

Diagrammes dynamiques en UML

Interactions, Activités, Machines à états

Faculté Des Sciences, Université de Montpellier

19 novembre 2017

Modélisation d'un système en UML

UML permet de construire plusieurs vues sur un système :

- vue **structurelle**
 - diagrammes de classes, d'instances, de composants
- vue **dynamique**
 - diagrammes d'interactions, d'activités, machines à états
- vues **complémentaires**
 - diagrammes de cas d'utilisation
 - diagrammes de déploiement

Sommaire

1 Interactions

- Diagrammes de séquence
 - La ligne de vie
 - Les messages
 - Composition de fragments de diagrammes de séquence
- Diagrammes de communication
- Interaction overview
- Timing diagram

2 Les machines à états

- États et transitions
- États initial et final
- États composites
- Pseudo-états

3 Les diagrammes d'activités

Sommaire

1 Interactions

- Diagrammes de séquence
 - La ligne de vie
 - Les messages
 - Composition de fragments de diagrammes de séquence
- Diagrammes de communication
- Interaction overview
- Timing diagram

2 Les machines à états

- États et transitions
- États initial et final
- États composites
- Pseudo-états

3 Les diagrammes d'activités

Les diagrammes d'interaction

En UML 2.5, quatre diagrammes sont introduits pour représenter les interactions :

- diagramme de séquence (traces d'événements)
- diagramme de communication (messages échangés sur un diagramme d'instances)
- diagramme "Interaction overview" (proche des diagrammes d'activités)
- diagramme de timing (vision centrée sur les délais)

Nous présentons dans ce cours plus en détails les deux premiers (séquence, communication)

Les diagrammes de séquence

- Les diagrammes de séquence permettent de représenter les interactions entre des instances particulières. Un diagramme met en jeu :
 - des instances, et éventuellement des acteurs (dans le cadre d'un diagramme de cas d'utilisation),
 - des messages échangés par ces instances. Un message définit une communication entre instances. Ce peut être par exemple l'émission d'un signal, ou l'appel d'une opération.
- Le diagramme de séquence permet d'insister sur la **chronologie** des interactions : le temps s'écoule principalement du haut vers le bas.
- Le diagramme décrit un ensemble cohérent de messages, dans le cadre d'une **fonctionnalité du système**.

Exemple

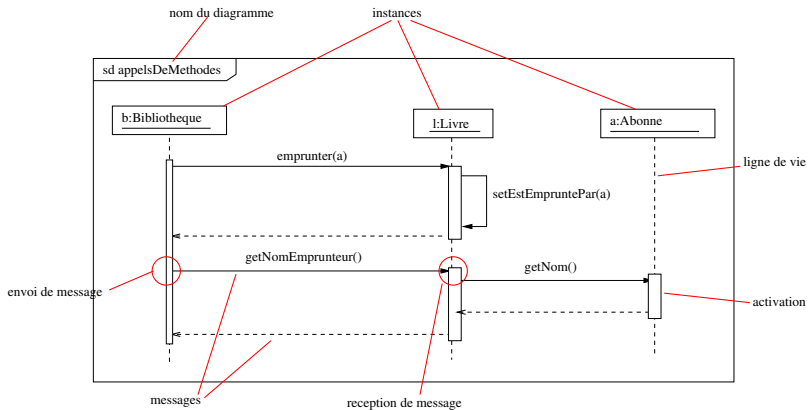


Figure : Diagramme de séquence pour une opération d'emprunt de livre

La ligne de vie

- À chaque instance est associée une ligne de vie, qui représente la vie de l'objet.
- Les événements survenant sur une ligne de vie (réception de message ou envoi de message) sont ordonnés chronologiquement.
- La ligne de vie est représentée par une ligne pointillée quand l'instance est inactive, et par une boîte blanche quand l'instance est active.
- Quand une instance est détruite, on stoppe la ligne de vie par une croix.

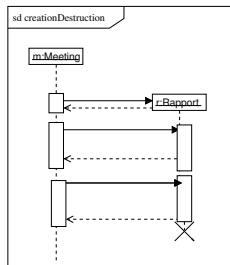


Figure : Ligne de vie

Les messages

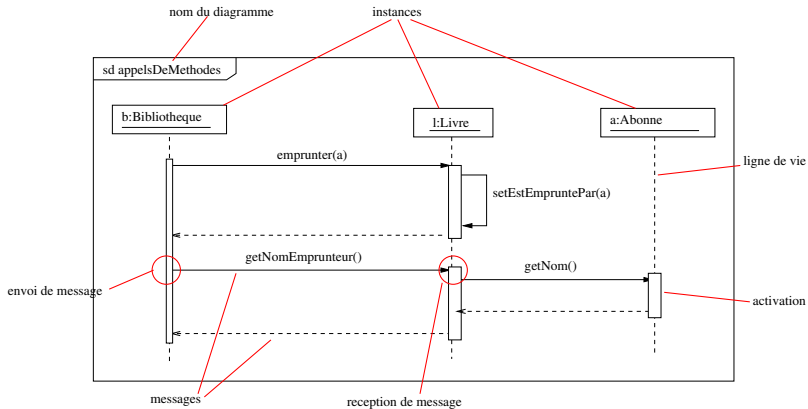
- Les messages sont représentés par des lignes fléchées.
- À chaque extrémité de la ligne fléchée correspond un événement (réception ou envoi).
- Le sens de la flèche permet de déterminer dans quel sens va le message.
- synchronie/asynchronie :

—————> message asynchrone

—————▶ appel synchrone

<----- retour synchrone

Exemple



Syntaxe des noms de message

La syntaxe pour le nom d'un message est :

```
([attribut =] signal-ou-NomOperation ([ liste-arguments ]))[:  
valeur-retour] | *
```

où la syntaxe pour un argument est :

```
([nomParam =] valeur-argument) | (attribut = nomParamOut [:  
valeurArgument]) | -
```

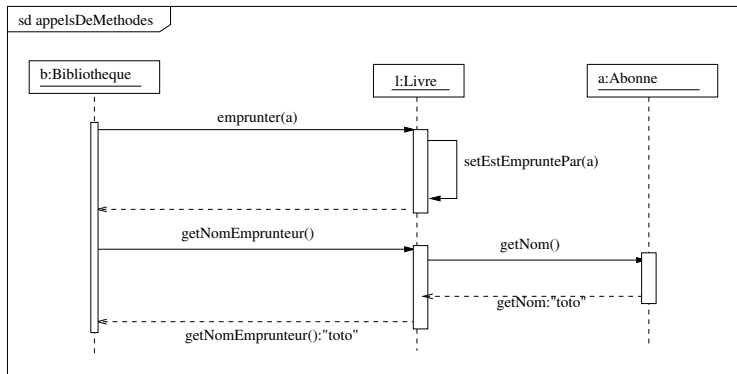
* signifie : n'importe quel type de message

- signifie : paramètre indéfini

Par exemple, on peut avoir les noms de message suivants :

- `getAge()`
- `getAge() :12`
- `age=getAge() :12`
- `setAge(age=15)`
- `setAge(-)`

Exemple d'appel de méthode



Les diagrammes de séquence sont à concevoir conjointement avec les autres diagrammes, comme par exemple avec le diagramme de classes

Composition de fragments de diagrammes de séquence

- possible depuis la version 2.0 d'UML
- plusieurs opérateurs de composition

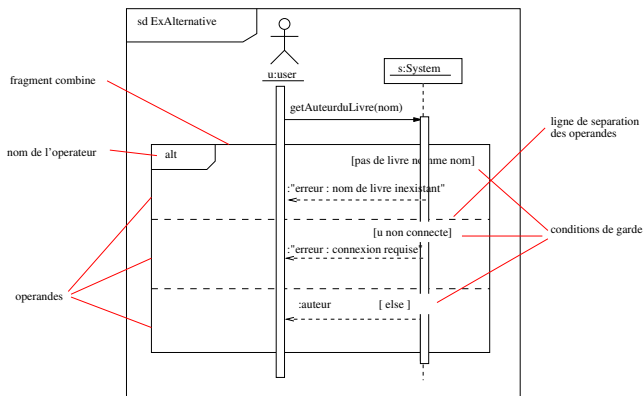
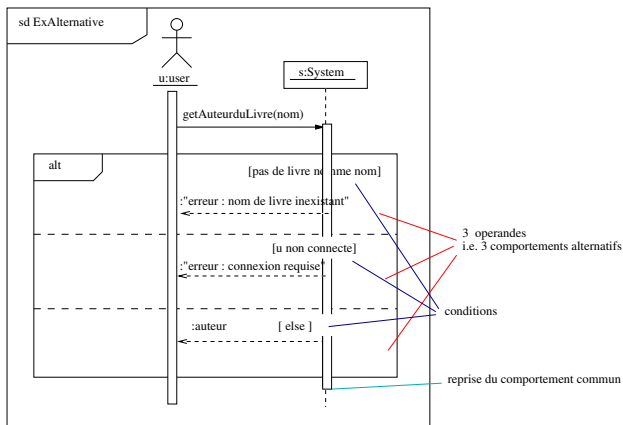


Figure : Opérateurs de composition et fragments combinés

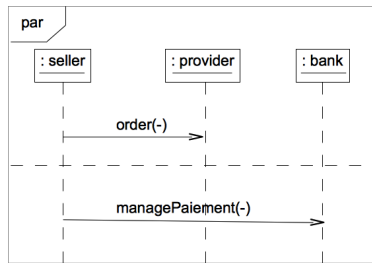
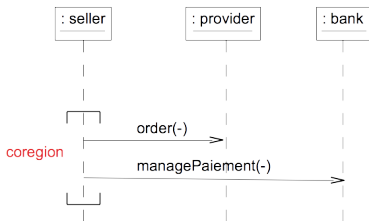
Alternative et optionnalité

- alternative (noté alt) permet de représenter le choix (exclusif) entre plusieurs comportements
- optionnalité (noté opt) permet de représenter un comportement qui n'a lieu que si une condition de garde est vraie



Composition parallèle

- L'opérateur de composition parallèle (noté **par**) permet de spécifier des comportements qui peuvent avoir lieu en parallèle les uns des autres.
- Quand un comportement A est en parallèle avec un comportement B, l'ordre partiel des événements de A et de B est conservé.
- Raccourci syntaxique : *corégion*



Composition séquentielle faible

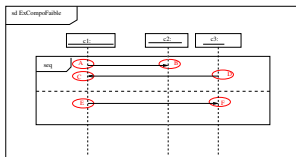


Figure : Composition séquentielle faible (seq)

- $A \prec B$
- $D \prec C$
- $E \prec F$
- $C \prec E$
- $D \prec F$

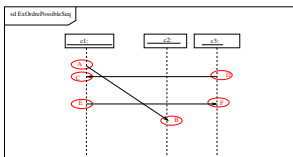


Figure : Un diagramme de séquence pouvant en résulter

Composition séquentielle forte

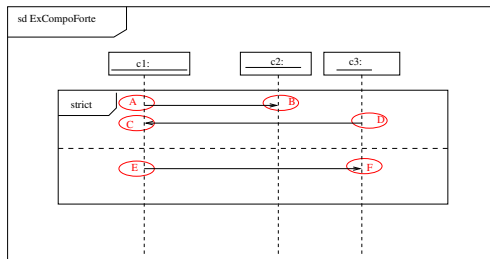


Figure : Composition séquentielle forte (**strict**)

- $A \prec B$
- $D \prec C$
- $E \prec F$
- $C \prec E$
- $D \prec F$
- $B \prec E$

Boucle

L'opérateur **loop** permet d'itérer des comportements.

On doit pour cela spécifier :

- le nombre minimum `minInt` de tours de boucles,
- le nombre maximum `maxInt` de tours de boucle (* signifie infini),
- une condition de garde,
- une unique opérande représentant le comportement sur lequel on boucle.

Syntaxe de la boucle :

```
loop[ (minInt [ , maxInt ] ) ]
```

Par défaut, `minInt=0` et `maxInt=*`..

Exemple de boucle

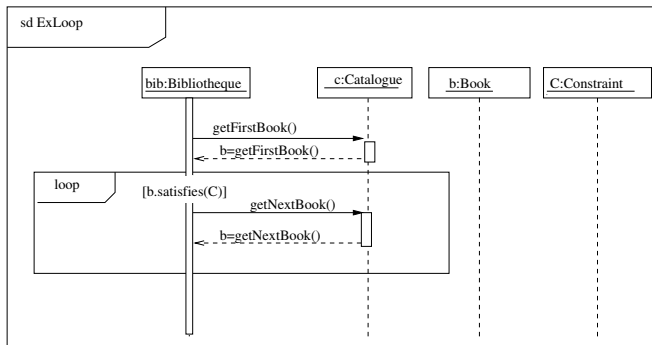


Figure : Boucles dans les diagrammes de séquence

Les diagrammes de communication

- basés sur un diagramme d'instances
- montrent les échanges de messages entre instances
- les messages sont numérotés de manière arborescente

Les diagrammes de communication

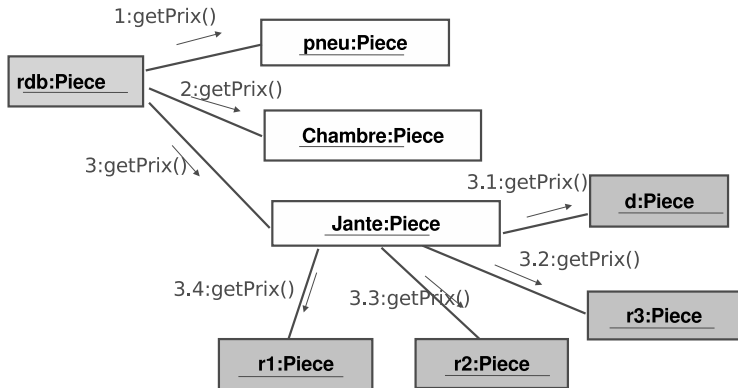
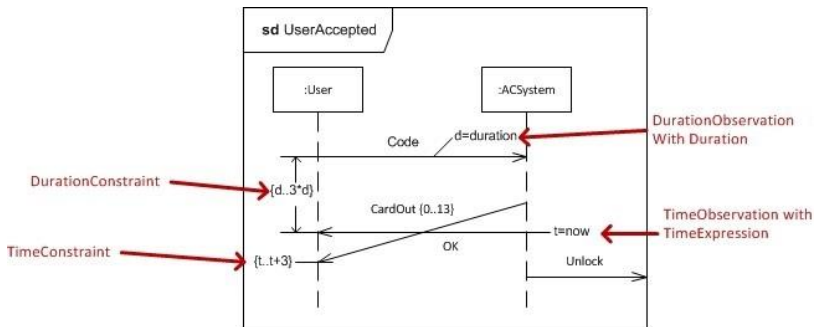


Figure : Diagramme de communication pour le calcul du prix d'une pièce composée (roue de brouette)

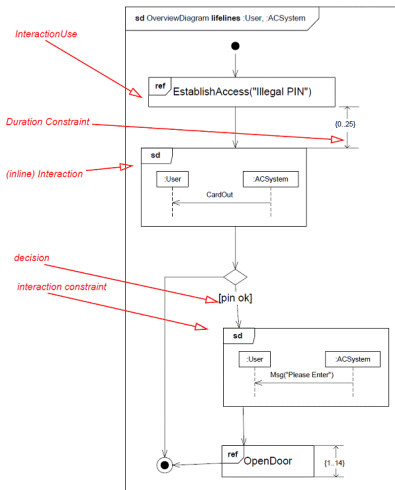
Timing in sequence diagram

Diagramme de séquence pour une ouverture de porte (OMG, UML2.5)



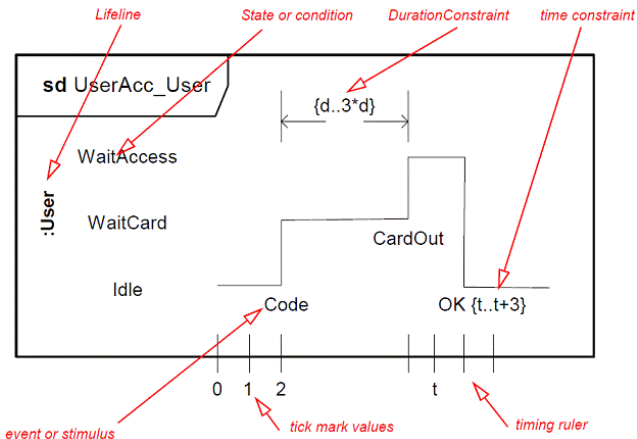
Interaction overview (séquences et activités)

Diagramme d'overview de l'interaction pour l'ouverture de porte (OMG, UML2.5)



Timing diagram

Diagramme de timing de l'interaction pour l'ouverture de porte (OMG, UML2.5)



Sommaire

1 Interactions

- Diagrammes de séquence
 - La ligne de vie
 - Les messages
 - Composition de fragments de diagrammes de séquence
- Diagrammes de communication
- Interaction overview
- Timing diagram

2 Les machines à états

- États et transitions
- États initial et final
- États composites
- Pseudo-états

3 Les diagrammes d'activités

Les machines à états

Les machines à états, aussi appelées diagrammes d'état-transition, servent à modéliser la dynamique d'un sous-système, souvent d'une classe.

Une machine à états associée à une classe décrit la dynamique de ses instances à la réception ou à l'envoi de messages

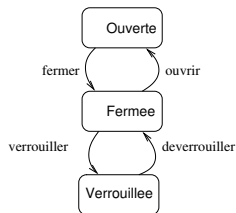


Figure : Diagramme d'état-transition pour une porte

États et transitions

- Un **état** modélise une situation où un certain invariant (généralement implicite) est maintenu
 - ex. la porte est fermée, un compte bancaire a un solde positif, ...
- Une **Transition** représente le passage d'un état à un autre
 - Il peut y avoir plusieurs événements déclencheurs possibles, auquel cas on les liste tous (en les séparant par des virgules).
 - L'action peut être une affectation d'attribut, un appel de méthode, la fin d'une période de temps...
 - Quand aucun événement déclencheur n'est spécifié, la transition est dite spontanée.

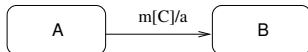


Figure : Une transition

États initial et final

Un **pseudo-état initial** représente un sommet qui est la source d'une seule transition vers l'état "par défaut" d'une machine à état ou d'un état composite. La transition initiale peut être munie d'une action.

L'**état final** matérialise le fait qu'une région (une machine à état ou une région d'état composite) est "terminée".

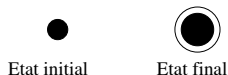


Figure : États initial et Final

États composites

Pour les diagrammes complexes, on peut utiliser la notion d'**état composite**.

Un **état composite** :

- soit contient une seule région
- soit se décompose en 2 ou plusieurs régions orthogonales

Un état inclus dans une région d'un état composite est appelé **sous-état** de cet état composite.

Exemple de machine à état avec état composite

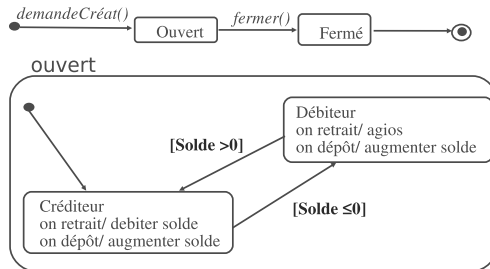


Figure : Exemple de machine à état avec état composite

Exemple avec état à régions orthogonales

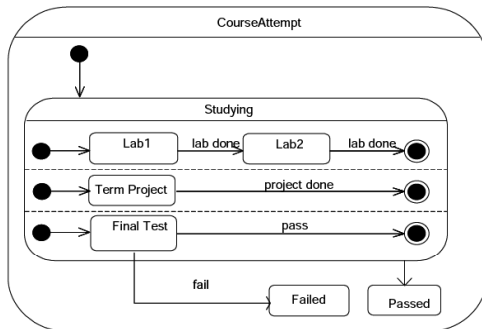


Figure : État composite orthogonal, extrait du document de spécification d'UML 2.0

Comportement d'entrée et de sortie, comportement dans un état

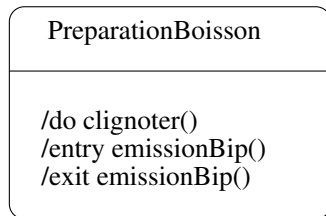


Figure : Actions d'entrée et de sortie des états, comportement dans les états

+ on evenement / action

États Historiques

Il existe des états dits “**mémoire**” qui permettent de réentrer dans un état composite dans le même sous-état que lors de la sortie.

Deux états mémoire :

Historique superficiel (Shallow history) (noté H). L'historique superficiel indique que l'on revient au sous-état actif le plus récent (mais pas dans un sous-état de ce sous-état).

Historique profond (Deep history) (noté H*). L'historique profond indique que l'on revient à la configuration active la plus récente de l'état composite qui contient directement l'historique profond (c'est-à-dire à la configuration active la dernière fois qu'on a quitté l'état composite, et précisément dans le dernier sous-sous-...-état).

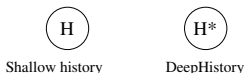


Figure : États ShallowHistory et DeepHistory

États Historiques

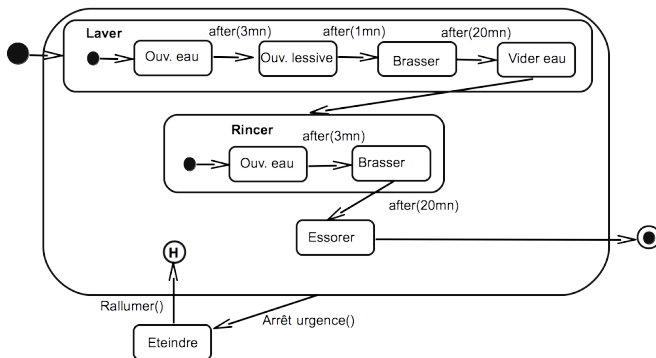


Figure : Machine à laver (avec historique superficiel)

États Historiques

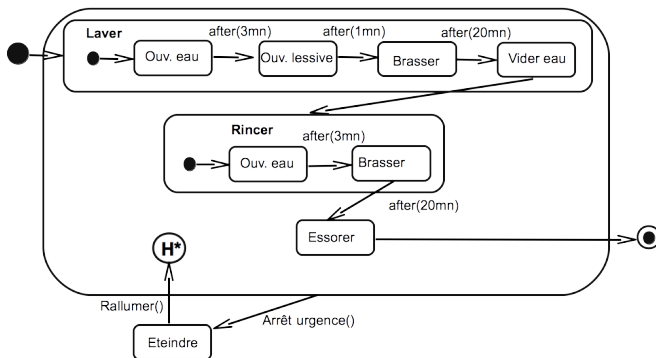


Figure : Machine à laver (avec historique profond)

Autres pseudo-états

Il existe d'autres pseudo-états comme les jonctions, les choix ou les branchements, nous ne les détaillerons pas ici (même esprit que dans les diagrammes d'activités).

Sommaire

1 Interactions

- Diagrammes de séquence
 - La ligne de vie
 - Les messages
 - Composition de fragments de diagrammes de séquence
- Diagrammes de communication
- Interaction overview
- Timing diagram

2 Les machines à états

- États et transitions
- États initial et final
- États composites
- Pseudo-états

3 Les diagrammes d'activités

Les diagrammes d'activité

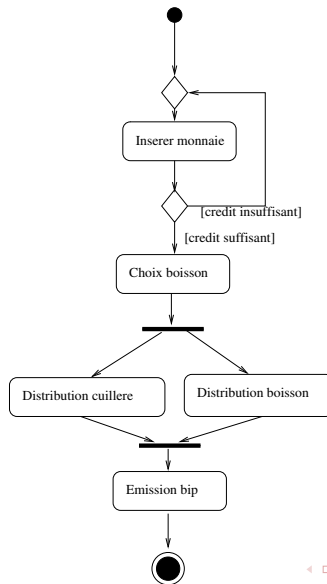
Diagramme d'activités :

- représentation de flots de contrôle et de données
- ex. comportement d'une opération ou d'un cas d'utilisation.
- graphes, avec différents types de nœuds et d'arcs.




Contenu :

- nœuds actions
- nœuds de contrôle permettant de spécifier l'enchaînement des actions (synchronisation, branchement, ...)
- nœuds d'objet permettant de représenter les objets créés ou utilisés au cours d'une activité
- arcs de transition permettant de relier les nœuds.



Exemple



Représentation de graphes de flot de contrôle

- Nœud initial (Initial node)  . Point d'entrée pour invoquer une activité. Un jeton de contrôle est placé au nœud initial quand l'activité commence.
- Nœud de fin d'activité (Activity final node)  . Stoppe tous les flots dans une activité. Un jeton atteignant un nœud de fin d'activité fait avorter tous les flots en cours, l'activité termine et le jeton est détruit (ainsi que tous les jetons circulant dans l'activité).
- Nœud de fin de flot (Flow Final node)  . Termine un flot. Le nœud de flot final détruit les jetons y entrant.

Représentation de graphes de flot de contrôle

- Nœud d'action (Action node) 
 - Unité fondamentale de la fonctionnalité exécutable d'une activité
 - Une action s'exécute quand toutes les contraintes sur ses *flots de contrôle* entrants sont satisfaites (jonction implicite).
 - L'exécution consomme les jetons de contrôle entrants puis présente un jeton sur chaque flot sortant (branchement implicite).
- Flot de contrôle (Control flow) 
 - Passage des jetons
 - Les jetons offerts par le nœud source sont offerts au nœud destination.

Représentation de graphes de flot de contrôle



- Nœud de décision (Decision node)

Choix parmi les flots sortants. Chaque jeton arrivant sur un nœud de décision ne peut traverser qu'un seul flot sortant. Les jetons ne sont pas dupliqués. Ce sont les gardes sur les flots sortants qui permettent le choix (les gardes doivent assurer le déterminisme du choix).



- Nœud de branchement (Fork node)

Partage d'un flot en flots concurrents. Les jetons arrivant d'un branchement sont dupliqués sur les flots sortants.

Représentation de graphes de flot de contrôle

- Nœud de jonction (Join node)



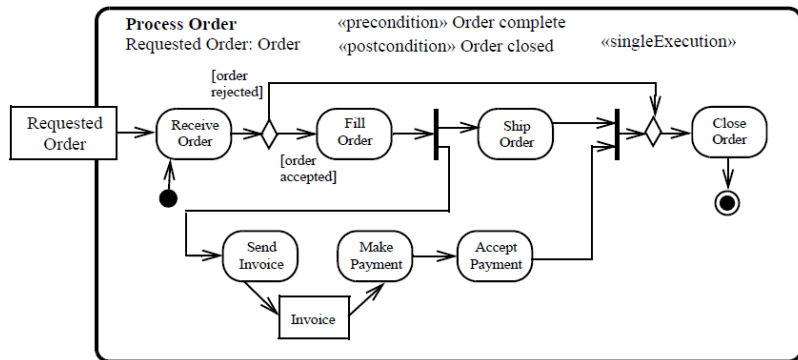
. Synchronisation de plusieurs flots. Si un jeton de contrôle est offert sur chaque flot entrant, alors un jeton de contrôle est offert sur le flot sortant.



- Nœud de fusion (Merge node) . Rassemblement de plusieurs flots. Tous les jetons offerts sur les flots entrants sont offerts sur le flot sortant sans synchronisation.
- Partition d'activité (Activity Partition). Identifie des actions ayant une caractéristique commune. Les partitions n'affectent pas le flot des jetons.



Objets dans les flots (boîtes rectangulaires)



Synthèse

Diagrammes dynamiques

- Interaction (4 types de diagrammes)
- Activité
- Diagrammes d'états

Particularités

- Généralement dans le contexte d'un diagramme structurel ou de cas d'utilisation
- Représente un ensemble cohérent d'actions, de messages
- Une complexité assez importante

Référence : <http://www.omg.org/spec/UML/2.5/>