Chapitre III.
Diagrammes des aspects dynamiques

Frédéric DADEAU

Département Informatique des Systèmes Complexes - FEMTO-ST

Bureau 410 C

Email : frederic.dadeau@univ-fcomte.fr

Modélisation et Conception Orientées Objet
Licence 3 – Année 2016-2017

Plan du cours

Diagrammes de cas d'utilisation

Diagrammes de séquence

Diagrammes de collaboration/communication

Diagrammes d'états-transitions

Diagrammes d'activité

Plan du cours

Diagrammes des cas d'utilisation

Les cas d'utilisation permettent de décrire le système du point de vue de l'utilisateur. Ils permettent de définir les limites du système et les relations entre le système et son environnement.

Identification des acteurs

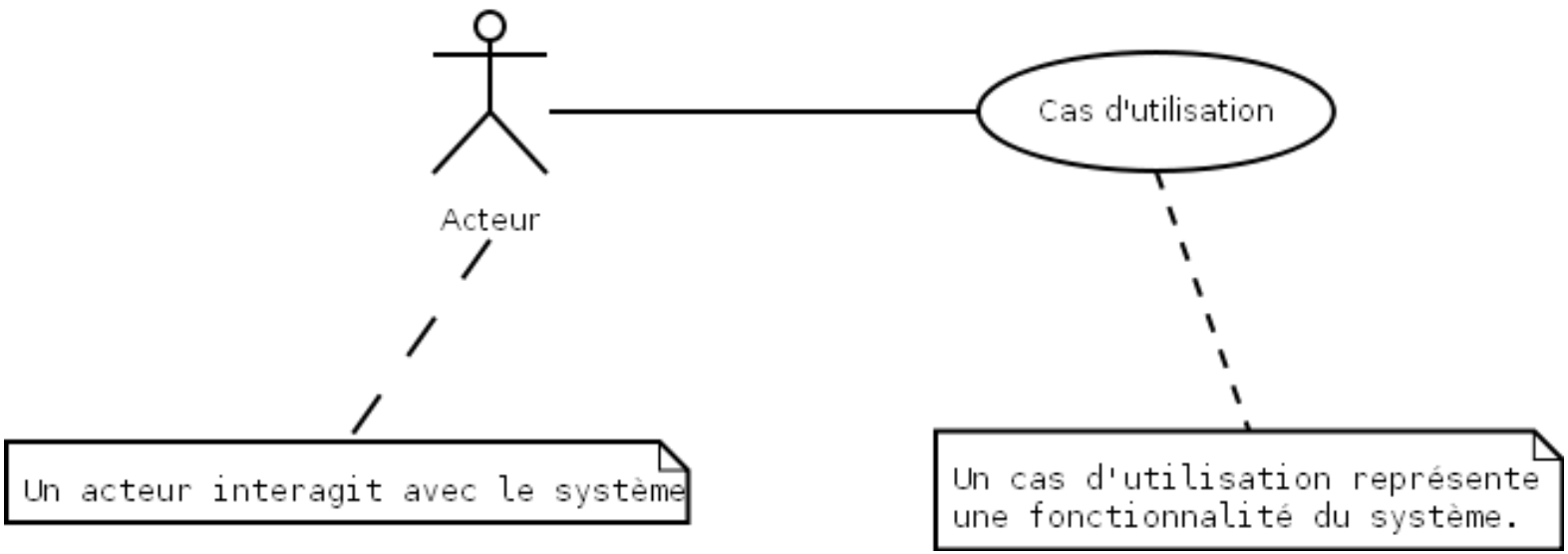
Un acteur représente un rôle joué par une personne ou une chose qui interagit avec le système.

N.B. : Une même personne physique peut correspondre à plusieurs acteurs si celle-ci peut jouer plusieurs rôles.

Acteurs et cas d'utilisation

Cas d'utilisation : fonctionnalité du système utilisée par un acteur.

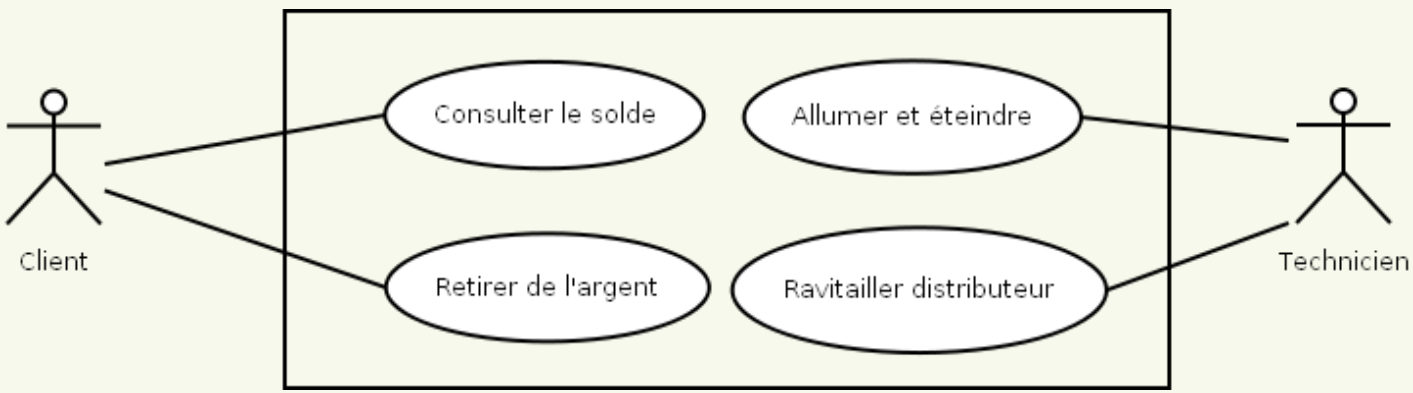
Un acteur qui interagit avec le système pour utiliser une fonctionnalité est représenté par une relation entre cet acteur et le cas d'utilisation représentant cette fonctionnalité. On dit que l'acteur *déclenche* le cas d'utilisation auquel il est lié.



Exemple de cas d'utilisation

Distributeur de billets

- Un premier acteur : "client", consulte le solde de son compte, ou retire de l'argent.
- Un second acteur : "technicien", allume, éteint ou ravitaille le distributeur.

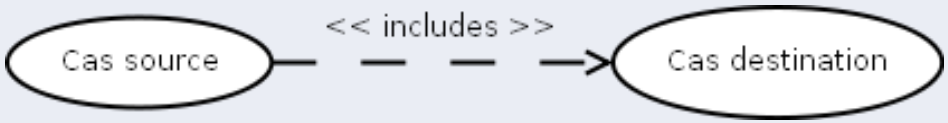


On note qu'un rectangle symbolise les limites du système.

Inclusion entre deux cas d'utilisation

Un cas d'utilisation, nommé *source*, **inclut** un cas d'utilisation nommé *destination*, si les comportements décrits par le cas source contiennent les comportements décrits par le cas destination : le cas source dépend du cas destination.
Lorsque le cas source est sollicité, le cas destination l'est obligatoirement comme une partie du cas source.

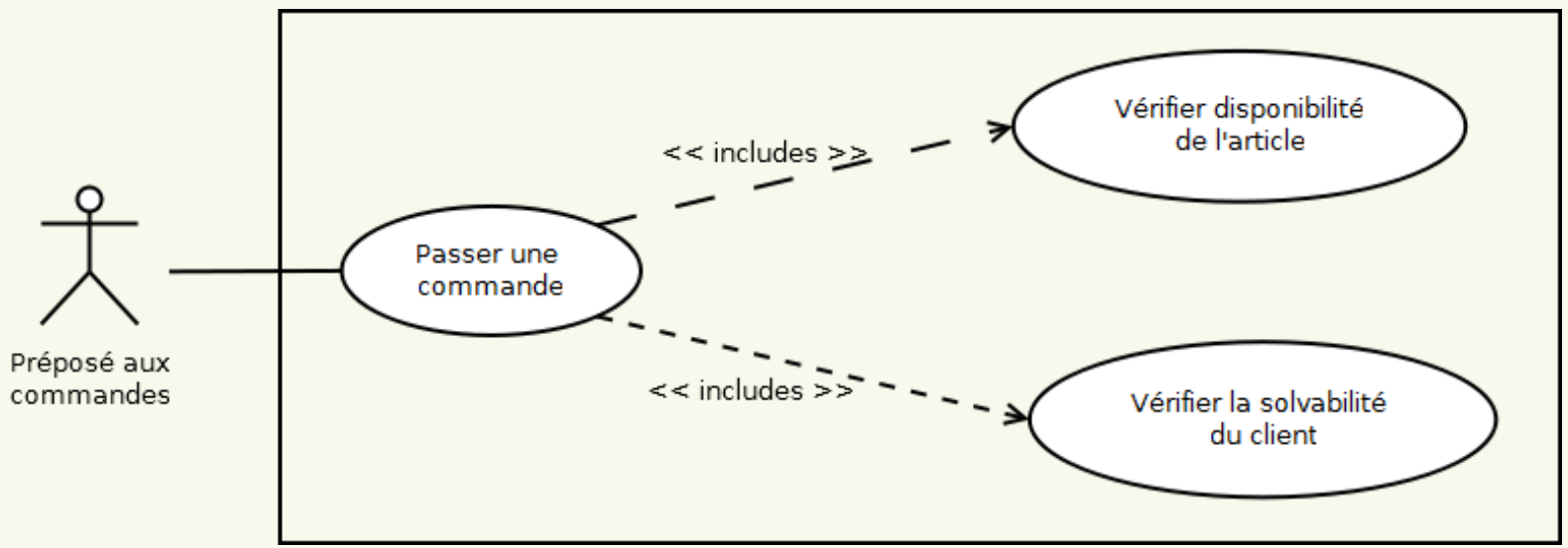
Notation de l'inclusion



Les inclusions sont utilisés en particulier pour factoriser une fonctionnalité partagée par plusieurs cas d'utilisation.
Elles permettent également de décomposer un cas complexe en sous-cas plus simples (sans spécifier l'ordre).

Inclusion entre deux cas d'utilisation

Exemple d'inclusion de cas d'utilisation



Extension d'un cas d'utilisation

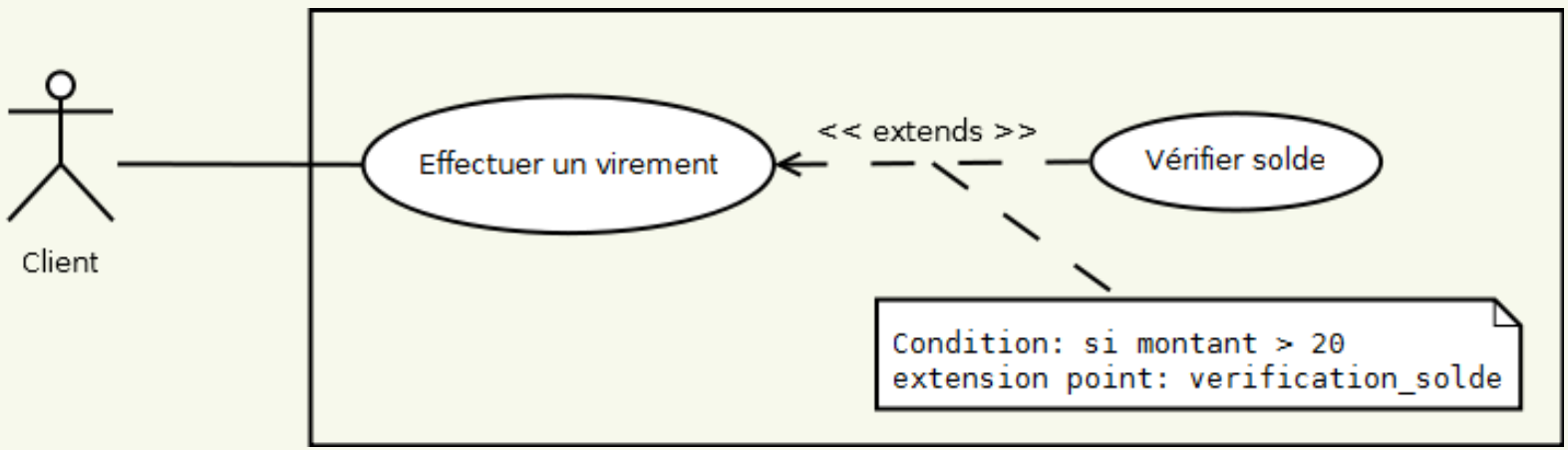
Un cas d'utilisation, nommé *source*, **étend** un cas d'utilisation nommé *destination*, lorsque le cas d'utilisation source peut-être appelé au cours de l'exécution du cas d'utilisation destination.
Exécuter le cas destination peut éventuellement entraîner l'exécution du cas source.
Contrairement à l'inclusion, l'extension est optionnelle.

Notation de l'extension

L'extension peut intervenir à un point précis du cas étendu appelé *point d'extension*. Il porte un nom, qui figure dans un compartiment du cas étendu sous la rubrique point d'extension, et est éventuellement associé à une contrainte indiquant le moment où l'extension intervient.

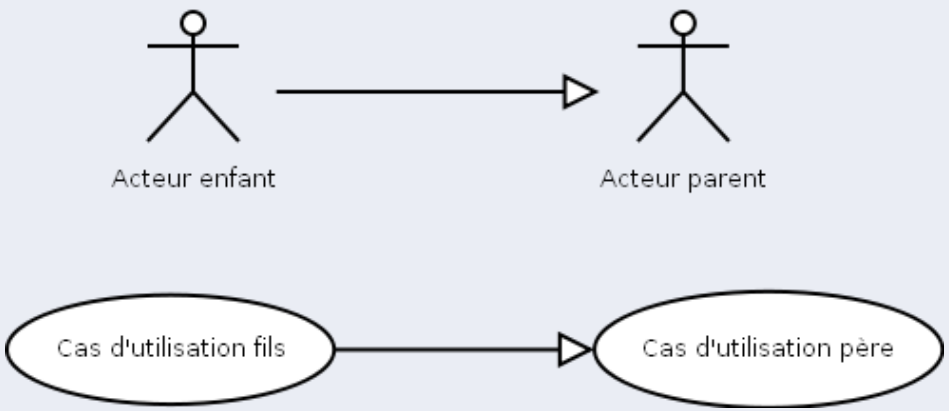
Extension d'un cas d'utilisation

Exemple d'extension d'un cas d'utilisation



Généralisation et spécialisation

Héritages possibles entre acteurs et entre cas d'utilisation



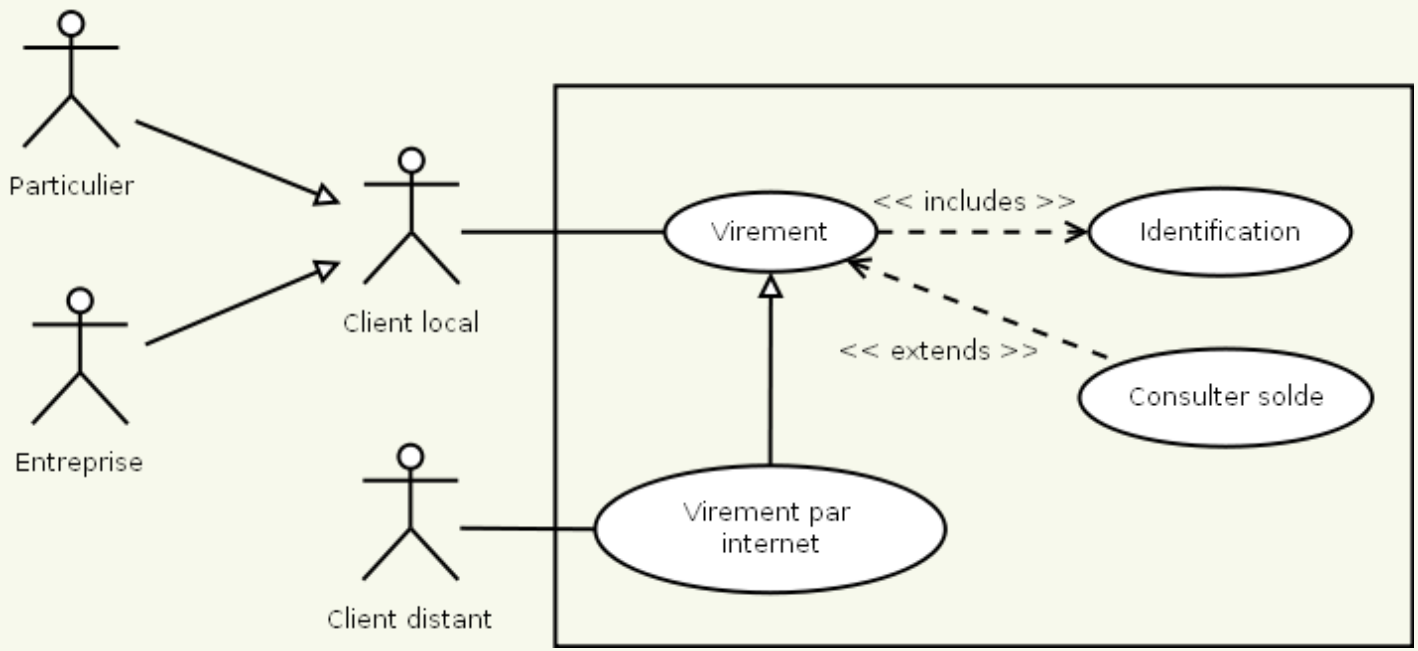
On retrouve la même sémantique que celle de l'héritage entre classes :

- pour les acteurs : héritage des cas d'utilisations déclenchables,
- pour les cas d'utilisation : héritage des inclusions/extensions

Exemple récapitulatif

Système de virements

- Client local : virements (incluant une identification)
- Client distant : virements par internet (spécialisation du virement standard)



Plan du cours

Les diagrammes de séquence

Les diagrammes de séquence permettent de représenter les *interactions* entre objets d'un point de vue chronologique.

Ils se concentrent sur la séquence des interactions d'un point de vue temporel.

Ils font partie des diagrammes dits d'interaction, avec les diagrammes de collaboration.

Les diagrammes de séquence

Les interactions se traduisent par des envois de messages entre les objets.

Les diagrammes de séquences peuvent s'appliquer

- à la documentation des cas d'utilisation, en représentant les interactions entre les acteurs et le système. Les étiquettes des messages sont des événements qui se produisent dans le système.
- à la représentation des interactions informatiques et la répartition des flots de contrôle. Les messages échangés unifient les différentes formes de communication entre les objets (appels de procédure, émission de signaux, etc.).

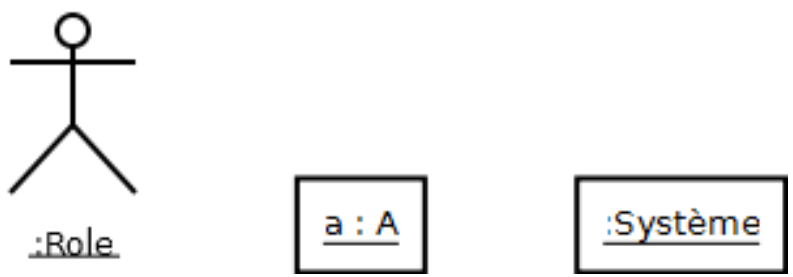
Les objets des diagrammes de séquence

Ce sont des entités appartenant au système (instance d'une classe), ou se trouvant à ses limites (les acteurs), ou le système lui-même.

Ils représentent :

- Soit des concepts abstraits, ou des acteurs (documentation des cas d'utilisations)
- Soit des objets d'implantation (interactions informatiques)

Les objets sont identifiés par l'intermédiaire des diagrammes de cas d'utilisation ou des diagrammes de classes. Ils sont dénotés classiquement.

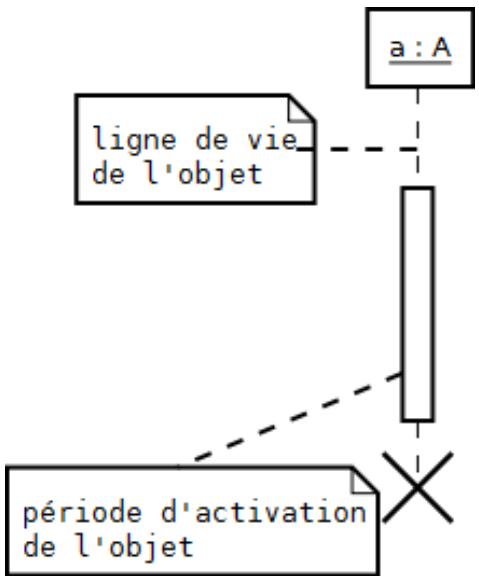


La ligne de vie des objets

Elle représente la période de temps durant laquelle l'objet existe.

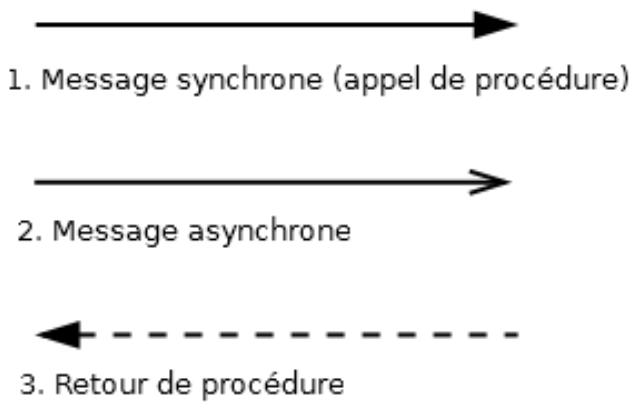
La période d'activation correspond à un intervalle de temps durant lequel l'objet effectue une actions, soit directement, soit par l'intermédiaire d'un autre objet qui lui sert de sous-traitant (appel de procédure).

La vie de l'objet commence par l'objet lui-même et se termine à la fin de la ligne de vie, éventuellement marquée par une croix indiquant la destruction de l'objet.



Les types de messages

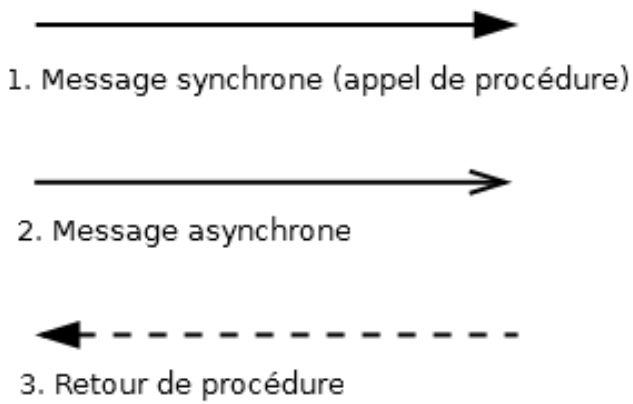
On distingue 3 types de messages.



1. Message synchrone : correspond à un appel de procédure ou flot de contrôle imbriqué. La séquence imbriquée est entièrement faite avant que l'appelant ne continue : l'appelant est bloqué jusqu'à ce que l'appelé termine.

Les types de messages

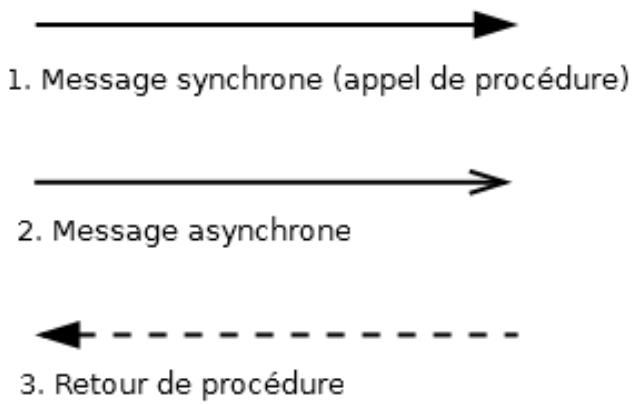
On distingue 3 types de messages.



2. Message asynchrone : correspond à un message "à plat", non imbriqué. L'appelant n'est pas bloqué jusqu'à ce que l'appelé termine.

Les types de messages

On distingue 3 types de messages.



3. Retour de procédure : correspond à la terminaison d'une procédure avec le retour éventuel d'une valeur. Ces messages peuvent ne pas être dessinés, ils sont alors implicites.

Un premier exemple

Exemple d'un système de communication téléphonique

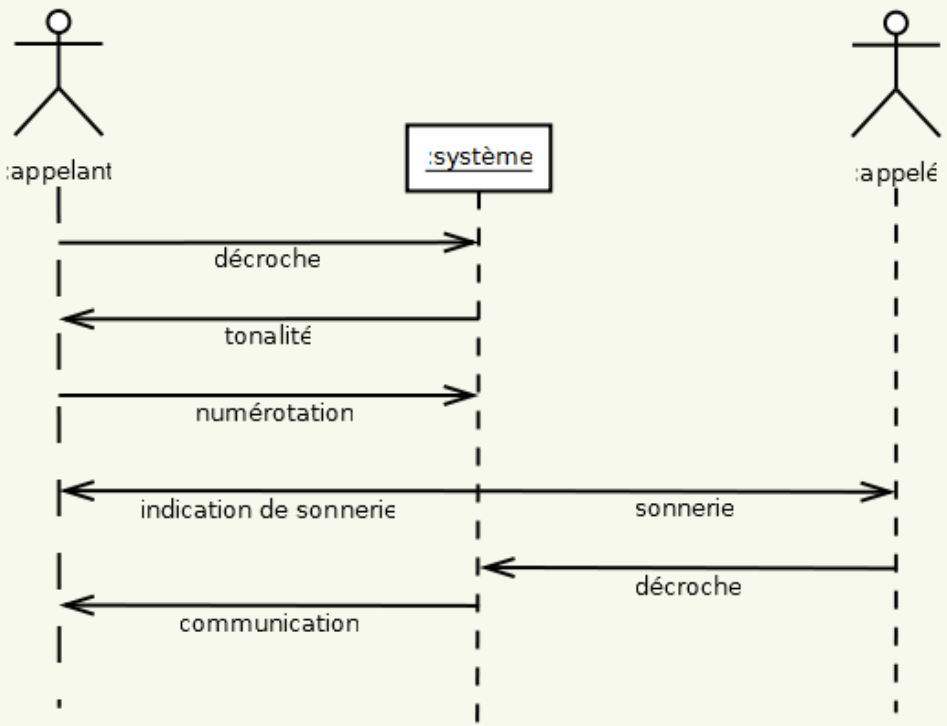
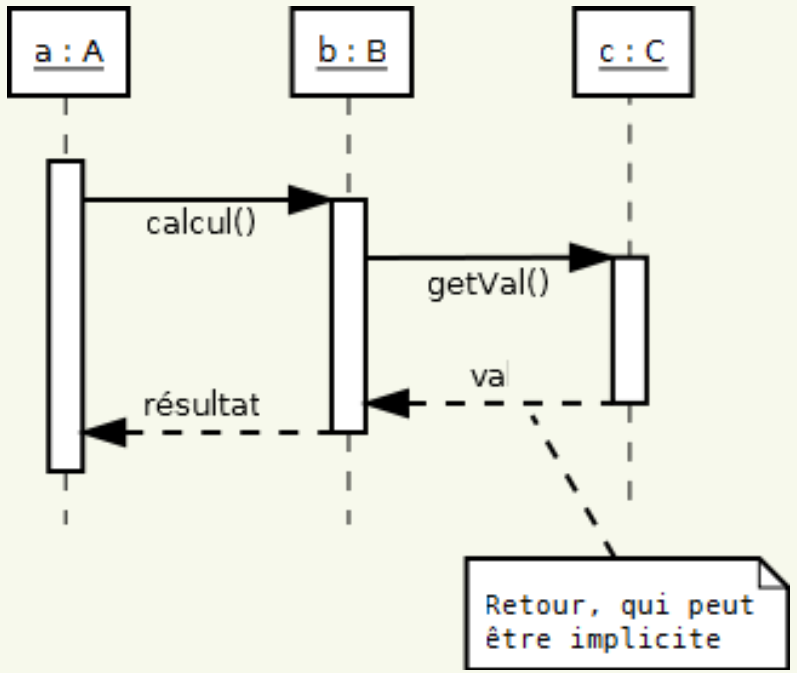


Illustration d'un diagramme de cas d'utilisation, avec messages asynchrones.

Un second exemple

Contrôle de flot d'exécution

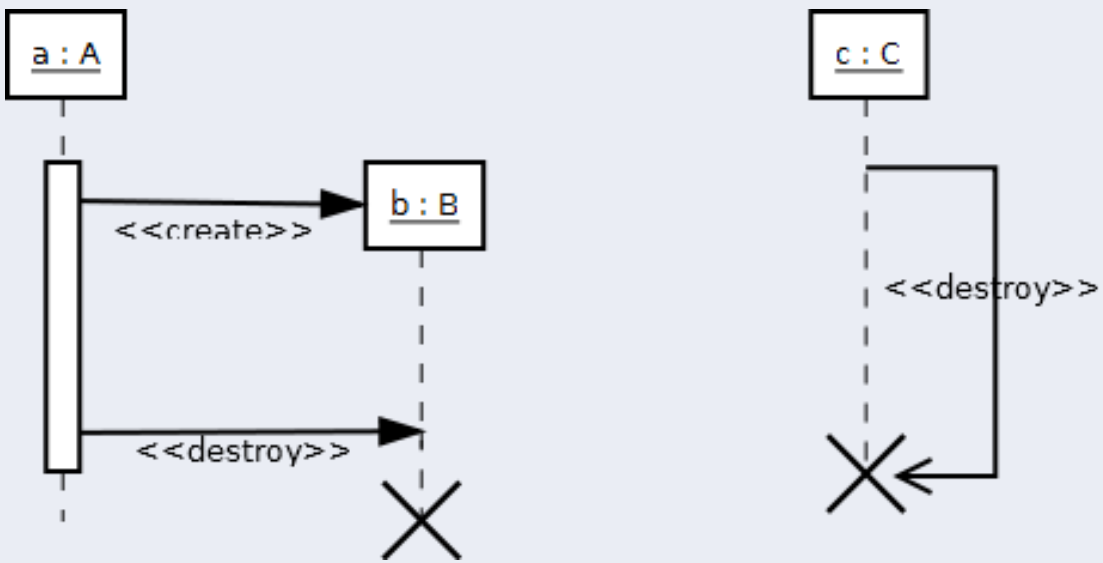


Création et destruction d'un objet

Créer et détruire des objets

Deux types de messages particuliers permettent de créer et de détruire des objets.

- La création se symbolise par un message << *create* >>
- La destruction se symbolise par un message << *destroy* >>



Etiquettes des messages

Syntaxe générale

La syntaxe des étiquettes de message est la suivante :

[Garde] *Iteration* *resultat :=* nom_message*(arguments)*

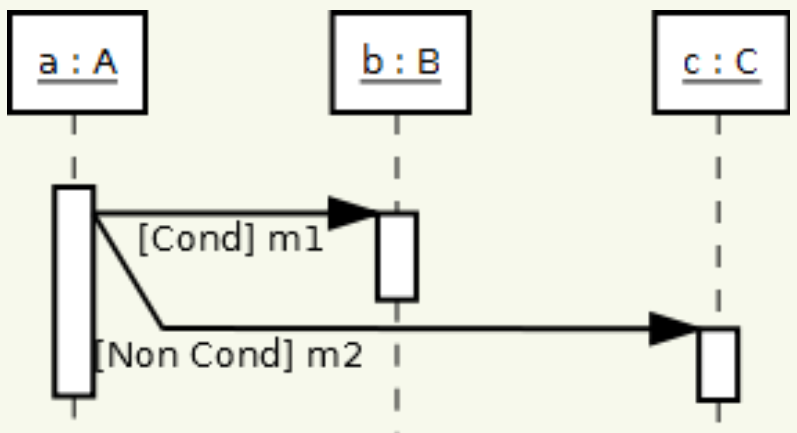
où

- *[Garde]* est la condition d'envoi du message (facultatif)
- *Iteration* décrit une éventuelle itération sur l'envoi du message (facultatif)
- *resultat :=* est l'affectation d'une valeur de retour (facultatif)
- nom_message est le nom du message
- *arguments* sont les arguments du message (facultatif)

Etiquettes : messages gardés

Les gardes permettent de spécifier des conditions pour l'envoi des messages. Les messages ne sont envoyés que si la condition est satisfaite.

Messages conditionnels

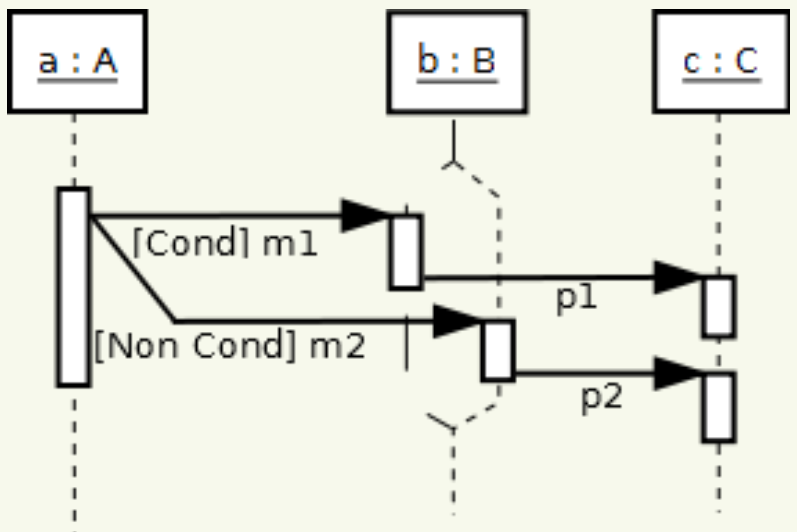


Etiquettes : messages gardés

Les gardes permettent de spécifier des conditions pour l'envoi des messages. Les messages ne sont envoyés que si la condition est satisfaite.

La ligne de vie d'un objet peut être dédoublée pour indiquer des actions qui sont effectuées suite à un message conditionnel (UML 1.5).

Dédoublement de la ligne de vie

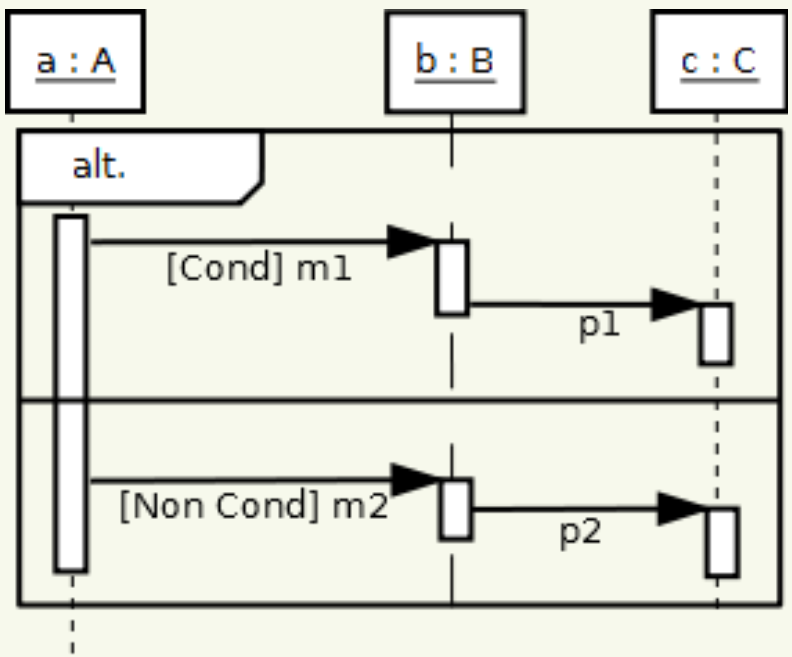


Etiquettes : messages gardés

Les gardes permettent de spécifier des conditions pour l'envoi des messages. Les messages ne sont envoyés que si la condition est satisfaite.

UML 2.0 ajoute la possibilité d'utiliser des fragments, plus lisibles.

Utilisation d'un fragment d'alternative



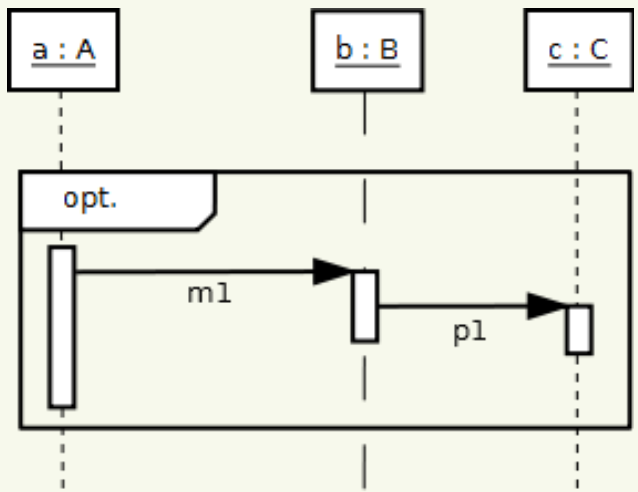
Etiquettes : messages gardés

Les gardes permettent de spécifier des conditions pour l'envoi des messages. Les messages ne sont envoyés que si la condition est satisfaite.

UML 2.0 ajoute la possibilité d'utiliser des fragments, plus lisibles.

Notamment pour les actions optionnelles.

Utilisation d'un fragment optionnel



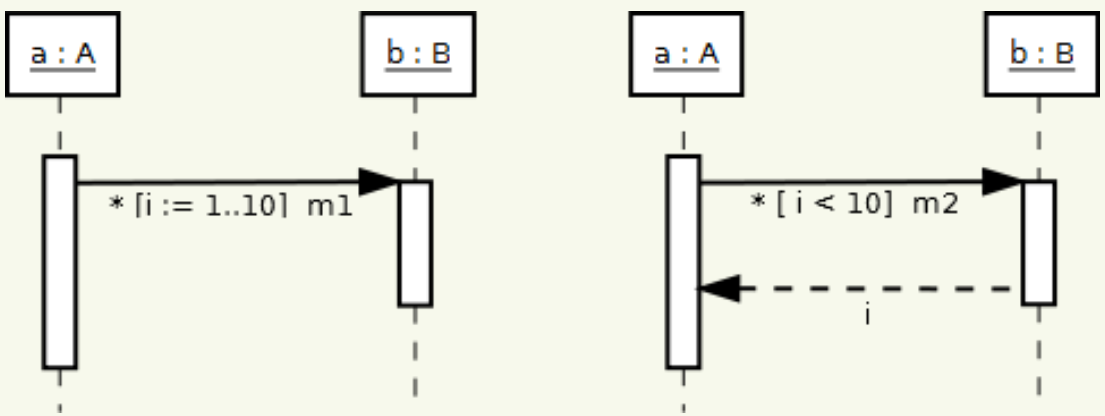
Etiquettes : itérations

Deux types d'itérations

On distingue deux types d'itérations :

- Itération *séquentielle* : envoi séquentiel de plusieurs instances du même message, dénoté par $*$[*clause d'itération*]
- Itération *parallèle* : envoi en parallèle de plusieurs instances du même message, dénoté par $* \parallel$ [*clause d'itération*]

Envoi séquentiel



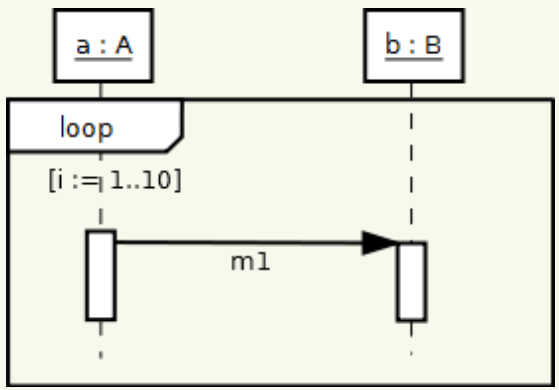
Etiquettes : itérations

Deux types d'itérations

On distingue deux types d'itérations :

- Itération *séquentielle* : envoi séquentiel de plusieurs instances du même message, dénoté par $*$[*clause d'itération*]
- Itération *parallèle* : envoi en parallèle de plusieurs instances du même message, dénoté par $* \parallel$ [*clause d'itération*]

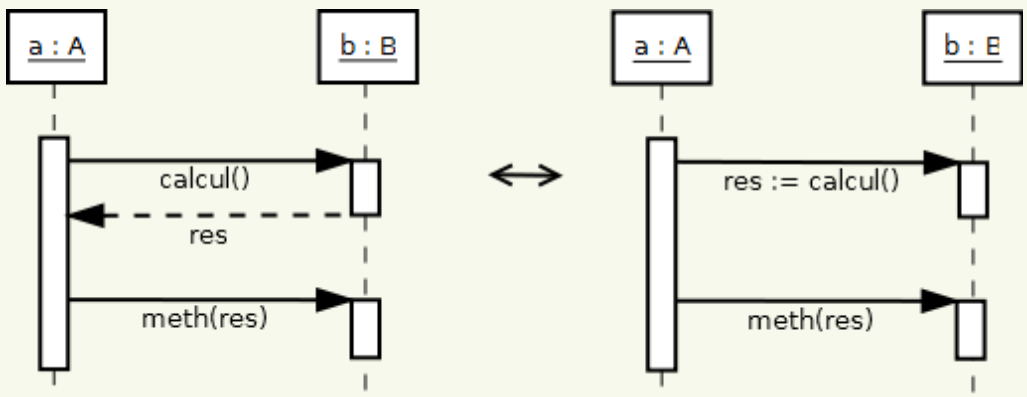
Envoi séquentiel, avec un fragment



Etiquettes : valeur de retour

Le résultat est constitué d'une liste de valeurs retournées par le message.
Ces valeurs peuvent être utilisées comme paramètres d'autres messages.

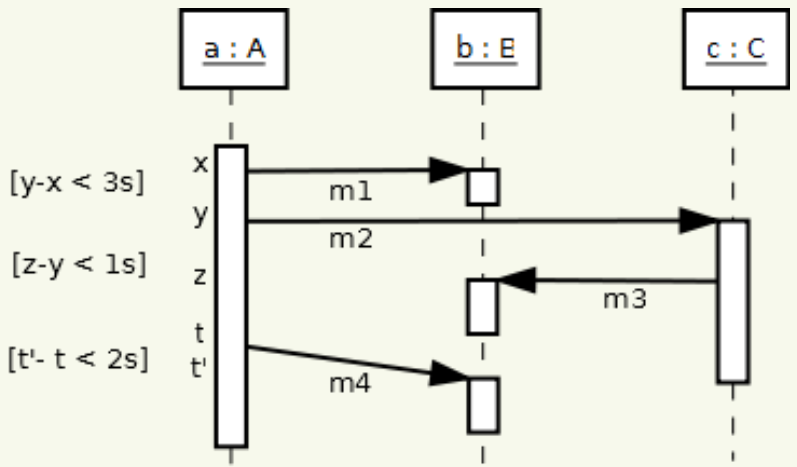
Deux notations possibles



Contraintes temporelles

On peut nommer l'instant d'émission d'un message, ainsi que l'instant de sa réception. Cela permet de poser des contraintes de temps sur l'envoi et sur la réception de messages.
Par convention, si l'instant d'émission est dénoté par x, l'instant de réception de ce même message est dénoté par x'.

Utilisation de contraintes temporelles



Objets actifs

Définition

Un objet actif est un objet qui a son propre flot d'exécution.

Quelques exemples d'objets actifs

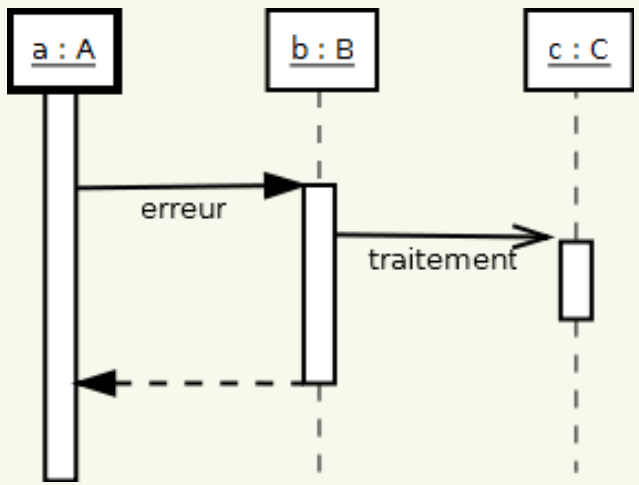
- les processus et les threads
- les objets sources d'événements, comme les boutons dans les interfaces graphiques
- les minuteries, comme les timers en Java

Ces objets permettent d'exécuter une opération soit une seule fois, après un certain intervalle de temps, soit de façon répétée à intervalles de temps données.

Objets actifs (suite)

Un objet actif peut, le temps d'une opération, activer un objet passif qui peut alors activer d'autres objets passifs. Lorsque l'opération est terminée l'objet passif redonne le contrôle à l'objet qui l'a activé.

Illustration



Un objet actif est représenté par des contours plus épais.

Dans un environnement multi-tâches, plusieurs objets peuvent être actifs simultanément.

Plan du cours

Diagrammes de collaboration

Les diagrammes de collaboration (également appelés diagrammes de communication en UML 2.0) permettent de représenter les *interactions* entre des objets (et éventuellement des acteurs).

Par opposition aux diagrammes de séquences, les contextes sont explicitements représentés, en précisant les états des objets qui interagissent. Pour mettre en évidence la dimension temporelle, les messages envoyés par les différents objets peuvent être numérotés.

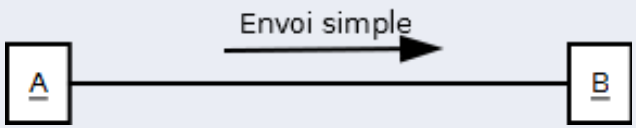
Ils font partie des diagrammes dits d'interaction, avec les diagrammes de sequence. Le passage d'un diagramme de collaboration à un diagramme de séquence (et inversement) est simple.

Les messages

Les objets communiquent en échangeant des messages représentés sous forme de flèche. Les messages sont étiquetés par le nom de l'opération ou du signal invoqué. L'envoi d'un message nécessite que le récepteur puisse réaliser l'opération.

Trois types de messages

On distingue 3 types de messages :



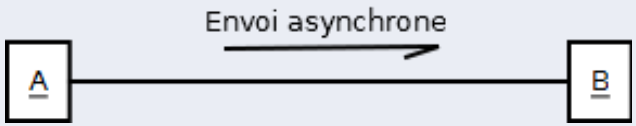
L'**appel de procédure** ou flot de contrôle emboîté : la séquence emboitée doit se terminer pour que la séquence englobante reprenne le contrôle.

Les messages

Les objets communiquent en échangeant des messages représentés sous forme de flèche. Les messages sont étiquetés par le nom de l'opération ou du signal invoqué. L'envoi d'un message nécessite que le récepteur puisse réaliser l'opération.

Trois types de messages

On distingue 3 types de messages :



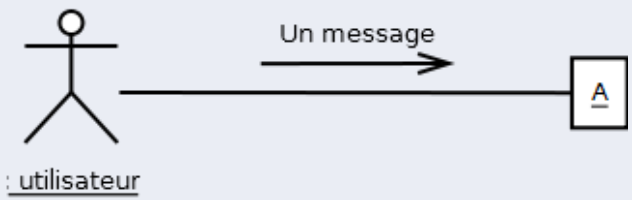
Le **flot de contrôle** asynchrone : pas de messages englobant, il n'y a donc pas d'obligation de terminaison d'autres messages pour poursuivre la séquence.

Les messages

Les objets communiquent en échangeant des messages représentés sous forme de flèche. Les messages sont étiquetés par le nom de l'opération ou du signal invoqué. L'envoi d'un message nécessite que le récepteur puisse réaliser l'opération.

Trois types de messages

On distingue 3 types de messages :



Le **flot de contrôle à plat** : cas particulier des messages asynchrones, il modélise une progression non procédurale souvent utilisée pour les messages entre un acteur et le système.

Etiquettes des messages

Syntaxe des messages

La syntaxe des étiquettes de message est la suivante :

Synchronisation *[Garde]* *Iteration* *resultat :=* nom_message*(arguments)*

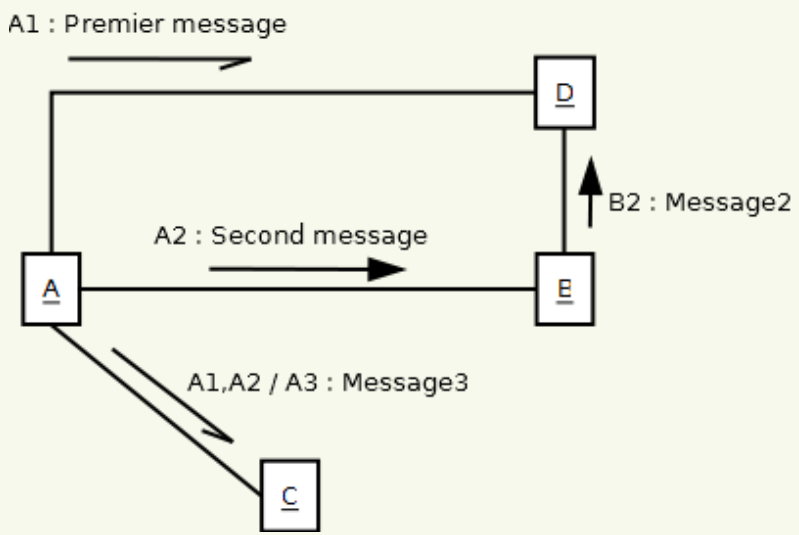
où

- *Synchronisation* permet de synchroniser le message avec d'autres (facultatif)
- *[Garde]* est la condition d'envoi du message (facultatif)
- *Iteration* décrit une éventuelle itération sur l'envoi du message (facultatif)
- *resultat :=* est l'affectation d'une valeur de retour (facultatif)
- nom_message est le nom du message
- *arguments* sont les arguments du message (facultatif)

Synchronisation des messages

La synchronisation de messages permet d'envoyer un message si et seulement si d'autres messages ont déjà été envoyés.
Ceci se dénote par : $M1, M2, .../M$ où $M1, M2$ sont les messages dont dépend M.

Synchronisation

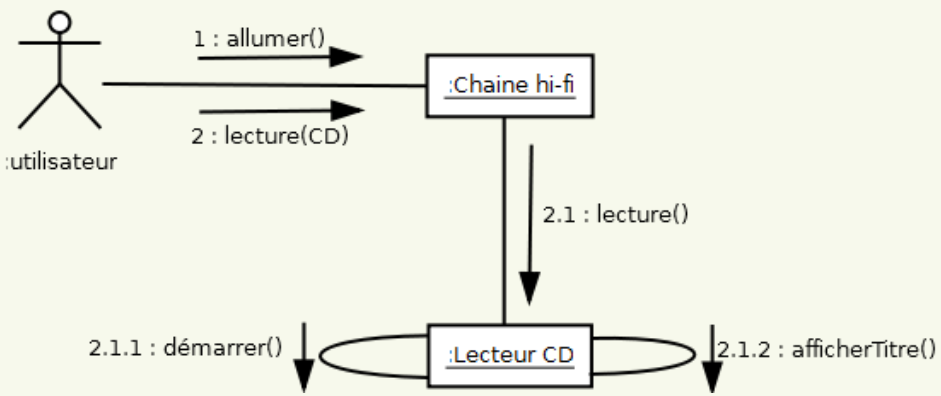


Séquencement

On préfixe le message par une numérotation de manière à indiquer l'ordre d'envoi des messages.

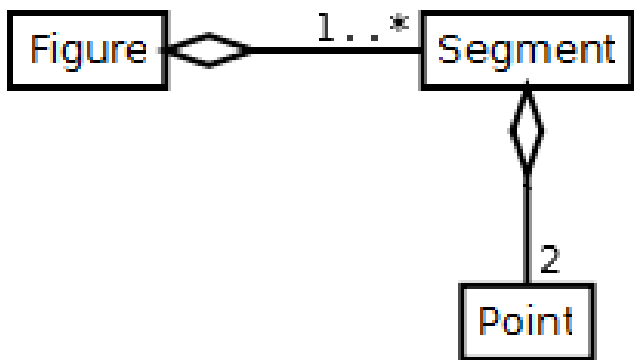
La numérotation est englobante dans le cas des appels de procédure : N (appel initial), $N.1$ (premier appel imbriqué), $N.2$ (deuxième appel imbriqué, $N+1$ (au même niveau que N).

Séquence de messages



Exercice

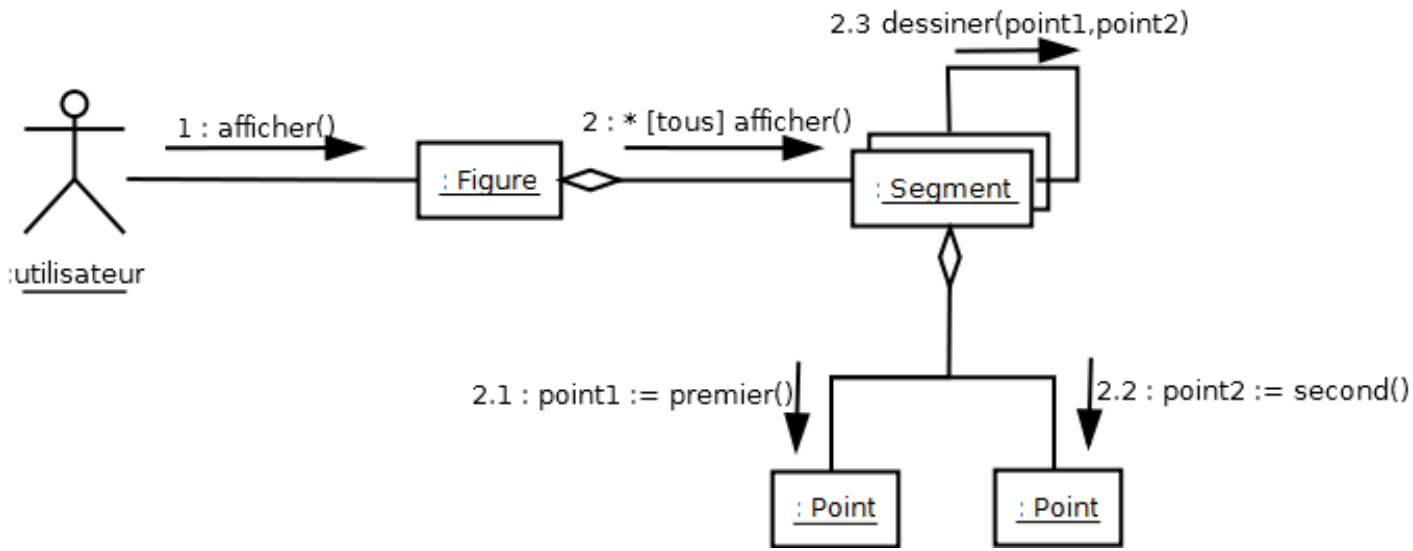
Soit le diagramme de classes suivant :



Donner les diagrammes de collaboration représentant l'affichage d'une figure demandé par un utilisateur.

Exercice - Solution

Diagramme de collaboration représentant l'affichage d'une figure demandé par un utilisateur.



Plan du cours

Les diagrammes d'états-transitions

Les diagrammes d'états-transitions permettent principalement de décrire le comportement des objets d'une classe.

Ils peuvent également décrire les aspects dynamique d'un cas d'utilisation, d'un acteur, d'un système, ou d'un sous-système.

Les diagrammes d'états-transitions UML sont inspirés des *Statecharts* de David Harel. Il s'agit d'automates hiérarchiques, qui peuvent être mis en parallèle.

Les états

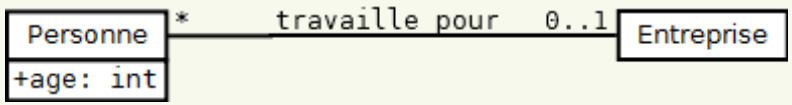
Etat d'un objet

L'état d'un objet correspond à l'ensemble des valeurs de ses attributs et à l'ensemble des liens qu'il entretient avec d'autres objets.

Un état est une condition ou une situation, dans la vie d'un objet, qui dure un certain temps pendant lequel cet objet satisfait une condition, effectue une activité, ou attend un événement.

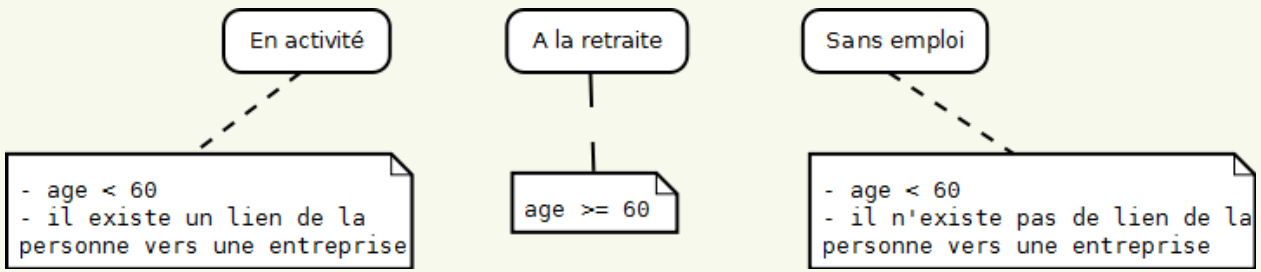
Les états : un premier exemple

Personnes employées dans une entreprise



Une personne peut avoir un des trois états suivants :

en activité, à la retraite, sans emploi



Les états : état initial et état final

Etat initial

Un *état initial* est un pseudo-état qui permet de montrer l'état dans lequel un objet se trouve au moment de sa création.

Etat initial

Etat final

Un *état final* est un pseudo-état qui permet de montrer la fin du comportement d'un objet, en particulier au moment de sa destruction.

Etat final

Transitions

Les transitions permettent à un objet de changer d'état, en fonction des événements qu'il reçoit.

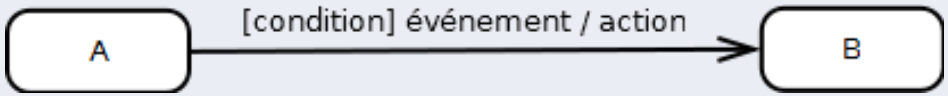
Contenu d'une transition

Une transition comporte :

- un état source
- un état destination
- un événement
- une condition, appelée garde
- une action

Transitions (suite)

Notation d'une transition



Sémantique

Si :

l'objet est dans l'état A **et** l'événement se produit **et** la condition est vraie

alors l'objet effectue l'action et passe dans l'état B.

Si aucune transition partant de l'état courant n'est étiquetée par l'événement reçu, alors rien ne se passe ; en particulier, l'objet ne change pas d'état.

Remarque : Le passage d'un état à un autre est considéré comme instantané.

Evénements

On distingue 4 sortes d'événements en UML :

- Un *événement d'appel* ("call event") est un événement causé par l'appel d'une opération. Dans ce cas, l'événement est de la forme $op(x_1, x_2, ..., x_N)$ où *op* est une opération de la classe.

Evénements

On distingue 4 sortes d'événements en UML :

- Un *événement d'appel* ("call event") est un événement causé par l'appel d'une opération. Dans ce cas, l'événement est de la forme $op(x_1, x_2, ..., x_N)$ où *op* est une opération de la classe.
- Un *événement modification* ("change event") est un événement causé par le passage d'une condition de la valeur faux à la valeur vrai, suite à un changement de valeur d'un attribut ou d'un lien.

Evénements

On distingue 4 sortes d'événements en UML :

- Un *événement d'appel* ("call event") est un événement causé par l'appel d'une opération. Dans ce cas, l'événement est de la forme $op(x_1, x_2, ..., x_N)$ où *op* est une opération de la classe.
- Un *événement modification* ("change event") est un événement causé par le passage d'une condition de la valeur faux à la valeur vrai, suite à un changement de valeur d'un attribut ou d'un lien.
- Un *événement temporel* ("time event") est un événement qui survient quand une temporisation arrive à expiration. Une temporisation peut être relative (délai), ou absolue (spécification de l'heure à laquelle la transition doit être effectuée).

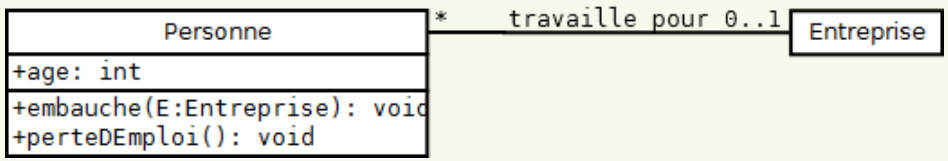
Evénements

On distingue 4 sortes d'événements en UML :

- Un *événement d'appel* ("call event") est un événement causé par l'appel d'une opération. Dans ce cas, l'événement est de la forme $op(x_1, x_2, ..., x_N)$ où *op* est une opération de la classe.
- Un *événement modification* ("change event") est un événement causé par le passage d'une condition de la valeur faux à la valeur vrai, suite à un changement de valeur d'un attribut ou d'un lien.
- Un *événement temporel* ("time event") est un événement qui survient quand une temporisation arrive à expiration. Une temporisation peut être relative (délai), ou absolue (spécification de l'heure à laquelle la transition doit être effectuée).
- Un *événement signal* ("signal event") est un stimulus asynchrone entre deux objets. Par exemple, un clic de souris est un signal.

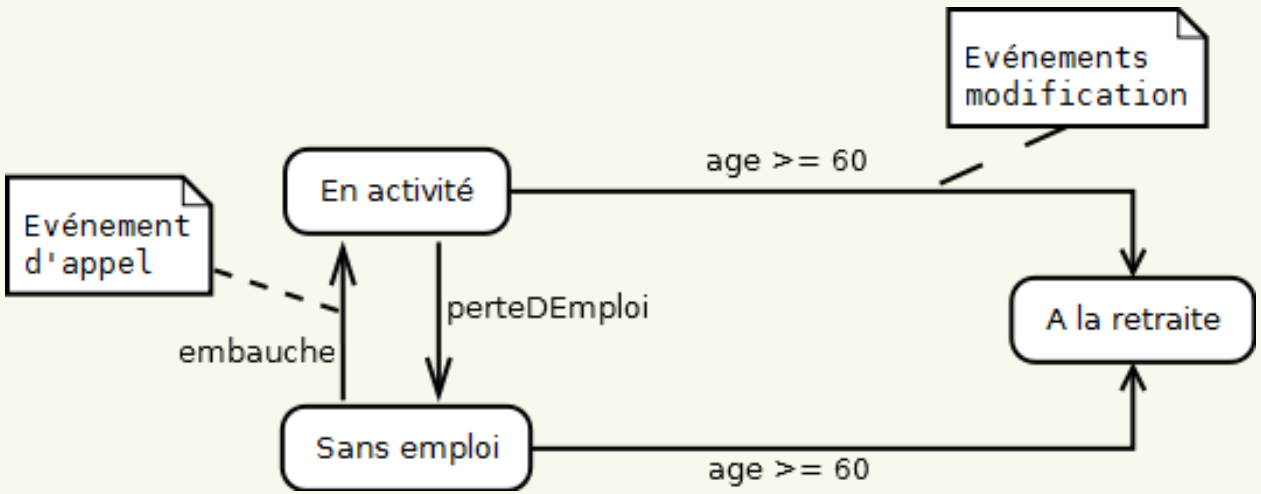
Exemple 1 : activité d'une personne

Diagramme de classe



Exemple 1 : activité d'une personne

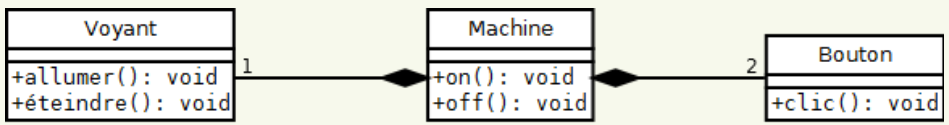
Diagramme d'états-transitions associé à la classe "Personne"



Exemple 2 : deux boutons et 1 voyant

On considère une machine avec deux boutons permettant de mettre une machine sous tension (signal "on") ou de l'éteindre (signal "off"). Le voyant indique si la machine est sous tension ou hors tension. Après une minute sans utilisation, la machine se met automatiquement hors tension.

Diagramme de classe



Exemple 2 : deux boutons et 1 voyant

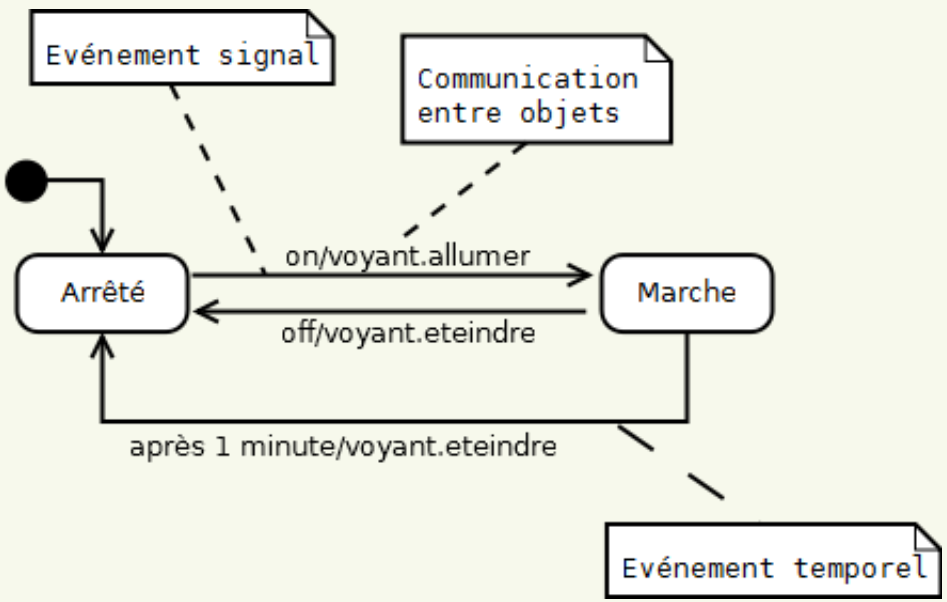
On considère une machine avec deux boutons permettant de mettre une machine sous tension (signal "on") ou de l'éteindre (signal "off"). Le voyant indique si la machine est sous tension ou hors tension. Après une minute sans utilisation, la machine se met automatiquement hors tension.

Diagramme d'états-transitions associé à la classe "Voyant"

Exemple 2 : deux boutons et 1 voyant

On considère une machine avec deux boutons permettant de mettre une machine sous tension (signal "on") ou de l'éteindre (signal "off"). Le voyant indique si la machine est sous tension ou hors tension. Après une minute sans utilisation, la machine se met automatiquement hors tension.

Diagramme d'états-transitions associé à la classe "Machine"



Remarque

Les automates considérés en UML sont *a priori* non-déterministes.

On peut donc avoir deux transitions étiquetées par le même événement qui partent du même état.

Néanmoins, d'un point de vue méthodologique, il est souvent préférable d'utiliser des **automates déterministes** pour plus de clarté.

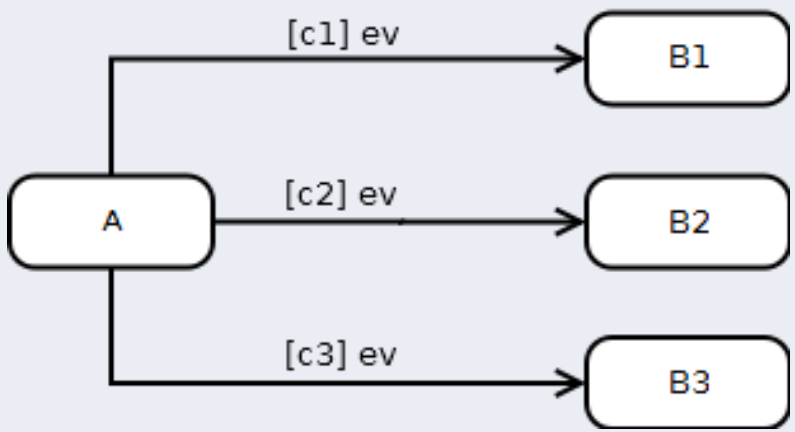
Gardes

Une garde est une condition booléenne, notée entre crochets [ ].

La garde de la transition est évaluée quand un événement se produit.

Gardes et déterminisme

Supposons que plusieurs transitions partant du même état A soient déclenchées par le même événement.



Pour que l'automate soit déterministe, il faut que les gardes $c1$, $c2$ et $c3$ soient mutuellement exclusives.

Gardes (suite)

Sémantique

Si aucune condition n'est vérifiée, alors rien ne se passe et l'objet ne change pas d'état.

Remarque : Il ne faut pas confondre un événement de déclenchement et une garde :

- Un événement de changement est un événement qui déclenche une transition lorsque la condition passe à vrai ;
- Une garde est une condition booléenne évaluée lorsque l'événement se produit.

Action et activité

Action

Une action consiste en la génération d'un signal ou l'invocation d'une opération. Une action est considérée comme *instantanée* (ie. le temps d'exécution est négligeable) et *atomique* (ie. non interruptible).

Activité

Une activité correspond à une opération qui prend un temps non négligeable et peut être interrompue.

Actions associées aux états

Les actions sont généralement associées aux transitions, mais on peut également les associer aux états. On peut en particulier :

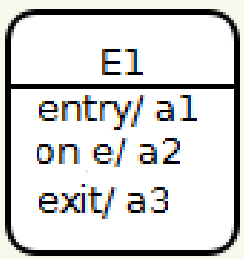
- spécifier une action à effectuer lorsqu'on entre dans un état :
  entry/ action
- spécifier une action à effectuer si un événement survient :
  on événement / action (l'événement est "interne")
- spécifier une action à effectuer lorsqu'on sort d'un état :
  exit / action
- spécifier une activité à effectuer lorsqu'on est dans l'état :
  do/ activité

Actions associées aux états (suite)

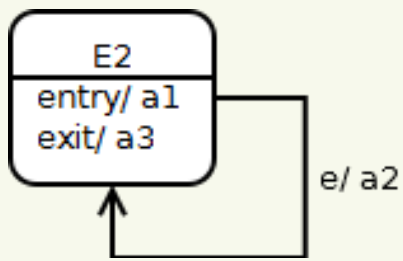
Remarque : le déclenchement d'un événement interne n'entraîne pas l'exécution des actions d'entrée et de sortie.

Illustration

On suppose qu'on entre dans l'un des deux états E1 ou E2 et qu'on reçoit une succession d'événements e.



a1 a2* a3



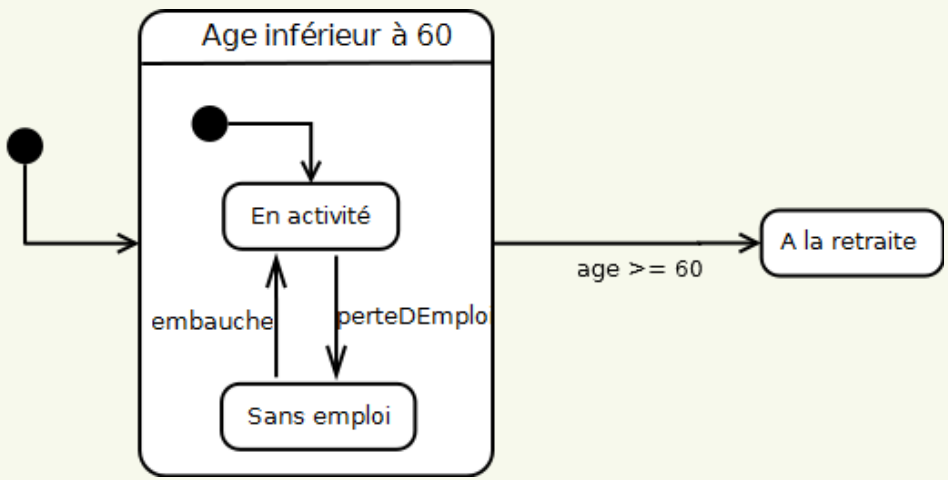
a1 (a3 a2 a1)* a3

Etats composites

Définition

Un état composite est un état qui se décompose en plusieurs sous états. Les états composites permettent de structurer les automates pour les rendre plus lisibles.
Ils permettent en particulier de factoriser des transitions similaires qui partent de plusieurs états.

Activités d'une personne



Etats composites

Définition

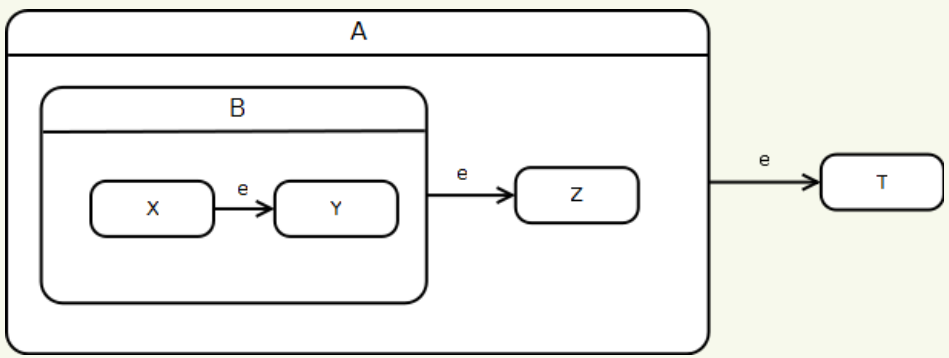
Un état composite est un état qui se décompose en plusieurs sous états. Les états composites permettent de structurer les automates pour les rendre plus lisibles.
Ils permettent en particulier de factoriser des transitions similaires qui partent de plusieurs états.

Remarque : Il peut exister des transitions entre différents niveaux de l'automate, mais il est préférable de limiter leur nombre.

Etats composites (suite)

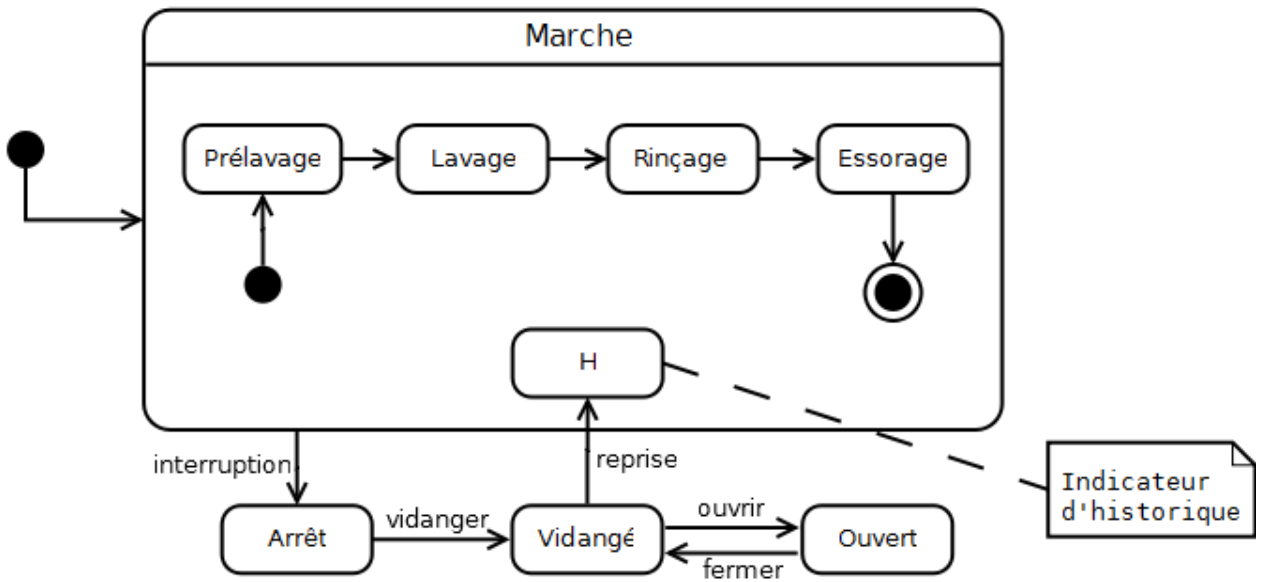
Lorsque deux transitions peuvent être effectuées, l'une sur un sous-état, et l'autre sur l'état englobant, c'est la transition sur le sous-état (la plus "spécialisée") qui est effectuée.

Priorité des transitions



Indicateurs d'historique

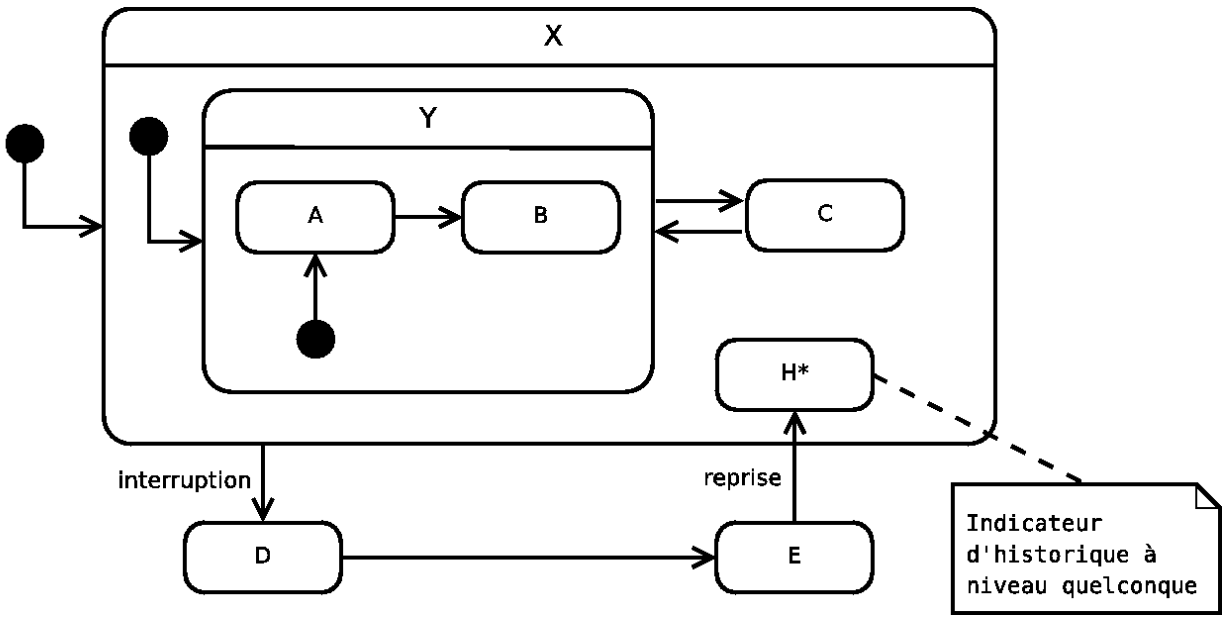
L'indicateur d'historique noté H est un pseudo état qui permet de mémoriser le dernier état visité d'un automate pour y retourner ultérieurement.



Si une interruption survient, suite à la reprise, l'état du système retournera à celui avant l'interruption.

Indicateurs d'historique

L'indicateur d'historique à niveau quelconque noté H* est un pseudo état qui permet de mémoriser le dernier état visité, à un état d'imbrication quelconque, pour y retourner ultérieurement.



Si une interruption survient, suite à la reprise, l'état du système retournera à celui avant l'interruption, c'est-à-dire dans l'état A, B ou C.

Automates en parallèle

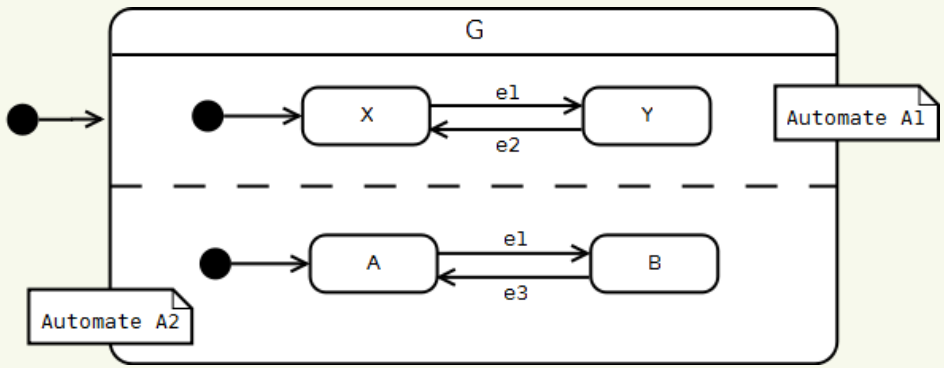
A l'intérieur d'un état, plusieurs automates peuvent s'exécuter en parallèle. Chaque sous-automate a un état initial et un certain nombre d'états finaux.

L'activité d'un tel état se termine lorsque tous les sous-automates arrivent à un état final.

Lorsqu'un événement se produit, toutes les transitions qui peuvent être exécutées sont effectuées.

Automates en parallèle (suite)

Exécution d'automates en parallèles



Il est possible d'aplatir l'automate pour obtenir un comportement équivalent.

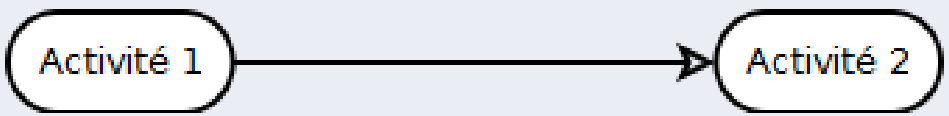
Typiquement, les automates en parallèle s'utilisent lorsqu'un objet est formé de la composition ou l'agrégation d'autres objets, en particulier si le comportement de l'objet composite correspond à la mise en parallèle des comportements des composants.

Plan du cours

Diagrammes d'activité

Les diagrammes d'activité permettent de modéliser le comportement d'une méthode ou le déroulement d'un cas d'utilisation, en reprenant un enchaînement d'activités.

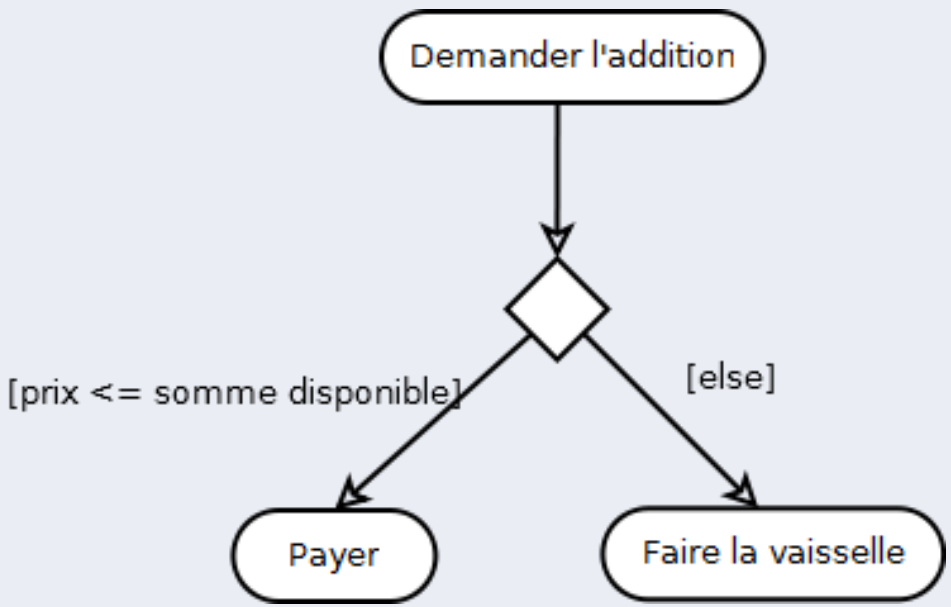
Activité et transition



Transitions conditionnelles

Il est possible de donner des conditions pour le franchissement de transitions.

Transitions conditionnelles

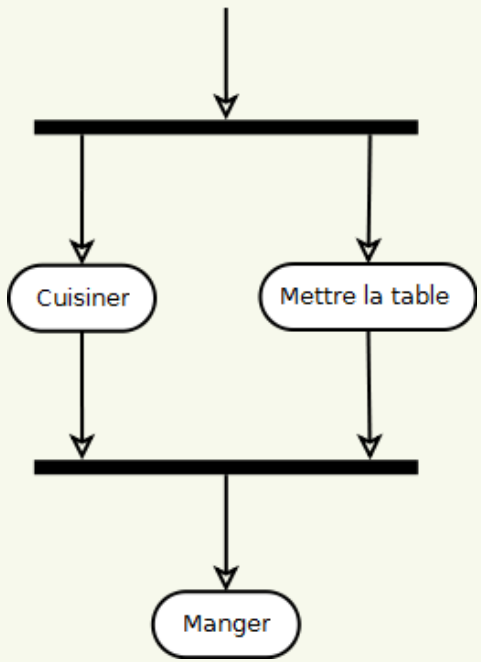


Synchronisation

Il est possible de synchroniser des transitions à l'aide d'une **barre de synchronisation**. Celle-ci permet d'ouvrir et de fermer des branches parallèles au sein d'un flôt d'exécution :

- Les transitions qui partent d'une barre de synchronisation ont lieu en même temps.
- On ne franchit une barre de synchronisation qu'après réalisation de toutes les transitions qui s'y rattachent.

Barre de synchronisation



Couloirs d'activités

Afin d'organiser un diagramme d'activités selon les différents responsables des actions représentées, il est possible de définir des "couloirs d'activités".

Couloirs d'activités