression booléenne.



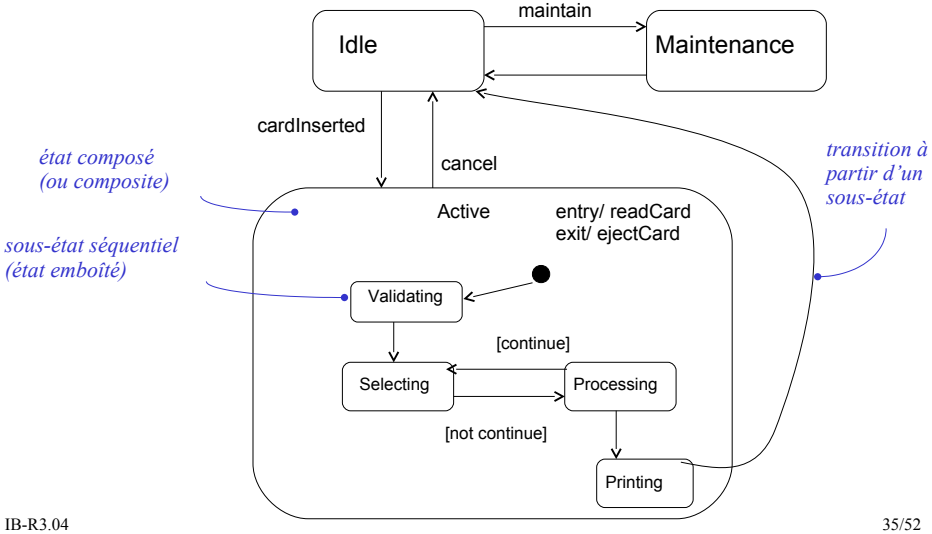
Synthèse des transitions (exemple de guidage de missile)



Événements et transitions avancées



Sous-états séquentiels : exemple d'ATM (guichet bancaire automatique)



Explications du schéma précédent

L'état de l'ATM change de `idle` à `Active` quand un client insère sa carte de crédit dans la machine.

L'état `Active` est structuré en sous-états

Quand on entre dans l'état `Active` la carte est lue (clause entry)

Après l'état `Processing` le contrôle peut retourner à `Selecting` (si le client a choisi une autre transaction) ou bien il peut aller à `Printing`.

Après `Printing` il y a une transition sans déclencheur qui retourne à l'état `Idle`.

A la sortie de l'état `Active` la carte est éjectée (clause exit).

Différences entre sous-états séquentiels et les sous-états concurrents

- Si on a 2 sous-états séquentiels ou plus sur le même niveau, l'objet se trouve dans l'un ou l'autre de ces sous-états.
- Si on a 2 sous-états concurrents ou plus sur le même niveau, l'objet se trouve dans un état séquentiel de chacun des sous-états concurrents.

L'exécution des sous-états concurrents se poursuit en parallèle.
A la fin, chaque automate à états finis emboîtés atteint son état final.

Sous-états concurrents

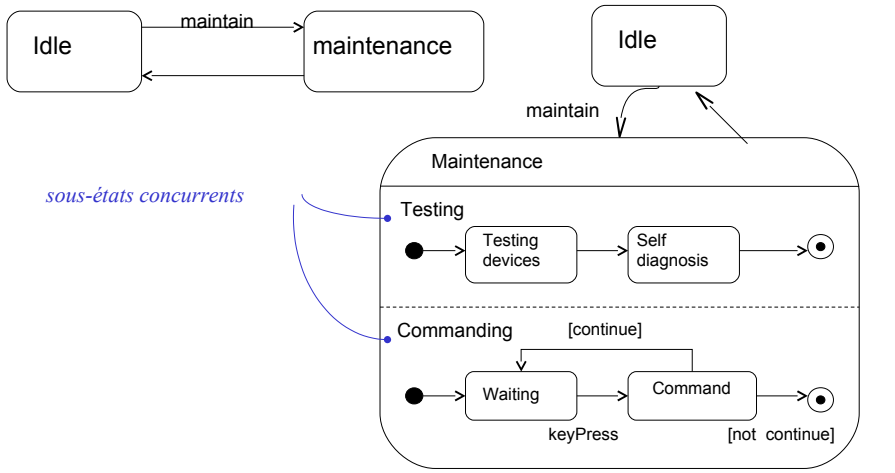
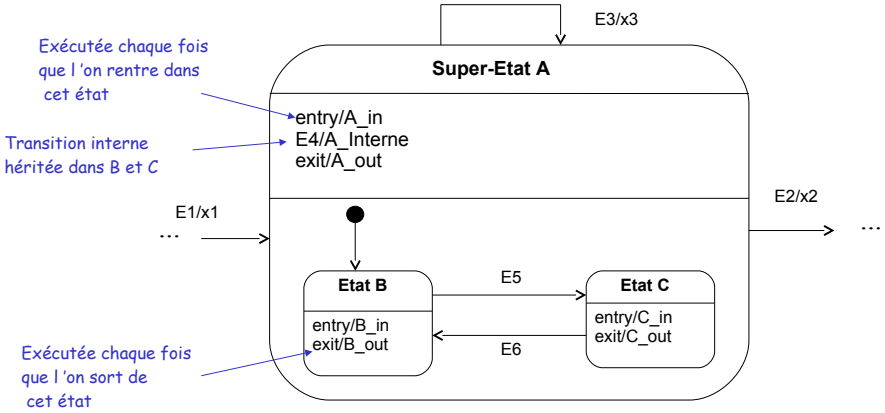
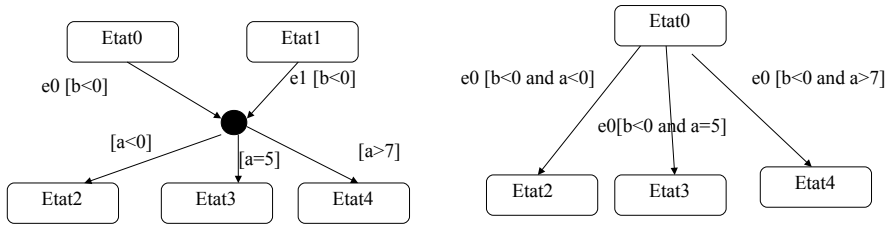


Diagramme d'états hiérarchiques



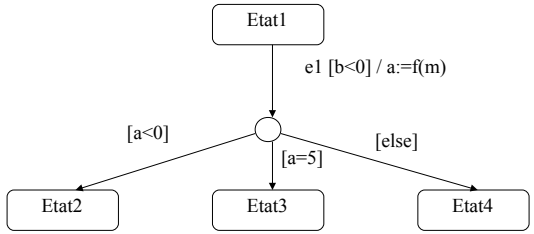
Point de jonction (notation supplémentaire)

- Deux ou plusieurs gardes émanant d'un même point de jonction représentent un point de branchement statique. Normalement les gardes sont mutuellement exclusives et c'est équivalent à un ensemble de transitions individuelles.
- Il y a une équivalence de représentation avec des transitions gardées.



Point de choix dynamique

- Deux ou plusieurs gardes émanant d'un même point de choix dynamique sont utilisées pour modéliser des choix dynamiques (calculés par une transition précédente) .
- Les gardes des transitions sortantes sont évaluées au moment où le point de choix est atteint.



Conclusions sur les diagrammes d'états-transitions

- Pratique pour représenter le comportement interne d'un objet.
- Bien adapté au génie logiciel basé sur les objets en raison des mécanismes qu'ils proposent et qui permettent de faire le lien avec les autres diagrammes de la norme.
- Certains outils permettent la génération automatique de programmes.

A méditer

- Quelles sont les différences entre diagramme d'états et diagramme de séquence ?
- A quelle vue de modélisation se rapporte un diagramme d'états ?

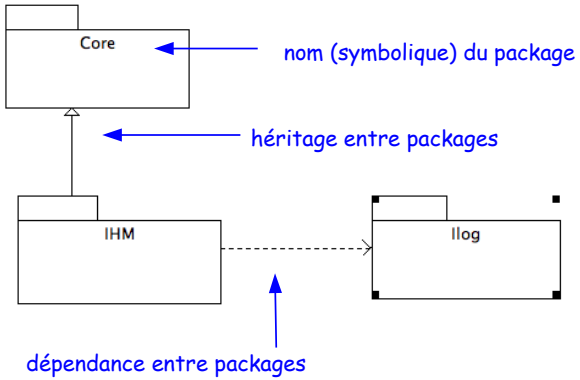
L'importance de l'architecture

- L'architecture est un ensemble de décisions sur:
  - Organisation du système logiciel
  - Sélection des éléments structurels et leurs interfaces de composition
  - Comportement
  - Composition des éléments structurels et comportementaux en des sous-systèmes de plus en plus gros
  - Style d'architecture qui guide cette organisation : les éléments statiques et dynamiques et leurs interfaces, leurs collaborations et compositions.

Structuration logique : les packages

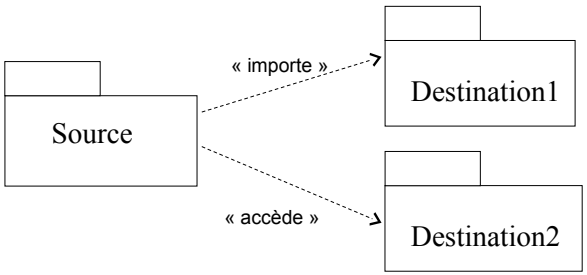
- Organisation des grands systèmes.
- Regroupement des éléments de modélisation selon des critères purement logiques.
- Structuration d'un système en catégories (vue logique) et sous-systèmes (vue des composants).
- Briques de base dans la construction d'une architecture
- Bon niveau de granularité pour la réutilisation.
- Ce sont aussi des espaces de noms

Diagramme de package et relations entre packages

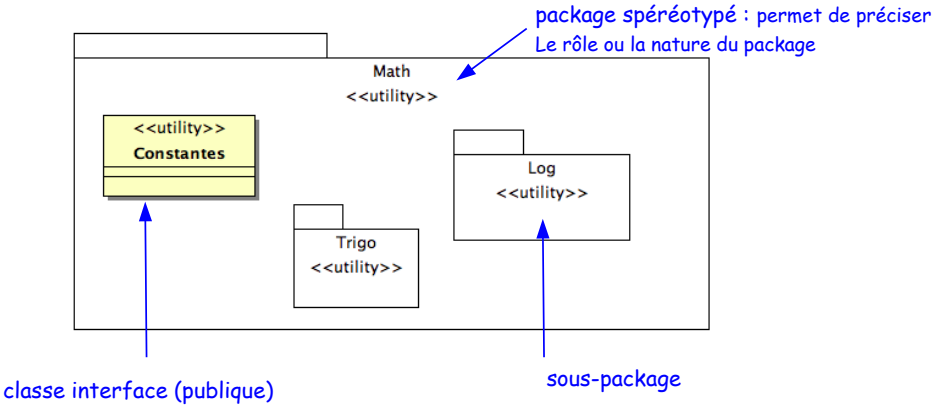


Stéréotypes de dépendances : importation et accès

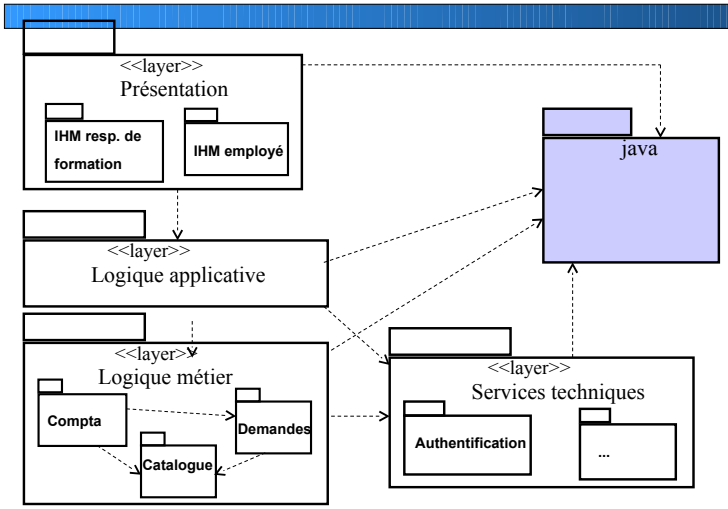
- La dépendance « importe » ajoute les éléments du package destination à l'espace de nommage défini par le package source
- La dépendance « accède » permet de référencer des éléments du package destination.



Package stéréotypé



Package et architecture en couches



Correspondances entre les diagrammes UML et Java(1) à compléter

Diagramme UML	Elément spécifique	Contrepartie Java
Package	instance de	
Classe	méthodes attributs association agrégation dépendance	
Séquence	instance de messages	
Communication	instance de messages	

Correspondance entre les diagrammes UML et Java(2)

Diagramme UML	Elément spécifique	Contrepartie Java
Machine à états	Etat Action(transition) Activité (état) Evénement	
Activités	Activité Action Flot de contrôle	

Respect des règles d'écriture UML

- Diagramme de classes
  - Stéréotype / propriété
  - Classe abstraite, classe « final », interface, exception
  - Méthodes héritées d'une superclasse
  - Type de retour des méthodes (pas de void)
  - Association, cardinalité , rôle
  - Association multiple, cardinalité, rôle
  - Association/Aggrégation
  - Réalisation d'une interface
  - Attribut statique, méthode statique
  - Attribut, méthode « final »
  - Dépendance (avec stéréotypes : local, paramètre, retour)