Chapitre 4 : le diagramme états-transitions

Synthex UML 2, PEARSON © Version ébauche Mai 2004. Yann Thierry-Mieg

Les diagrammes états-transitions ou Statecharts d'UML permettent de décrire le comportement interne d'un objet à l'aide d'un automate à états finis. Il décrit les séquences possibles d'états et d'actions qu'une instance de classe peut traiter au cours de son cycle de vie en réaction à des événements discrets (type signaux, invocations de méthode). Les statecharts sont habituellement utilisés pour décrire le comportement d'une instance de classeurs (classes et composants), mais ils peuvent aussi être utilisés pour décrire le comportement interne d'autres éléments tels que les cas d'utilisation, les sous systèmes, les méthodes…

Ces diagrammes sont bien adaptés à la description d'objets ayant un comportement d'automate, et permettent de cerner l'ensemble des comportements d'un élément du système. Cependant, la vision globale du système est moins apparente sur ces diagrammes, car on ne s'intéresse ici qu'a un seul élément du système en isolation par rapport à son environnement. Les diagrammes d'interaction (Chapitre 3) seront utilisés pour lier les parties du système entre elles.

1. Modélisation à l'aide d'automates

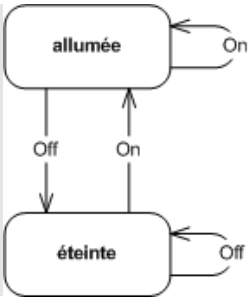
1.1. Notion d'état

Un diagramme état-transition est un graphe qui représente un automate à états finis, c'est-à-dire une machine dont le comportement des sorties ne dépend pas seulement de l'état de ses entrées, mais aussi d'un historique des sollicitations passées : cet historique est caractérisé par un état.

Ainsi, l'effet d'une action sur l'objet va dépendre de son état interne, qui peut varier au cours du temps. Par exemple, considérons une lampe munie de deux boutons poussoirs, un qui allume « On » et l'autre qui éteint la lampe « Off ». L'appui sur « On » ne produit pas d'effet si la lampe est déjà allumée ; la réaction d'une instance de lampe à cet événement va dépendre de son état interne.

Exemple 1 : La Lampe

Cet exemple simple exhibe la notion d'état, et l'effet des invocations de méthodes en fonction de l'état courant.

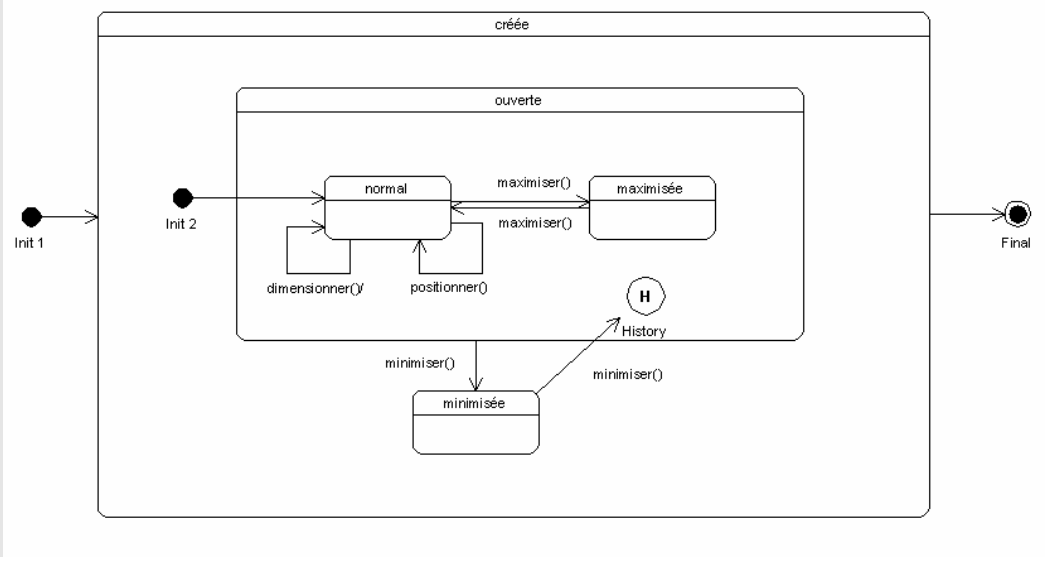


Les états sont représentés par des rectangles aux coins arrondis, tandis que les transitions sont représentées par des arcs orientés liant les états entre eux. Certains états, dits composites, peuvent contenir des sous diagrammes. UML offre également un certain nombre de constructions à la sémantique particulière, tels que l'historique qui permet de retrouver un état précédent, les points de choix pour exprimer des alternatives, ou encore un support pour l'expression de mécanismes concurrents.

Exemple 1 Fenêtre d'application

Ce diagramme présente le comportement simplifié d'une fenêtre d'application, telle qu'elle répond aux stimuli des trois boutons placés dans l'angle. Une fenêtre peut être dans trois états principaux : minimisée, normal, maximisée. A l'état minimisée elle est représentée par une icône dans la barre des tâches, à l'état normal elle peut être déplacée et redimensionnée, à l'état maximisée elle occupe toute la surface disponible de l'écran et ne peut être déplacée ou redimensionnée.

Les arcs sont étiquetés par le nom de l'événement qui déclenche la transition associée. L'état initial (*Init 1* et *2* ici) noté par un cercle plein, désigne le point d'entrée du diagramme ou du sous-diagramme. Une nouvelle instance de fenêtre sera initialisée à l'état créée, dans le sous état Ouverte, dans le sous état normal. Le pseudo état historique, désigné par un H dans un cercle, permet de retrouver quand on dé-iconifie la fenêtre son état précédent : normale ou maximisée. L'état final est désigné par un point dans un cercle (*Final* ici) correspond à la fin de vie de l'instance et à sa destruction.



1.2. Evénements

La vue proposée par les diagrammes états-transition est orientée sur les réactions d'une partie du système à des événements discrets. Un état représente une période dans la vie d'un objet dans laquelle l'objet attend un événement, ou accomplit une activité. Quand un événement est reçu, une transition peut être déclenchée qui va faire basculer l'objet dans un nouvel état.

Les transitions d'un diagramme état transition sont donc déclenchées par des *événements*. La notion d'événement est assez large en UML, et peut désigner

- Un appel de méthode sur l'objet courant. Ceci génère un évènement de type *call*.
- Le passage de faux à vrai de la valeur de vérité d'une condition booléenne. Ceci génère implicitement un événement de type *change.*
- La réception d'un signal asynchrone, explicitement émis par un autre objet. Ceci génère un événement de type *signal.*
- Le passage d'une durée déterminée après un événement donné. Par défaut on prendra le temps à partir de l'entrée dans l'état courant. Ceci génère un événement de type *after*.
- La terminaison d'une activité interne à un état de type *do/* génère implicitement un événement à sa terminaison, dit *completion event*. Ceci peut déclencher le tir de transitions dites automatiques, qui ne portent pas de déclencheur explicite, mais quittent l'état courant.

Notation et spécification

Un événement de type *call* ou *signal* est déclaré sous la forme : *nom-événement **'('**liste- paramètres **')'*** où chaque paramètre est donné sous la forme : *nom-paramètre **':'** type-paramètre*

Les événements de type *call* sont donc des méthodes déclarées au niveau du diagramme de classes. Les signaux sont déclarés par la définition d'une classe portant le stéréotype <<*signal*>>, ne portant pas d'opérations, et dont les attributs sont interprétés comme des arguments.

Un événement de type *change* est introduit sous la forme : *when **'('** condition-booléenne **')'***. Ceci est vu comme un test continu possiblement déclenché à chaque changement des valeurs des variables intervenant dans la condition.

Un événement de type *after* est spécifié par *after **'('** paramètre **')'*** où le paramètre s'évalue comme une durée prise par défaut depuis l'entrée dans l'état courant. Par exemple : *after(10 secondes)* ou *after(10 secondes après l'entrée dans l'état A)*. On peut aussi spécifier des déclenchements d'événements temporels par une clause *when( )*, par exemple *when(date=1 janvier 2000)*.

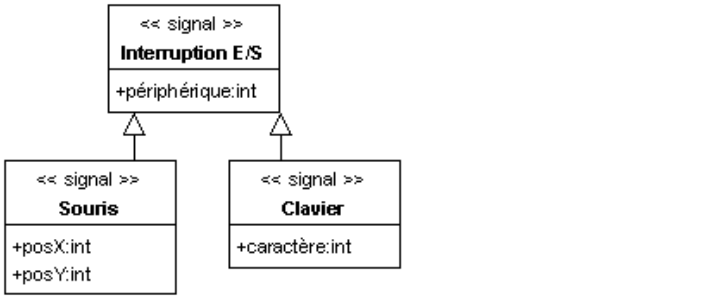
Les déclarations des événements ont une portée de niveau paquetage, et peuvent être utilisés dans tout diagramme d'état des classes du paquetage. Elles ne sont en aucun cas locales à une classe.

Le cas <<signal>>

Un signal est un message émis de façon asynchrone, c'est-à-dire que l'appelant peut poursuivre son execution sans attendre la bonne réception du signal. Ce type de communication n'a un sens que lorsqu'il y a plusieurs processus ou objets actifs en concurrence, en particulier lorsque l'instance cible est munie d'un flot de contrôle indépendant de celui de l'appelant. L'utilisation de signaux à la place de méthodes peut accroitre les possibilités de concurrence, et permet de modéliser certains types de communications matérielles (interruptions matérielles, entrées/sorties) ou en mode déconnecté (le protocole UDP sur IP par exemple). L'émission et la réception d'un signal constituent donc deux évenements distincts.

Exemple : Declaration de signaux

Nous déclarons ici trois signaux. Les signaux sont déclarés comme des classes du paquetage, et peuvent être liés entre eux par des relations d'héritage. Transitivement, si le signal A dérive du signal B alors il déclenche toutes les activités déclenchées par A en plus des siennes propres. Les attributs de classe sont interprétés comme les arguments de l'événement de type *signal*.



1.3. Transitions simples

Une transition entre deux états simples E1 et E2 est représentée par un arc liant ces deux états. Elle indique qu'une instance dans l'état E1 pourra entrer dans l'état E2 et exécuter certaines activités, si un événement déclencheur se produit, et que les conditions de garde sont vérifiées. On parle de *tir* d'une transition quand elle est déclenchée.

Notation

Une transition d'un diagramme état transition est représentée par un arc plein étiqueté par une expression de la forme :

nom-événement ***'('*** *liste-param-évenement* ***')' '['*** *guarde* ***']' '/'*** *activité*

Où les paramètres éventuels de l'événement sont séparés par des virgules, la garde (par défaut *vrai*) désigne une condition qui doit être remplie pour pouvoir tirer la transition, et l'activité exprimée dans une syntaxe laissée libre désigne des instructions à effectuer au moment du tir.

Le déclencheur de la transition est un événement de type *call*, *signal*, *change* ou *after*, ou laissé vide pour les transitions automatiques, comme décrit dans le paragraphe **Evénements**. L'événement peut porter des paramètres, visibles dans les activités associées à la transition ainsi que dans l'activité *exit* de l'état source et l'activité *entry* de l'état cible. Les événements entrants sont traités séquentiellement. Si aucune transition n'est déclenchée par l'événement il est détruit. Si une transition est déclenchée, on évalue sa garde ; celle-ci est exprimée en fonction des variables d'instance et éventuellement de tests d'état des instances d'objet accessibles (par exemple obj1 *in* Etat1 ou obj1 *not in* Etat1) ; si elle s'évalue à *faux* l'événement est de nouveau détruit. Si plusieurs transitions sont simultanément franchissables, l'une d'entre elles est choisie de façon arbitraire.

Représentation graphique des transitions ? à mettre ? signal, activité associée dans un carré… cf. p.631 et 632 de SuperStructure. Pas supporté par les outils, pas décrit dans UML 2 reference… Sera décrit dans activités, donc ajouter un lien peut suffire.

1.4. Points de décision

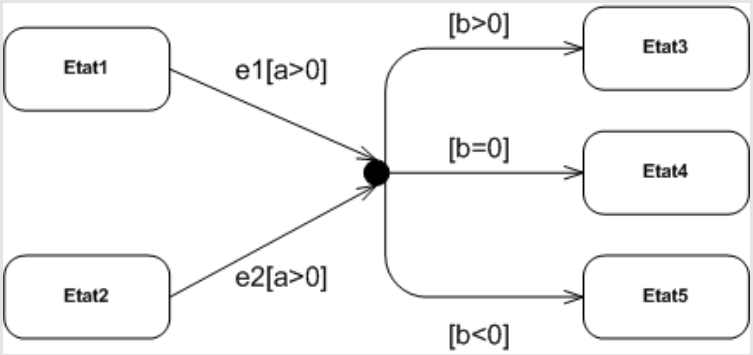
Mise en forme : Puces et numéros

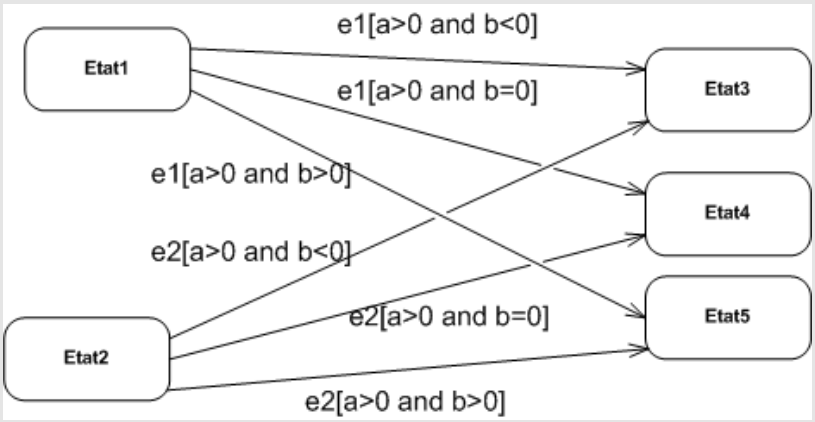
Il est possible de représenter des alternatives pour le franchissement d'une transition. On utilise pour cela des pseudo-états particuliers : les points de jonction (représentés pas un petit cercle plein) et les points de choix (représentés par un losange).

Les points de jonction sont un artefact graphique qui permet de partager des segments de transitions. Plusieurs transitions peuvent viser et/ou quitter un point de jonction. Tous les chemins à travers le point de jonction sont potentiellement valides. On peut donc représenter un comportement équivalent en créant une transition pour chaque paire de segments avant et après le point de jonction. L'intérêt est de permettre une notation plus compacte, et de rendre plus visible les chemins alternatifs.

Exemple : Equivalences de représentations

Les deux représentations suivantes sont équivalentes.





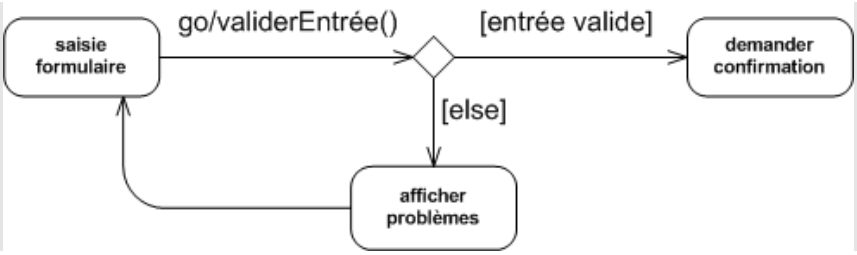
On admettra facilement que l'utilisation des points de jonction peut rendre les diagrammes plus lisibles.

Il faut noter que cette équivalence avec une représentation par plusieurs transitions implique que pour emprunter un chemin, toutes les gardes le long de ce chemin doivent s'évaluer à « vrai » *avant* de débuter le franchissement.

Au contraire, on utilise un point de choix (dit parfois point de choix dynamique), représenté par un losange, si les gardes après le point de choix doivent être évaluées *au moment où le point de choix est atteint*. Si, quand le point de choix est atteint, aucun segment après le point de choix n'est franchissable, le modèle est mal formé. Au contraire si plusieurs segments sont franchissables, on suit la règle habituelle quand plusieurs transitions sont simultanément franchissables : l'une d'entre elle est choisie aléatoirement. Ceci permet en particulier de baser le choix sur des résultats obtenus en franchissant le segment avant le point de choix.

Exemple d'utilisation d'un point de choix dynamique

Un formulaire en ligne est rempli par un utilisateur. Quand il valide son formulaire par le bouton « go », une vérification de la cohérence des données fournies est réalisée par *validerEntrée()*. Si les informations paraissent correctes on lui demande de confirmer, sinon on affiche les erreurs détectées et il doit resaisir son formulaire. Notez que la validation se fait sur le segment avant le point de choix, ce fonctionnement ne peut être décrit par un simple point de jonction.

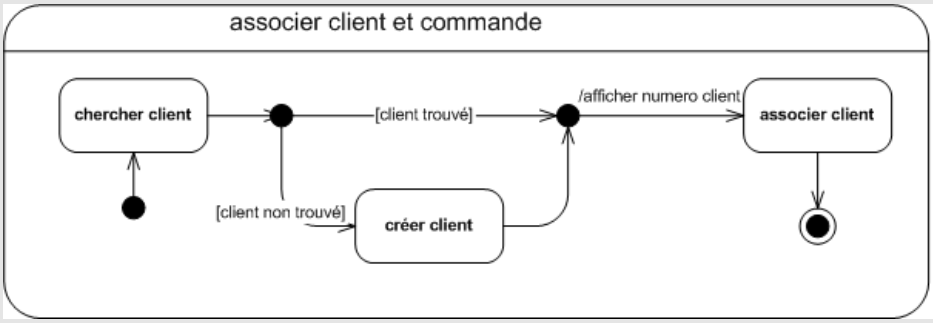


Il est possible d'utiliser une garde particulière sur un des segments après un point de choix ou de jonction : [*else*]. Ce segment n'est franchissable que si les gardes des autres segments sont toutes fausses. L'utilisation du *else* est recommandée après un point de choix pour garantir que le modèle est bien formé.

Si l'on utilise un point de jonction pour représenter le branchement d'une clause conditionnelle, il est conseillé d'utiliser également un point de jonction pour faire apparaître la fin de l'embranchement, pour être homogène.

Exemple de point de décision représentant des alternatives

Le point de décision est bien adapté à la représentation de clauses conditionnelles de type if/endif.



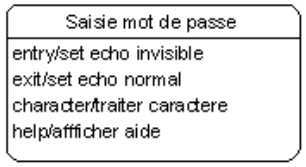
2. La Hiérarchie dans les machines à états

2.1. Etats et transitions internes

Un état est une période dans la vie d'un objet où il vérifie une condition donnée, exécute une certaine activité, ou plus généralement attends un événement. Conceptuellement, un objet reste donc dans un état durant une certaine durée, au contraire des transitions qui sont vues comme des événements ponctuels (sauf dans le cas particulier où la transition déclenche elle-même un traitement).

Un état peut être décomposé en deux compartiments séparés par une barre horizontale. Le premier compartiment contient le nom de l'état, le second contient les *transitions internes* à l'état, ou activités associées à cet état. On peut omettre la barre de séparation s'il n'y a pas de transitions internes à l'état. Une transition interne ne modifie pas l'état courant, mais suit globalement les règles d'une transition simple entre deux états. Trois déclencheurs particuliers sont introduits permettant le tir de transitions internes : *entry/, do/,* et *exit/.*

Exemple 2 : Représentation d'un état simple



Notation : Transitions internes

La syntaxe pour la définition d'une transition interne est la suivante :

Nom-évenement ***('****liste-paramètres****)' '[****'condition-d'activation****]' '/'*** *activité-à-réaliser*

Le nom de l'événement peut être le nom d'une méthode de l'objet auquel appartient l'état, d'un événement déclaré comme tel au niveau paquetage, ou un des mots clés réservés suivants :

- **entry :** activité à effectuer à chaque fois que l'on rentre dans cet état
- **exit :** activité à effectuer quand on quitte cet état
- **do :** identifie une activité continue qui est réalisée tant que l'on se trouve dans cet état, où jusqu'à ce que le calcul associé soit terminé. On pourra dans ce dernier cas gérer l'événement correspondant à la fin de cette activité (« *completion event »)*
- **include :** permet d'invoquer un sous diagramme états transitions

La liste des paramètres (optionnelle) correspond aux arguments de l'événement déclencheur de l'activité. La condition d'activation, ou garde, est une condition booléenne qui permet d'autoriser ou non le déclenchement de l'activité. La façon de spécifier l'activité à réaliser est laissée libre à l'utilisateur ; en général on utilisera du langage naturel documentant l'activité à entreprendre, ou du pseudo code qui utilise potentiellement les arguments de l'événement déclencheur.

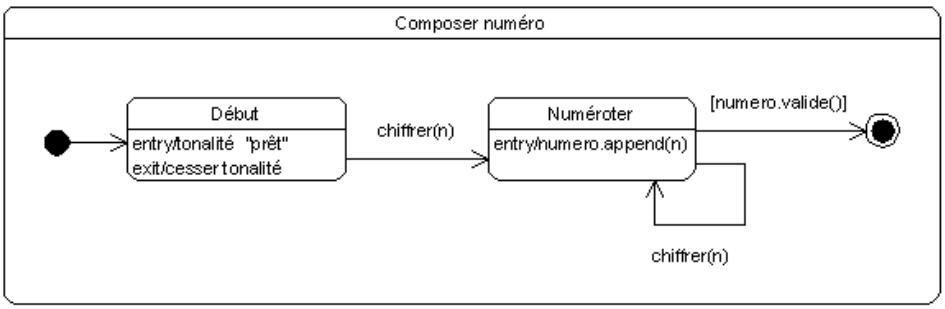
A noter qu'une transition interne se distingue d'une transition représentée par un arc qui boucle sur l'état, car lors de l'activation d'une transition interne, les activités *entry* et *exit* ne sont pas appelées (on ne quitte pas l'état courant). Implicitement, tout diagramme d'état est contenu dans un état externe qui n'est usuellement pas représenté ; ceci apporte une plus grande homogénéité dans la description : tout diagramme est implicitement un sous-diagramme.

2.2. Etats Composites

Un état composite, par contraste avec les états dits *simples*, est graphiquement décomposé en deux ou plusieurs sous-états. Tout état ou sous-état peut ainsi être décomposé en sous-états enchaînés sans limite a priori de profondeur. Un état composite est noté par les deux compartiments de nom et d'actions internes habituelles, et d'un compartiment contenant le sous diagramme.

Exemple 3 : Combiné téléphonique

Dans cet exemple nous représentons les étapes de l'action « composer numéro ». A l'entrée dans l'état composite (par exemple par une action *décrocher*), une tonalité sonore annonce que le téléphone est prêt à composer. Les chiffres du numéro de téléphone sont saisis un par un par l'appel à l'opération *chiffrer*. On peut noter la transition automatique de *numéroter* vers l'état final, qui est franchie dès que sa garde devient vraie.

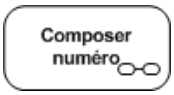


Un nouvel objet est créé dans l'état initial le plus externe du diagramme, et franchit la transition par défaut qui en part. Un objet qui atteint l'état final le plus externe est détruit. Ces transitions peuvent être étiquetées par un événement correspondant au constructeur ou destructeur d'instance, et éventuellement associées à une activité.

Une transition qui atteint l'état final d'un sous diagramme correspond à la fin de l'action associée à l'état l'encapsulant. La fin d'une activité interne à un état peut être associée au déclenchement d'un changement d'état, représenté par une transition sans étiquette qui quitte l'état externe courant. On parle alors de *transition automatique*, déclenchée par un événement implicite de terminaison ou *completion event*. Un *completion event* est également déclenché par la fin d'une activité interne de type *do/*.

L'utilisation d'états composites permet de développer une spécification par raffinements. Il est parfois souhaitable de séparer la définition des sous-états de l'utilisation de l'état englobant. On peut noter graphiquement le fait qu'un état est un état composite dont la définition est donnée sur un autre diagramme.

Exemple : Notation abrégée des états composites.



2.3. Transitions et états composites

Les transitions peuvent avoir pour cible la frontière d'un état composite, et sont alors équivalentes à une transition ayant pour cible l'état initial de l'état composite. De même une transition ayant pour source la frontière d'un état composite est équivalente à une transition qui s'applique à tout sous-état de l'état composite source. Cette relation est transitive : la transition est franchissable depuis tout état imbriqué, quelle que soit sa profondeur. Si la transition ayant pour source la frontière d'un état composite ne porte pas de déclencheur explicite (déclenchée par un *completion event),* elle est franchissable quand l'état final de l'état composite est atteint.

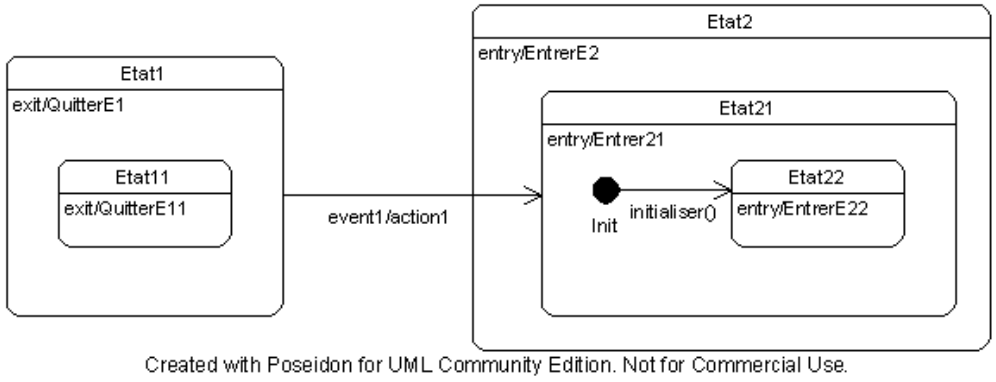
Par exemple la transition *minimiser* de la fenêtre d'application (p.2) est franchissable depuis les états *normale* et *maximisée*.

Les transitions peuvent également toucher des états de différents niveaux d'imbrication, en traversant les frontières des états.

Dans tout les cas, avant d'entrer dans un état les activités *entry/* sont réalisées, et avant de le quitter les activités *exit/* sont réalisées. En cas de transition menant vers un état imbriqué les activités *entry/* de l'état englobant sont réalisées avant celles de l'état imbriqué. En cas de transition depuis la frontière de l'état englobant, les activités *exit/* du sous-état actuellement actif sont réalisées, puis celles de l'état englobant.

Exemple : Ordre d'appel

Depuis l'état *Etat11*, la réception de l'événement *event1* provoque la séquence d'activités : *QuitterE11,QuitterE1,action1,EntrerE2,EntrerE21,initialiser,EntrerE22*, et place le système dans l'état *Etat22*.



2.4. Historique et états composites

Chaque région ou sous-diagramme état-transition peut être muni d'un pseudo état permettant l'historique, noté par un cercle contenant un **'H'**. Une transition ayant pour cible le pseudo-état historique est équivalente à une transition qui aurait pour cible le dernier état visité dans la région contenant le **'H'**.

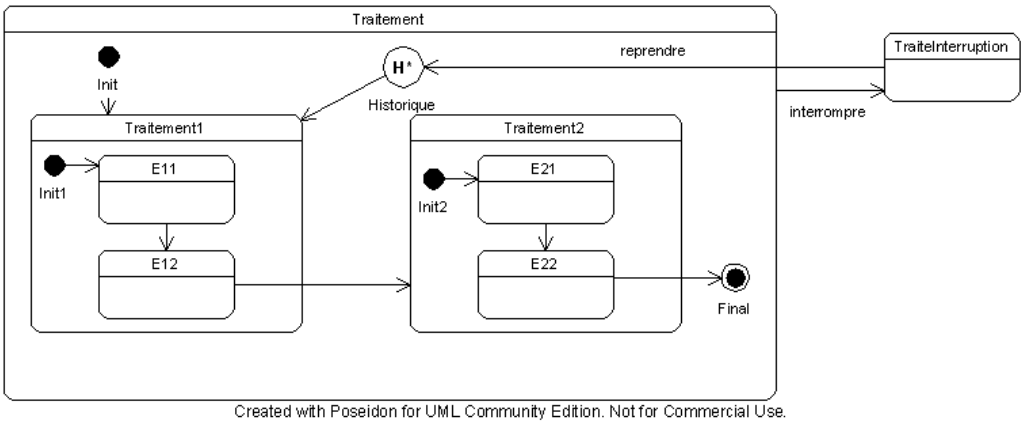
Dans l'exemple de la fenêtre, en tête de ce chapitre, il permet de retrouver l'état précédent (*normale* ou *maximisée*) quand on sort de l'état *minimisée*.

Il est possible de définir un dernier état visité par défaut, par une transition ayant pour source le pseudo-état **H.** Cet état par défaut sera utilisé si la région n'a pas encore été visitée.

Il est également possible de définir un pseudo-état historique profond noté par un cercle contenant **'H*'**. Cet historique profond permet d'atteindre le dernier état visité dans la région, quel que soit son niveau d'imbrication, au lieu d'être limité comme le **H** aux états de son niveau d'imbrication dans la région.

Exemple : Historique

L'utilisation d'un historique profond permet de retrouver après une interruption le sous-état précédent. A noter que l'utilisation d'un historique de surface **'H'** au lieu de **'H*'** permettrait de retrouver l'état *Traitement1* ou *Traitement2* dans leur sous-état initial, mais pas le sous-état imbriqué *E11, E12, E21, E22*, qui était occupé avant l'interruption. Toute région peut être munie à la fois d'un historique profond et d'un historique de surface.



2.5. Interface des états composites

Pour cacher la complexité, il est possible de masquer sur un diagramme les sous-états d'un état composite, et de les définir dans un autre diagramme. Pour exprimer la connection des diagrammes entre eux, on peut utiliser des points de connexion. On dénote graphiquement les points d'entrée par un cercle vide sur la frontière de l'état, et les points de sortie par un cercle barré sur la frontière de l'état.

Si on veut utiliser le comportement « par défaut » d'une machine à état, c'est-à-dire entrer par l'état initial par défaut, et considérer les traitements finis quand l'état final est atteint, il est inutile de se servir de ces points de connexion. On peut se servir plus simplement de transitions ayant pour cible la frontière de l'état englobant.

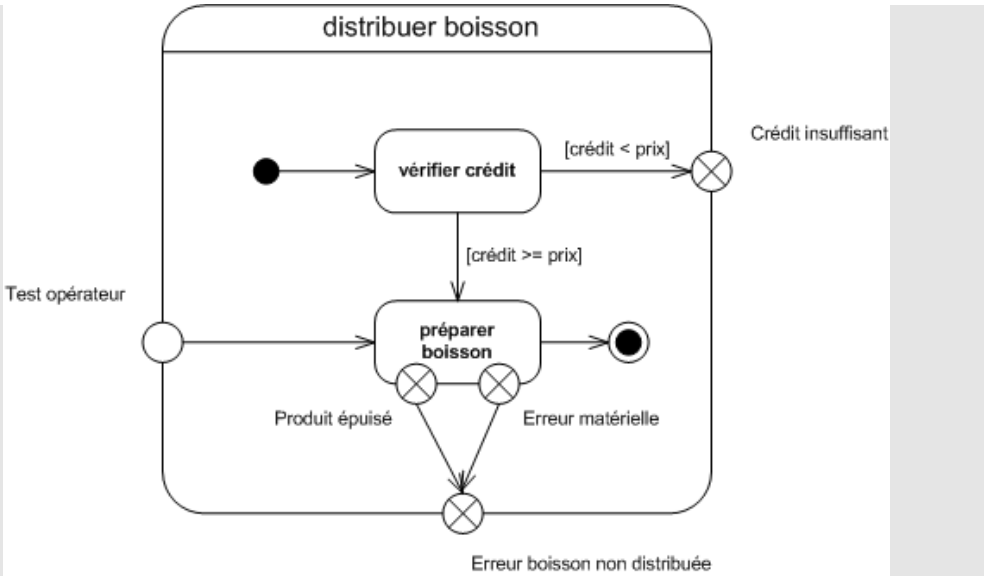
On s'en sert s'il existe plusieurs façons d'entrer ou de sortir de la machine à états, donc typiquement pour représenter des transitions traversant la frontière de l'état englobant qui viseraient directement un sous-état autre que l'état initial, ou ayant pour source un sous-état et qui viseraient un état externe.

Un point de connexion n'est qu'une référence vers un état défini ailleurs. L'état qu'il désigne est signalé par l'unique transition quittant le pseudo-état. Cette transition peut éventuellement être étiquetée par une action (qui sera éxecutée avant l'activité *entry* de l'état cible), mais pas d'une garde ni d'un déclencheur explicite : c'est le prolongement de la transition qui vise le point de connexion.

Plusieurs transitions peuvent cibler un même point de connexion, ceci peut rendre les diagrammes plus lisibles.

Exemple : Points de connexion

Cet exemple exhibe l'utilisation des points de connexion. L'état « distribuer boisson » comporte deux entrées et trois sorties. L'entrée par défaut vérifie d'abord que le crédit de l'utilisateur est suffisant pour la boisson sélectionnée, et peut provoquer une sortie sur erreur par le point de connexion « crédit insuffisant ». Un deuxième point d'entrée est fourni pour l'opérateur de maintenance qui peut déclencher la préparation d'une boisson sans insérer d'argent. La préparation de la boisson peut elle-même être soumise à des erreurs. Dans le cas nominal, la boisoon est préparée et la sortie de l'état « distribuer boisson » se fait par l'état final.



Les points de connexion sont plus que des facilités de notation, ils permettent de découpler la modélisation du comportement interne à un état de la modélisation d'un comportement plus global. Les points de connexions sont la façon de représentent l'interface (au sens objet) d'une machine à états, en masquant l'implémentation du comportement. Ceci permet un développement par raffinements (ou en « bottom-up ») en deux étapes indépendantes du processus de conception. C'est donc essentiel pour permettre de traiter des modèles de grande taille.

3. Contrat de comportement

UML 2.0 introduit la notion de contrat de comportement, ou protocole d'utilisation pour un classeur. On utilise pour cela une version simplifiée des diagrammes état-transition, et l'on note avec le mot clé *{protocol}* la nature du diagramme.

Un contrat de comportement définit les séquences d'actions légales sur un classeur, ainsi qu'un certain nombre de pré et post conditions qui doivent être validées. Un contrat de comportement est relativement abstrait, il n'exprime pas la nature des traitements réalisés, simplement leur séquencement logique. Ainsi, les états d'un diagramme de type *{protocol}* ne comportent pas de clauses déclenchant des activités (*entry/, exit/, do/*), et les transitions ne comportent pas d'activités à déclencher.

Notation : Transitions de protocole

La syntaxe pour l'étiquette d'une transition liant un état source E1 à un état destination E2 sur un diagramme de type *{protocol}* est la suivante :

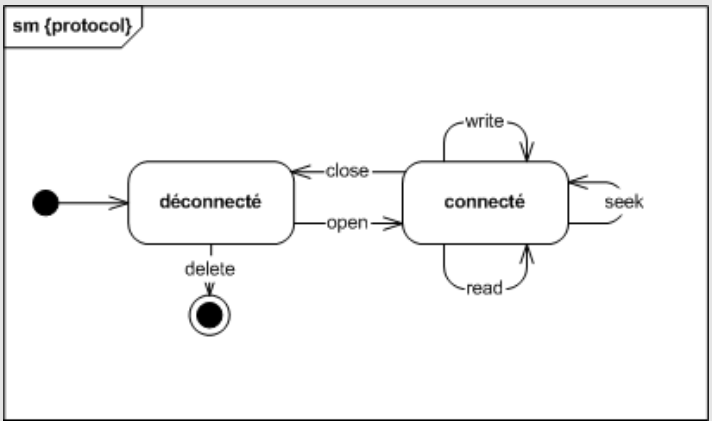
'[' pré** condition **']'** Déclencheur '/' '[' **post** condition **']'

Une telle transition signifie que quand le classeur est dans l'état protocolaire E1, si la pré-condition est vraie, la réception de l'événement *déclencheur* doit amener dans l'état E2, et qu'à l'issue du franchissement, la post-condition doit être vraie.

Les protocoles servent à expliquer la façon correcte de se servir d'une classe ou d'un composant, sans spécifier les traitements qui réalisent l'action. On peut facilement avoir plusieurs implémentations différentes d'une classe qui respectent le protocole.

Exemple : Protocole d'une classe Fichier

Ce diagramme décrit la façon de se servir d'une instance de classe descripteur de fichier. Avant de pouvoir l'utiliser vraiment (lectures, écritures, positionnement du curseur dans le fichier) il faut d'abord l'associer à un fichier àl'aide de *open.* Avant de détruire l'instance, l'utilisateur doit appeler la méthode *close*, pour éviter d'éventuels problèmes d'entrées/sorties.



4. Gestion de la concurrence

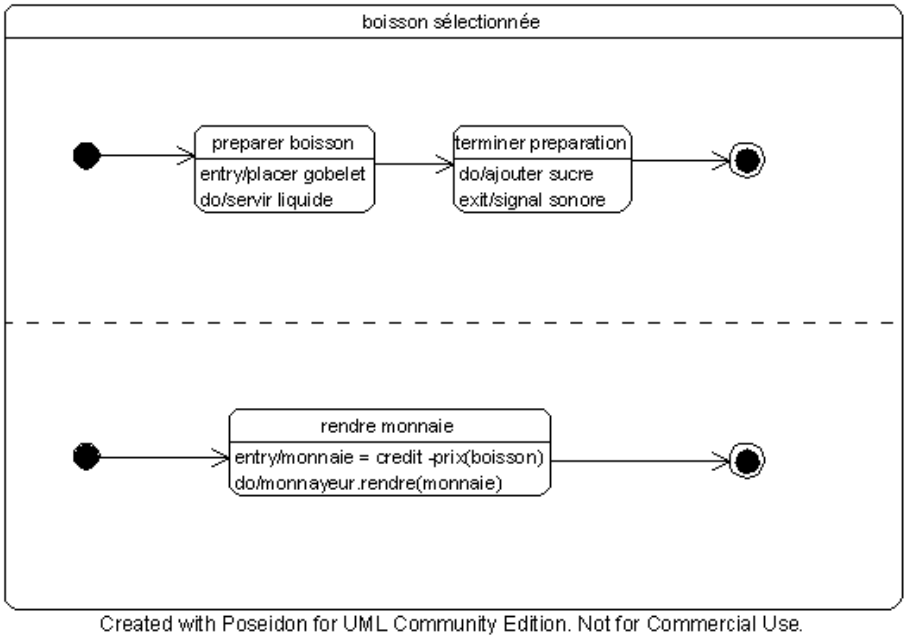
Les diagrammes état transition offrent un bon support pour la description de mécanismes concurrents. Un état composite qui comporte des sous-états peut également être munis d'un comportement concurrent, à travers l'utilisation de *régions concurrentes*. Les régions concurrentes permettent de représenter des zones ou l'action est réalisée par des flots à l'exécution parallèle.

Chaque *région* d'un état peut avoir un état initial et final. Une transition qui atteint la bordure d'un état composite est équivalente à une transition qui atteindrait l'état initial du sous-diagramme, ou les états initiaux de toutes ses régions concurrentes si elles existent. Les transitions ayant pour origine un état initial interne ne sont pas étiquetées, et correspondent à toute transition atteignant la frontière de l'état externe.

Dans le cas d'un état présentant des régions concurrentes, toutes ses régions doivent atteindre leur état final pour que l'action soit considérée terminée (génération d'un *completion event*).

Exemple : Etat concurrent

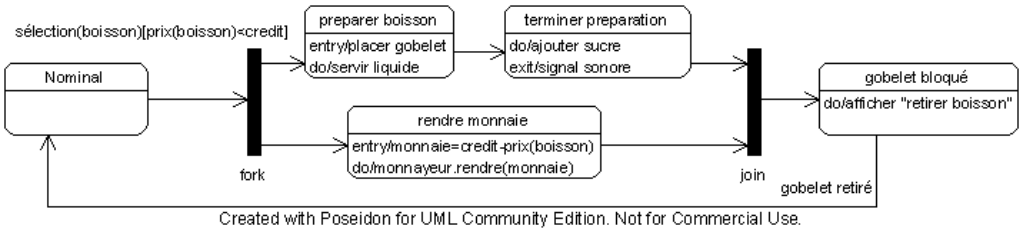
Cet exemple exhibe un état concurrent, au sein d'un distributeur type machine à café. Quand la boisson a été sélectionnée et le montant validé par rapport au crédit, deux séquences d'actions sont déclenchées en parallèle : la préparation de la boisson et le rendu de la monnaie.



Il est également possible de représenter ce type de comportement à de transitions concurrentes dites complexes. Ces transitions sont représentées par une barre verticale épaisse et courte, possiblement associée à un nom. Un ou plusieurs arcs peuvent avoir pour origine ou destination la transition. La sémantique associée est celle d'un *« fork/join »*.

Exemple : Transition concurrente

L'exemple d'état concurrent du distributeur de boisson ci-dessus peut être spécifié de façon équivalente à l'aide de deux transitions concurrentes. La transition nommée ici *fork* correspond à la création de deux tâches concurrentes qui sont créées dans les états « préparer boisson » et « rendre monnaie ». La transition nommée *join* correspond à une barrière de synchronisation par rendez-vous. Les taches concurrentes ne peuvent continuer leur exécution avant que toutes ne soient prêtes à franchir la transition de rendez-vous.



Dans le cas d'une transition ayant pour cible un état cible muni de régions concurrentes, la transition devient équivalente à une transition complexe de type *fork* ayant pour cibles les états initiaux de chaque région concurrente. Dans le cas d'un état source muni de régions concurrentes, la transition devient équivalente à une transition complexe de type *join* ayant pour sources les états finaux de chaque région concurrente.

5. Conclusion

Les diagrammes état-transition sont bien adaptés à la représentation du comportement interne d'un objet, à fortiori actif, sous une forme qui met en avant le modèle événementiel ou *réactif* des traitements. Ce modèle est bien adapté au génie logiciel orienté objet, de par les mécanismes qu'il propose (*call, signal, after*), qui permettent de faire le lien avec les autres diagrammes de la norme. La grande flexibilité apportée par la possibilité de définir des *transitions internes* (*entry, do, exit)*, et par les mécanismes de hiérarchisation des diagrammes, permet de représenter de façon concise et formelle l'ensemble des comportements que peut adopter l'instance modélisée.

De ce point de vue, c'est le seul diagramme de la norme à offrir une vision complète et non ambiguë de l'ensemble des comportements, les diagrammes d'interaction n'offrant que la vue d'un scénario, sans vraiment préciser comment les différents scénarios ébauchés peuvent interagir entre eux. Cependant le grand niveau de détail et la lourdeur relative de ces diagrammes les rendent surtout adaptés à la phase de réalisation, et l'absence de vision globale du système rend difficile la modélisation de systèmes composés de plusieurs sous-systèmes.

Au contraire, les diagrammes de protocole donnent une vision de haut niveau du comportement d'un composant. C'est un élément important qui permet de spécifier les *responsabilités* d'un composant, sans rentrer dans les détails de son implémentation. Ceci est relativement proche de la notion d'interface d'un composant, mais apporte une information supplémentaire par rapport à la seule description des signatures des opérations qu'il porte : elle ajoute des contraintes sur l'ordre d'appel aux opérations déclarées.