Etude des systèmes à évènements discrets : SED.

Introduction.

Définitions

a- Par opposition aux systèmes dynamiques dont l'évolution est **continue dans le temps** et peut être décrite par des **équations différentielles**, les Systèmes à Evénements Discrets (**SED**) sont des systèmes dynamiques dont l'espace d'états est un ensemble **discret** (*un objet est dit **discret** lorsque ses points sont écartés les uns des autres, le contraire étant dénommé un objet **continu***) et dont les transitions entre états sont associées à des événements.

Des théories et des modèles spécifiques à cette classe de systèmes dynamiques sont nécessaires pour les modéliser, analyser leurs performances et les commander.

b- **Une machine à état** est **un comportement** qui spécifie une séquence d'états qu'un objet traverse pendant sa durée de vie, en réponse aux événements et aux réponses qu'un objet donné donne à ces événements.
Un diagramme de machine à état dépeint une machine d'état.

Remarque

Pour aborder ce cours la connaissance des diagrammes d'états ainsi que la notion de structure algorithmique sont nécessaires.

1 : La machine d'état

11 : Présentation

La **machine à nombre fini d'états** (FSM : Finite State Machine), est aussi appelée **automate fini**. Elle peut être une entité matérielle (un microprocesseur par exemple), mais aussi une entité conceptuelle comme un algorithme. Elle comporte un nombre fini d'états
.
La machine à états est un système à évènement discrets, capable de **mémoriser** des données, de les **traiter** et de les **restituer** selon des scénarios définis au préalable.

Elle peut être définie par :

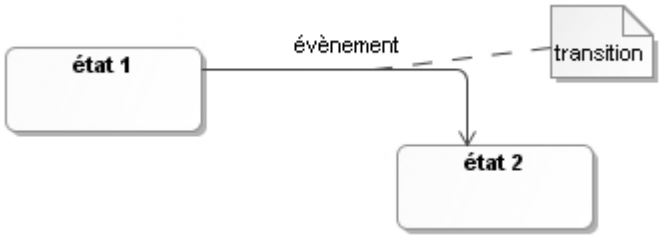
- un **état initial**
- un ensemble d'**évènements**
- un ensemble d'**états**
- un ensemble d'**activités**
- un ensemble de **règles** permettant de déterminer le comportement du système.

Dans un état, un système peut avoir une activité ou être en attente. Les états d'un système se succèdent en fonction d'évènements.

Un **évènement** est une description d'occurrence qui conduit à une évolution du comportement du système. On l'appelle aussi un déclencheur (trigger).

12 : Le graphe d'état

Le graphe d'état est utilisé pour décrire le **comportement** d'une machine d'état.
Le **nœud** symbolise un **état interne**. Il montre ce qui se passe. L'arc qui relie deux nœuds est appelé **transition**.
Il indique la possibilité de passer d'un état à un autre lorsqu'un **évènement** se produit. **Le système ne peut être que dans un seul état**.



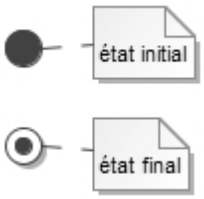
13 : Le diagramme d'états

131 : Présentation

Le diagramme d'états (state machine diagram ou **stm**) est un diagramme normalisé SysML.
Il permet de décrire le comportement d'un système ou d'une de ses parties.
Le diagramme d'états est donc rattaché à un bloc ou plutôt à son instance (objet du type du bloc).

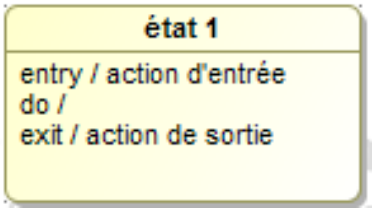
132 : Etat initial/état final

L'**état initial** correspond à la création de l'instance du bloc pour lequel le diagramme d'état est spécifié.
L'**état final** correspond à la destruction de cette instance de bloc. Il peut y en avoir plusieurs dans un diagramme d'états. En effet, plusieurs scénarios peuvent être possibles pour mettre fin à un comportement



133 : Activité et action

A un état, on peut principalement rattacher une activité, une action d'entrée et une action de sortie.
Une **activité** peut être considérée comme une unité de comportement. Elle prend du temps et peut être interrompue. On la trouve à l'intérieur des nœuds du diagramme (mot clé « do »).

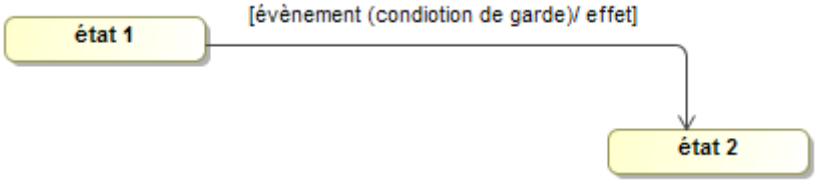


A contrario, une **action** ne prend pas de temps et ne peut pas être interrompue. Son exécution peut par exemple provoquer un changement d'état, l'émission d'un ordre pour un préactionneur ou un retour de valeur. On peut les trouver dans les transitions (effet) ou dans les états (mots clé « entry » ou « exit »). Les actions sont les éléments de base permettant de spécifier les activités dans des diagrammes d'activité.

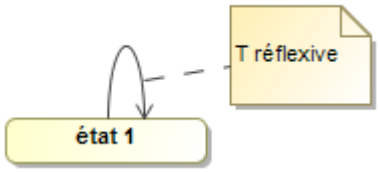
Dans l'outil SysML, les activités et actions sont directement liées aux fonctions (opérations) définies dans les propriétés de bloc

134 : Evènement, condition de garde et effet

Une **transition** peut être associée à un **évènement**, à une **condition de garde** *(expressions logique qui définissent si cette transition doit être exécutée*) et / ou à un **effet** (*il s'agit d'une activité (action) facultative à réaliser au déclenchement de la transition).*



Une transition réflexive entraîne une sortie d'état puis un retour dans ce même état. Cela n'est donc pas sans conséquences selon les cas.



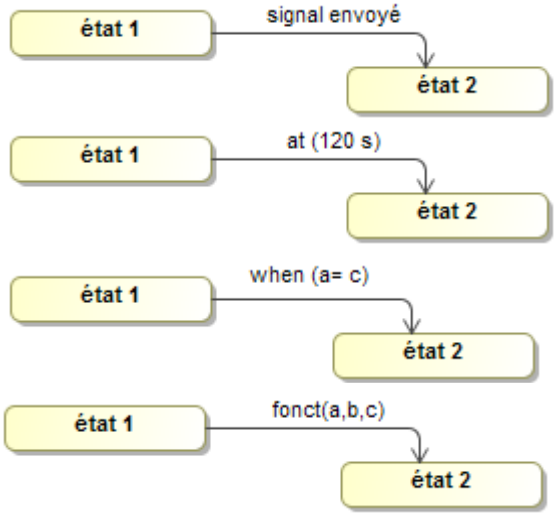
Il existe quatre types d'évènements associés à une transition :

– le **message** (signal event) : un message asynchrone est arrivé,

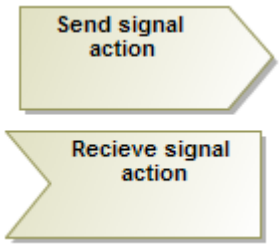
– l'**évènement temporel** (time event) : un intervalle de temps s'est écoulé depuis l'entrée dans un état (mot clé « after ») ou un temps absolu a été atteint (mot clé « at »),

– l'**évènement de changement** (change event) : une valeur a changé de telle sorte que la transition est franchie (mot clé « when »),

– l'**évènement d'appel** (call event) : une requête de fonction (opération) du bloc a été effectuée. Un retour est attendu. Des arguments (paramètres) de fonction peuvent être nécessaires.



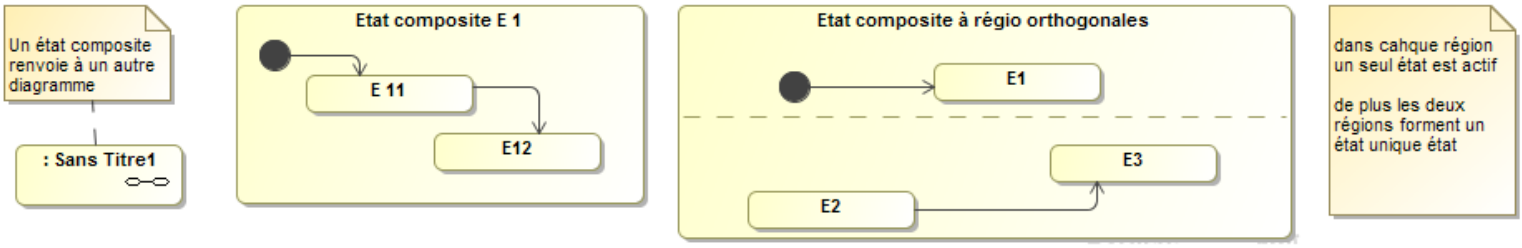
Les évènements peuvent être utilisés pour décrire les interactions entre les différents blocs d'un système.
En effet, un évènement peut être émis par un autre bloc (send signal action) que celui pour lequel le diagramme d'états est spécifié. Il est alors par défaut destiné à tous les blocs du système (diffusion « broadcast) ». De même, un évènement peut être reçu (receive signal action).



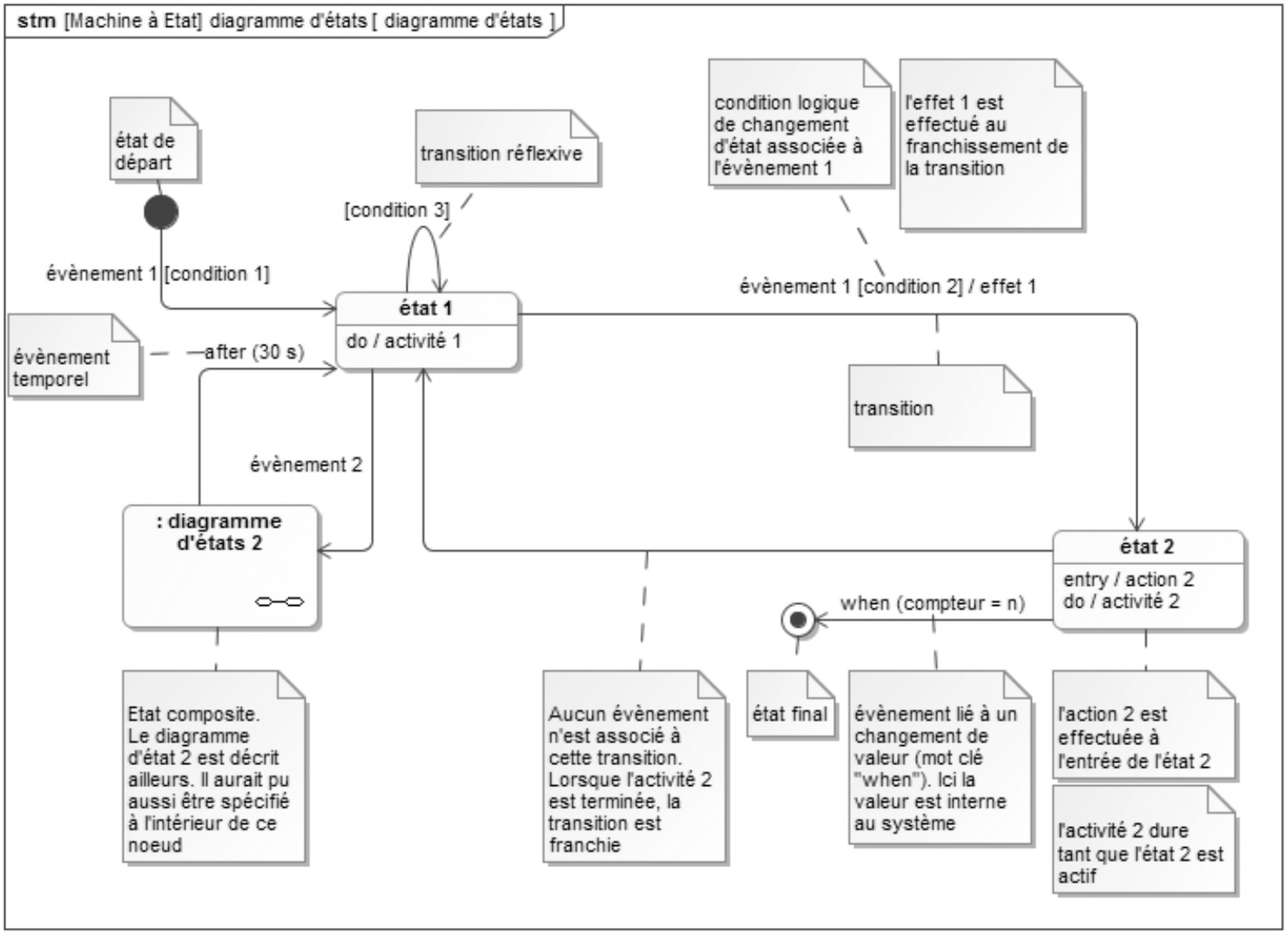
La **condition** de garde est une expression booléenne faisant intervenir des entrées et / ou des variables internes. Elle autorise le passage d'un état à un autre. Il est possible d'utiliser les notations non booléennes de **front montant** (↑) et **front descendant (**↓).
L'**effet** associé à une transition est effectué lorsque la transition est franchie.

135 : Etat composite

Un **état composite** est constitué de sous-états liés par des transitions. Cela permet d'introduire la notion d'état de niveau hiérarchique inférieur et supérieur.



136 : Synthèse du formalisme SysML de base pour les diagrammes d'états



137 : Les pseudo-états

Un **pseudo-état** est un élément de commande qui influence le comportement d'une machine d'état.

Ils peuvent être utilisés dans un **diagramme d'états** ou dans un **diagramme d'activité**.

Le formalisme SysML admet neuf pseudo-états :

- « shallow history » : permet à un état de niveau hiérarchique supérieur (état composite) de se souvenir du dernier sous-état, avant qu'il n'évolue vers un autre état,

- « deep history » : idem que précédemment mais avec la propagation de l'historique à tous les sous-états composites de niveaux hiérarchiques inférieurs,

- « fork » et « join » : divergence et convergence de séquences parallèles,

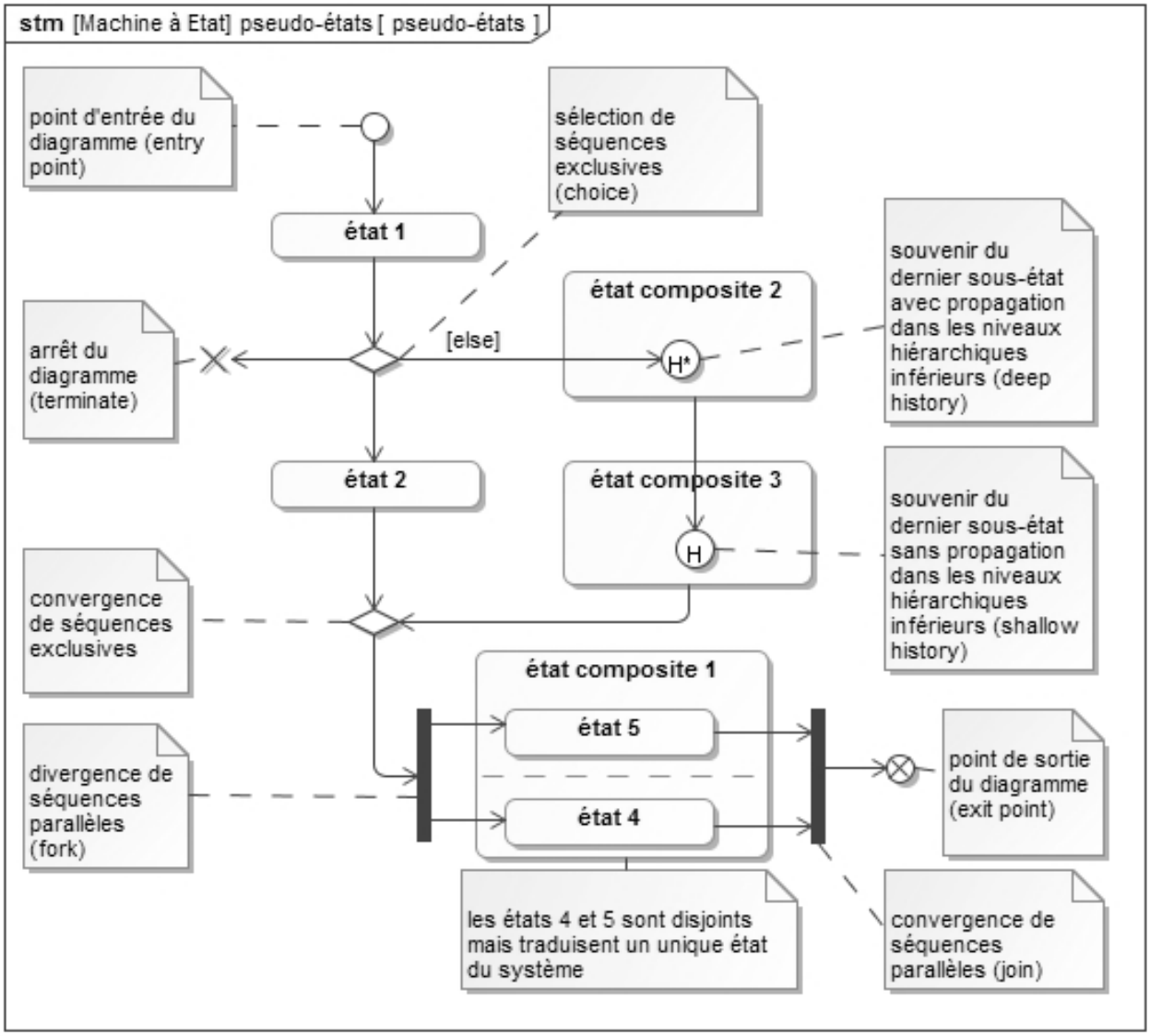
- « choice » : sélection et convergence de séquences exclusives. Il est nécessaire qu'une condition située en aval soit vraie pour que l'évolution du système se poursuive.

Les conditions de gardes doivent être exclusives. Le mot clé « else » peut être utilisé pour englober tout ce qui n'est pas décrit dans les autres expressions booléennes. Les conditions de garde situées en aval sont toutes évaluées une fois le pseudo-état atteint,

- « junction » : idem au pseudo-état « choice », à la différence que pour qu'un chemin soit emprunté, toutes les conditions de garde situées en aval et en amont, doivent être vraies. L'évaluation des conditions avales est réalisée avant que le pseudo-état soit atteint,

- « entry point » et « exit point » : permet de créer un point d'entrée du diagramme et un point de sortie vers un autre diagramme,

- « terminate » : permet de terminer une séquence sans destruction de l'instance de bloc.

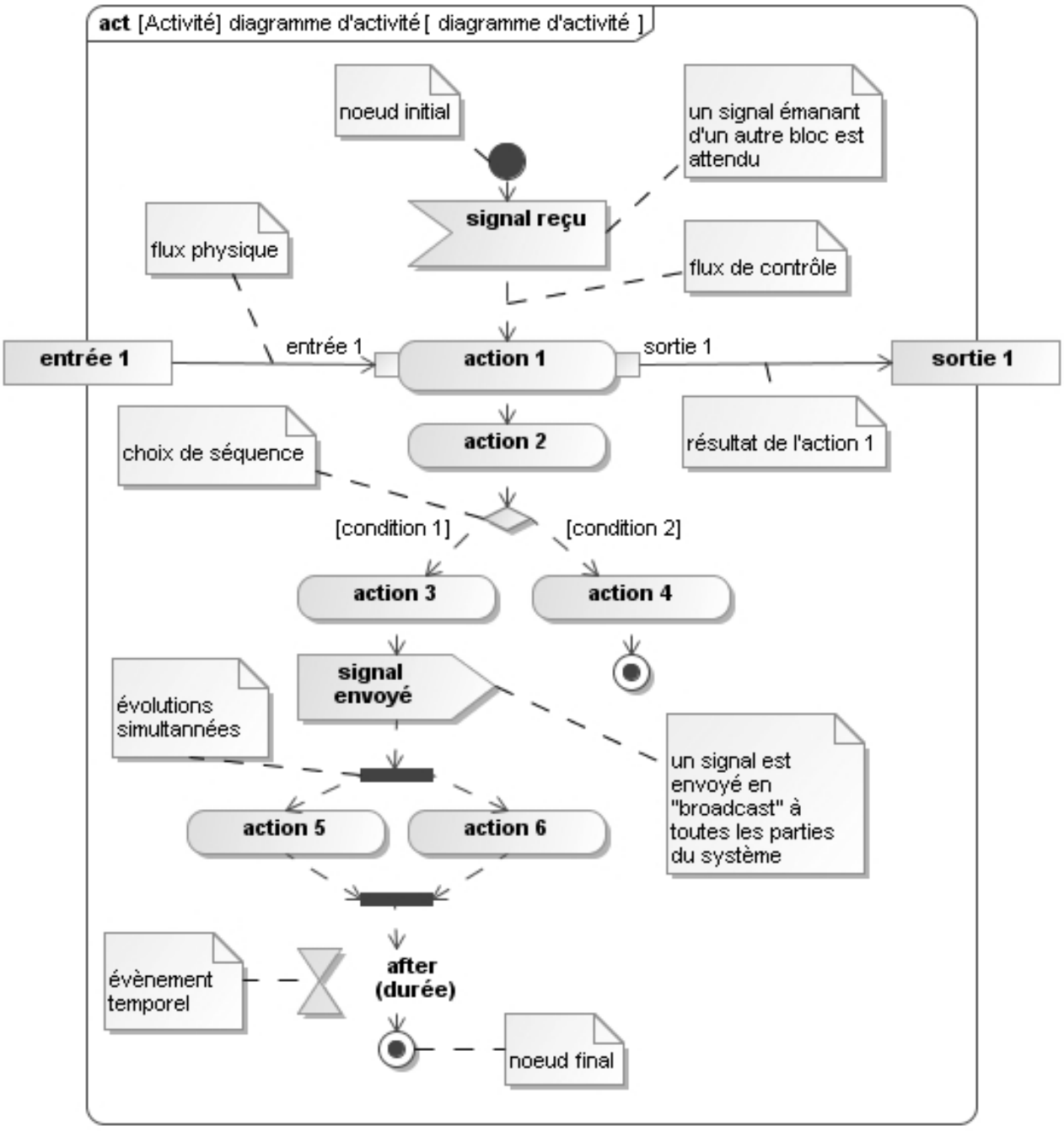


13 : Le diagramme d'activité

131 : Présentation

Le diagramme d'activité (activity diagram ou **act**) est un diagramme normalisé SysML
Il permet de décrire la transformation des **flux d'entrées** en **flux de sorties** (matières, énergies, informations) par le biais de séquences d'actions ou activité déclenchées par des **flux de contrôle**.
Le diagramme d'activité n'est pas explicitement au programme des classes préparatoires aux grandes écoles. Cependant, il semble essentiel de le présenter ici étant donné son importance fondamentale. La plus grande partie de son formalisme a déjà été décrit avec les diagrammes d'états. Enfin, il permet aussi de décrire les structures algorithmiques, étant elles, au programme. Nous nous limiterons ici à l'essentiel.

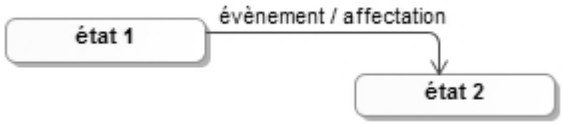
132 : Formalisme SysML de base pour les diagrammes d'activité

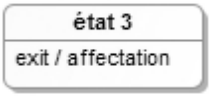


133 : Les structures des algorithmes de base.

L'affectation

L'affectation d'une valeur à une variable peut se faire à l'aide d'une action. Cela ne prend pas de temps significatif.





Formalisme du diagramme d'états *formalisme du diagramme d'activité*

Le groupe ou bloc d'instructions

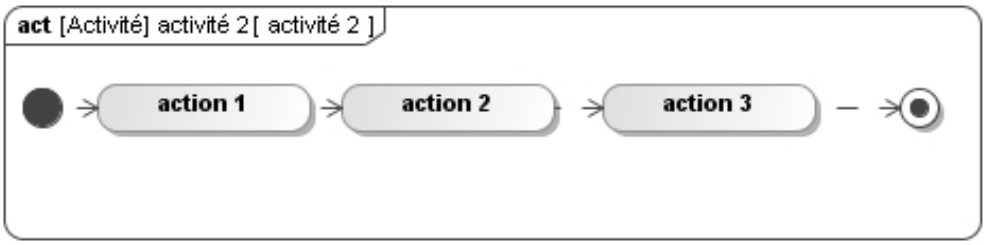
Un groupe ou un bloc d'instructions peut être une séquence d'un diagramme d'activité.

Cela correspond à une succession d'actions et / ou d'activités.

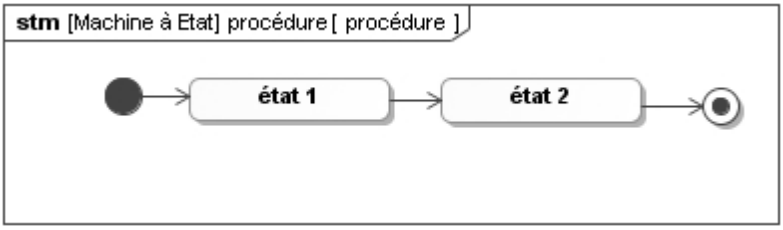
Fonctions et procédures

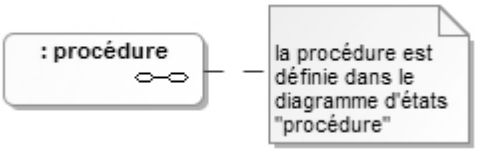
La décomposition d'un algorithme en fonctions et procédures, permet :
– d'une part, de scinder une problématique générale en plusieurs problématiques élémentaires,
– d'autre part, de pouvoir réutiliser des sous-programmes réalisant des tâches élémentaires.

Une **procédure** comporte une succession d'instructions mais ne renvoie rien.

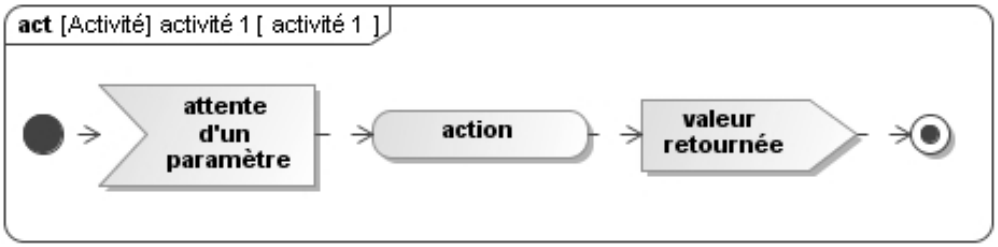


On peut aussi utiliser les états composites d'un diagramme d'états :



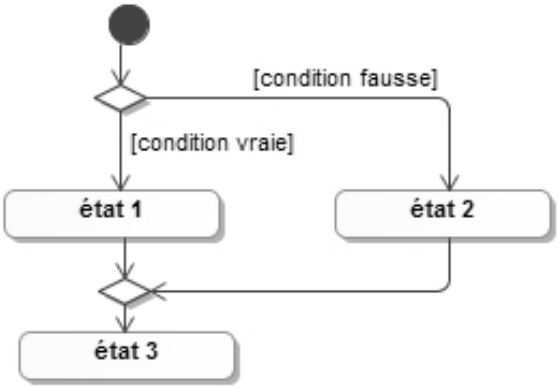


A la fin de l'exécution d'une **fonction**, il y a le retour d'une valeur, d'une liste, d'un objet, etc.

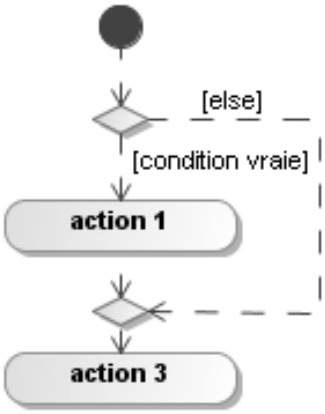


La structure alternative (conditionnelle)

Si …, alors faire …, sinon faire …



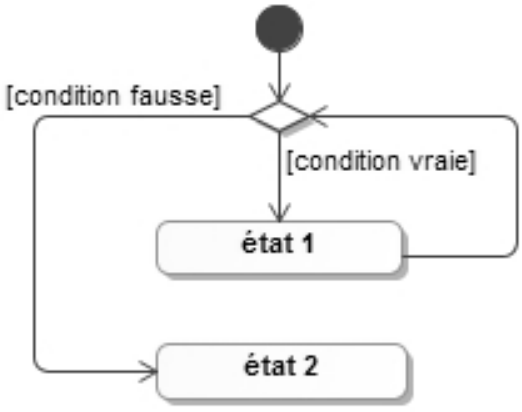
Structure alternative complète
Formalisme du diagramme d'états



Structure alternative avec saut
Formalisme du diagramme d'activité

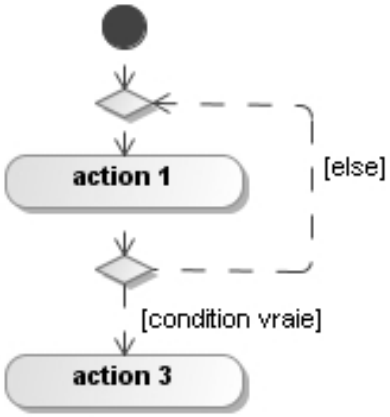
Les structures répétitives (itératives)

Tant que condition vraie, faire …



Formalisme du diagramme d'états

Répéter… jusqu'à condition vraie



Formalisme du diagramme d'activité

2 : Méthode

21 : Comment analyser un système à contrôle logique ?

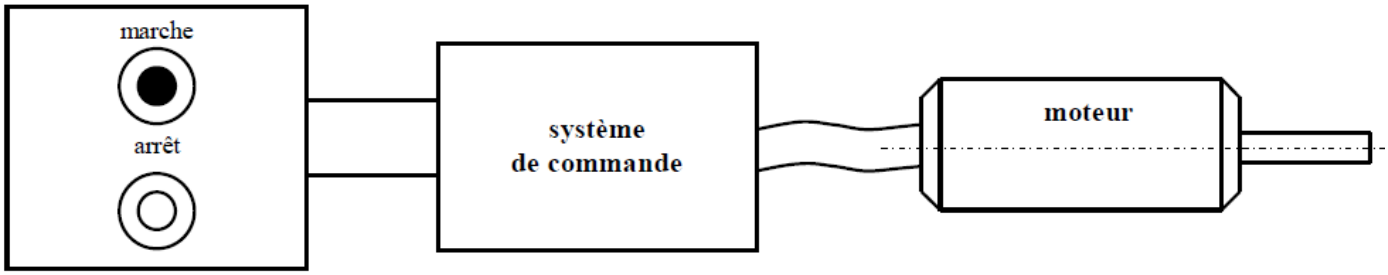
211 : Méthode : détermination de la nature d'un système à contrôle logique

On peut établir la table de vérité d'un système à contrôle logique en indiquant l'état des sorties pour toutes les combinaisons des entrées.
Si à un état du vecteur des entrées correspondent plusieurs états possibles du vecteur des sorties, le système est dit séquentiel. Dans le cas contraire, il est qualifié de combinatoire.

Il est aussi possible d'analyser un chronogramme montrant l'évolution des entrées et des sorties, pour donner la nature d'un système à contrôle logique. Il faut toutefois s'assurer que toutes les occurrences (apparition dans le temps) des entrées apparaissent, pour ne pas aboutir à des conclusions erronées.

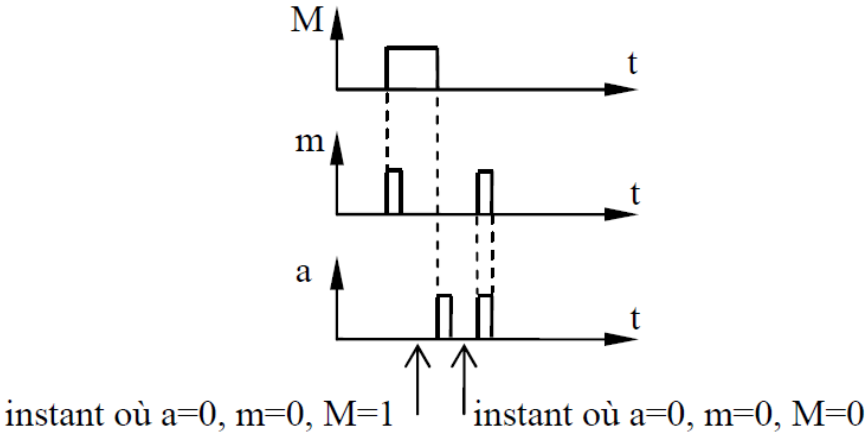
Exemple : commande tout ou rien (TOR) d'un moteur

Deux boutons poussoir « m » (marche) et « a » (arrêt) commandent le fonctionnement d'un moteur que l'on caractérise par une variable logique de sortie « M ».



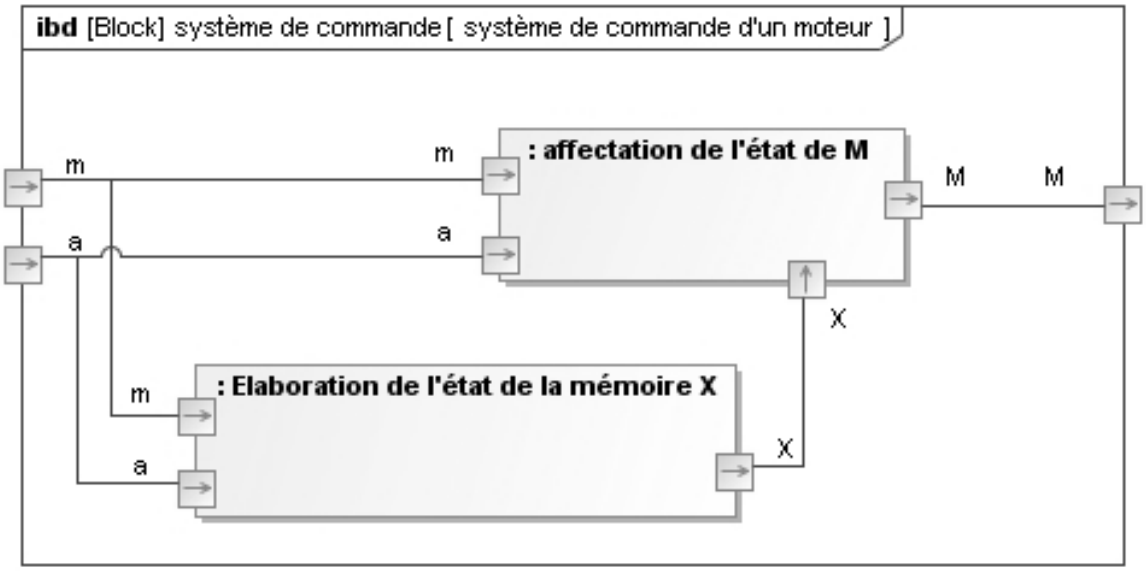
Le cahier des charges est le suivant :

– le moteur doit démarrer si le bouton poussoir marche « m » est actionné,
– une fois ce dernier relâché, le moteur doit continuer à tourner jusqu'à l'appui sur « a »,
– en cas d'appui simultané sur les deux boutons, on privilégie l'arrêt du moteur.



Le système est donc séquentiel

Si on dresse la table de vérité de la variable de sortie "M", deux valeurs de M sont possibles pour un certain état du vecteur des variables d'entrées.
En effet, si m=0 et a=0 :
– le moteur marche si le dernier bouton poussoir appuyé est « m »,
– le moteur est à l'arrêt si le dernier bouton poussoir appuyé est « a ».
Le système de commande doit comporter une variable interne au système « X » ou mémoire, qui lui permettra de faire la distinction entre les deux cas précédents :



La table de vérité du sous-système « affectation de l'état de M » est alors la suivante :

m	a	X	M
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	0

Notons aussi, qu'il est alors possible d'établir la loi de commande suivante : M = (m+ X).$\bar{a}$.

22 : Comment modéliser le comportement d'un système séquentiel ?

221 : Méthode : construction et lecture d'un diagramme d'états

Pour chaque activité, il faut définir un état représenté par un nœud. Au cas où aucune activité n'est souhaitée, c'est un état d'attente.

Il faut ensuite identifier les cas d'évolution du système, c'est-à-dire les transitions possibles d'un état à un autre. On trace alors un lien entre les nœuds correspondants.

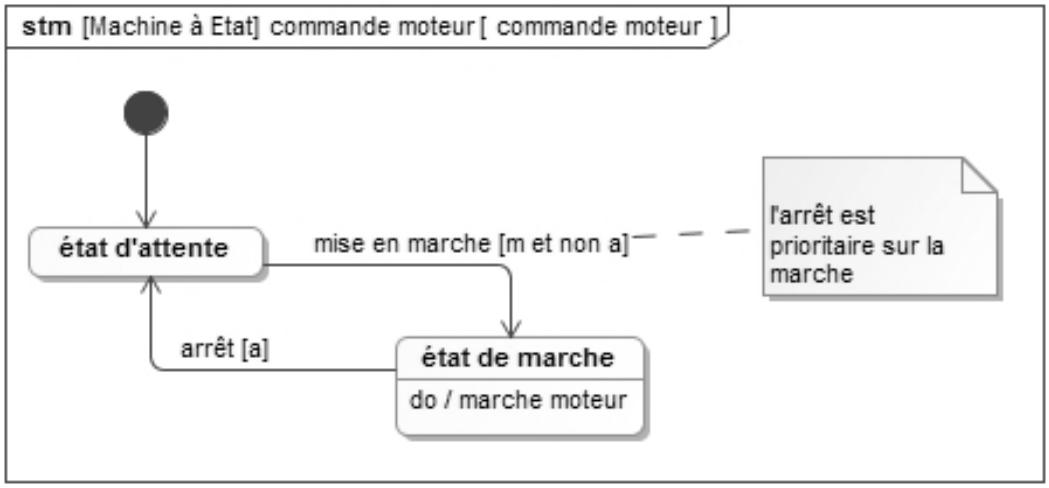
Chaque transition peut être qualifiée par un évènement, une condition de garde et un effet.

Exemple : commande tout ou rien (TOR) d'un moteur (exemple précédent)

Deux états internes du système cohabitent. Dans un cas le moteur est en marche, dans l'autre il est à l'arrêt.

Les conditions de passage d'un état à un autre se font à l'aide des variables d'entrée arrêt « a » et marche « m ».

Le diagramme d'état est alors décrit en page suivante.



L'activité « marche moteur » peut être détaillée dans un diagramme d'activité. Elle comporterait l'action « distribuer l'énergie ». Son exécution ne nécessite pas de temps significatif, elle est associée à un ordre de commande en direction du préactionneur du moteur (un contacteur électrique à arrêt prioritaire).

222 : Méthode : utilisation des pseudo-états

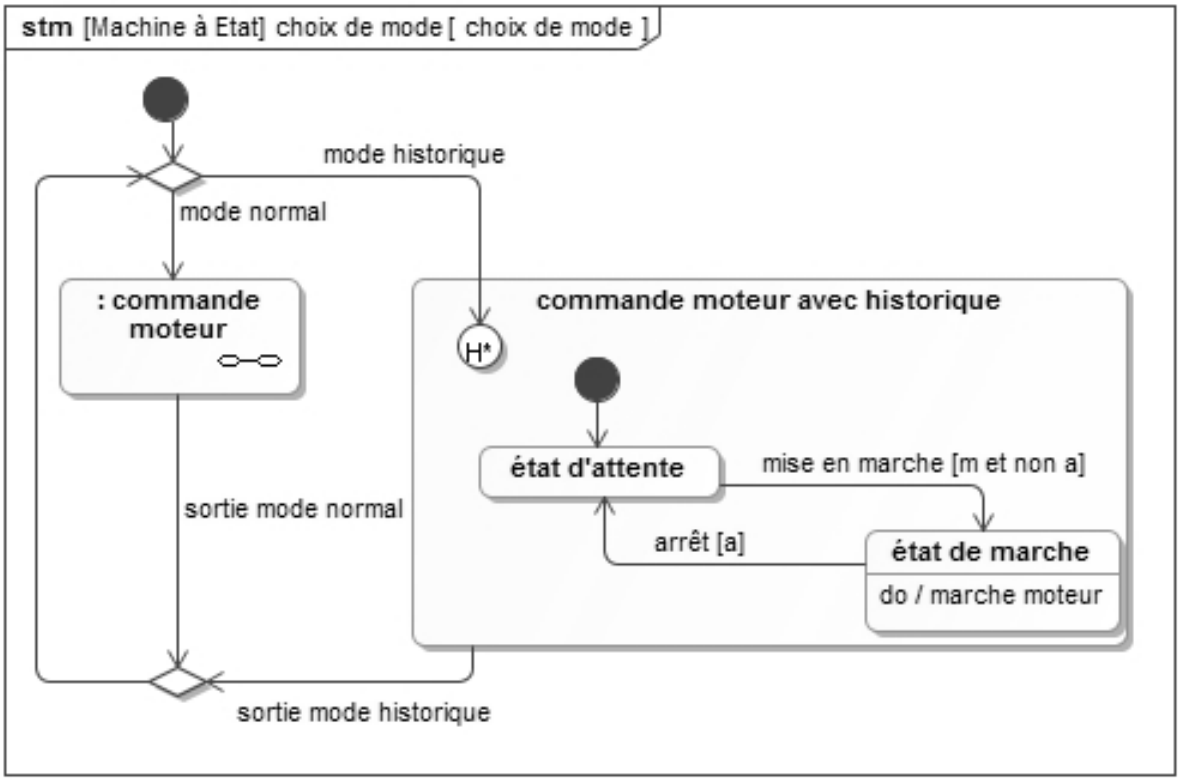
Les pseudo-états permettent d'utiliser des fonctionnalités avancées sans trop compliquer le diagramme d'état avec le formalisme de base.

Il est alors possible de montrer des sélections de séquences d'états (évolutions avec alternatives), des structures parallèles (évolutions simultanées), etc.

Exemple : commande tout ou rien (TOR) d'un moteur (exemple précédent suite…)

Un diagramme d'états « choix de mode » de niveau hiérarchique supérieur à celui décrit précédemment (« commande moteur »), permet un démarrage selon deux modes :

– « mode normal » : moteur à l'arrêt dans l'« état d'attente »,
– « mode historique » : dernier état avant que l'on sorte du diagramme « commande moteur » (« état d'attente » ou « état de marche »).



223 : Méthode : choix d'une structure algorithmique

Les structures alternatives permettent d'envisager des cas entraînant des comportements différents. Le cahier des charges doit être de la forme « si cas 1, alors faire groupe d'instructions 1, sinon faire groupe d'instructions 2 ».

Les structures répétitives permettent des exécutions multiples d'un même bloc d'instructions. Pour un cahier des charges du type « *répéter groupe d'instructions jusqu'à condition vraie* », le groupe est exécuté au moins une fois.

Pour un cahier des charges du type « *tant que condition vraie, faire groupe d'instructions* », le groupe peut ne jamais être exécuté.

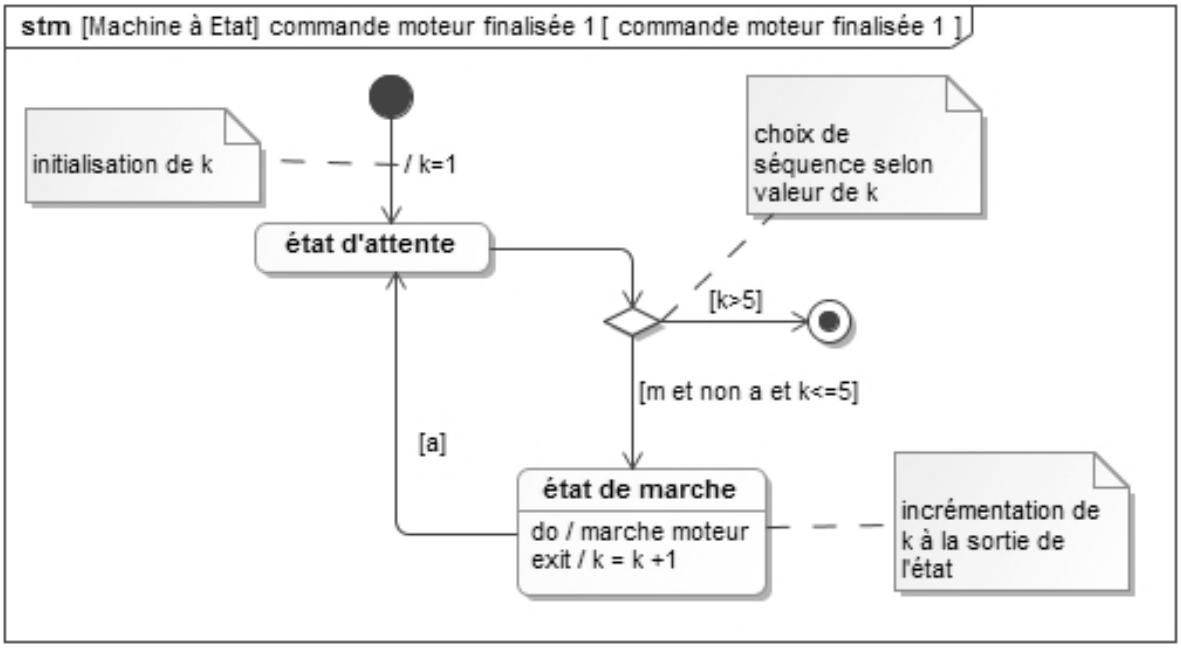
Exemple : commande tout ou rien (TOR) d'un moteur (encore lui !!!!)

Le cahier des charges met clairement en évidence deux modes de fonctionnement.

On peut maintenant essayer de satisfaire à un complément de cahier des charges par rapport à la méthode précédente : après cinq mises en route du moteur, le diagramme d'états conduit à son état final.

Une structure du type « *Pour k = 1, jusqu'à 5, faire …* » semble particulièrement adaptée étant donné que le nombre d'exécutions est connue à l'avance.

On propose la solution suivante (d'autres seraient possibles)



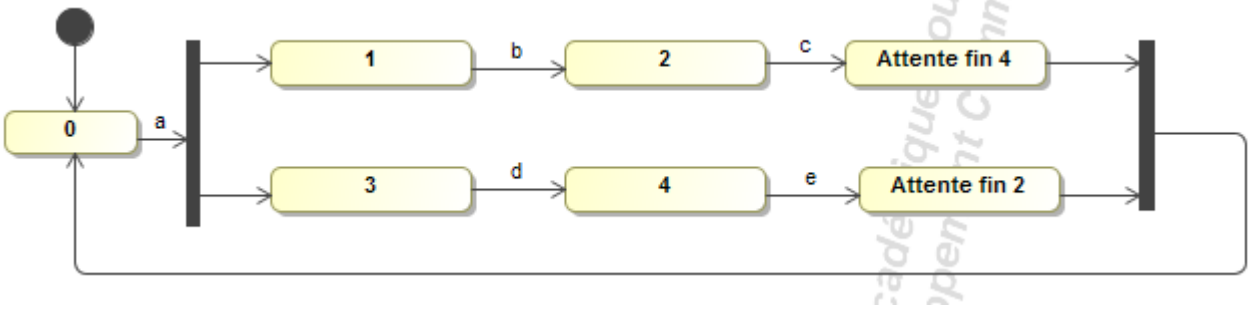
3 : Les structures en parallèle

31 : Exemple n°1

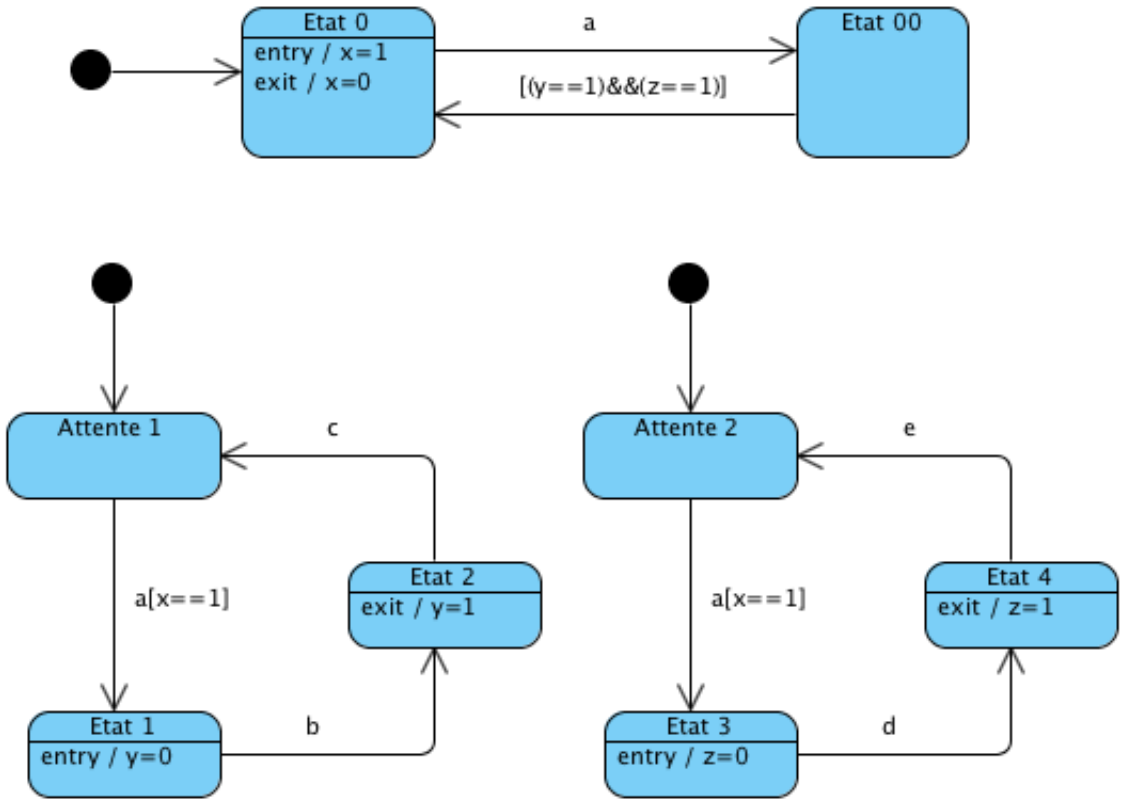
Proposition de trois solutions permettant de décrire le même fonctionnement.

À partir de l'état initial, l'occurrence de l'événement « a » conduit à la réalisation de deux séquences en parallèle. Lorsque ces deux séquences sont terminées, l'état initial est à nouveau activé.

Proposition 1 : utilisation de fork et join

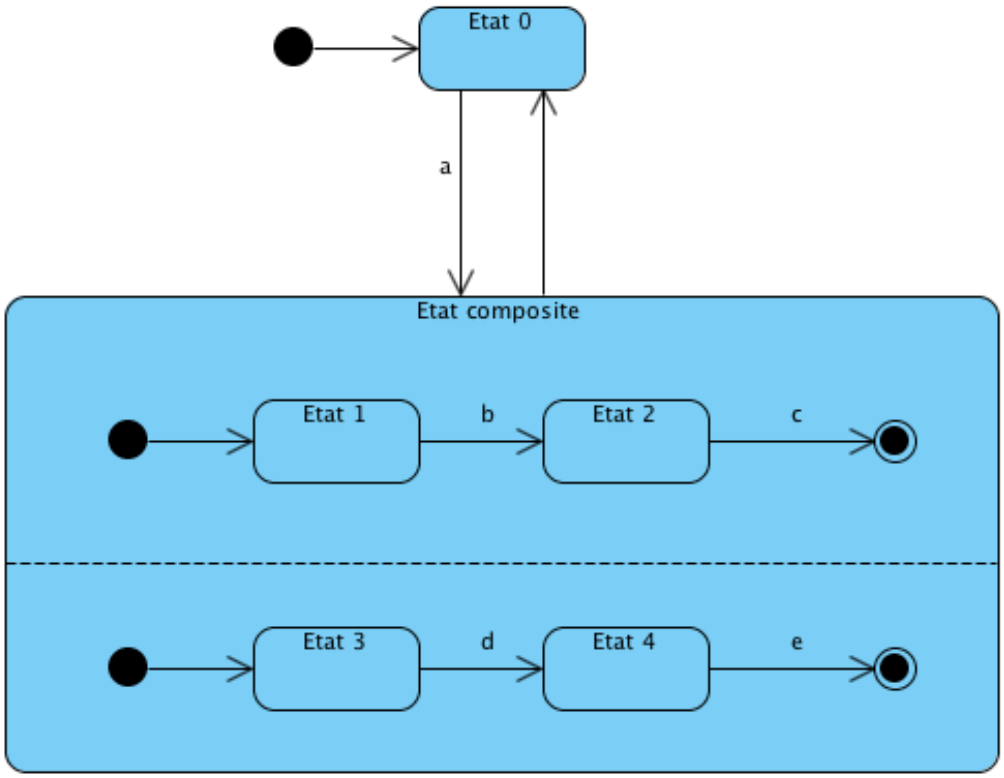


Proposition 2 : solution multi graphe avec création de variables de synchronisation



Proposition 3 : utilisation d'un état composite avec deux régions orthogonales.

Cette solution est sans doute plus dans l'esprit du diagramme d'état ?

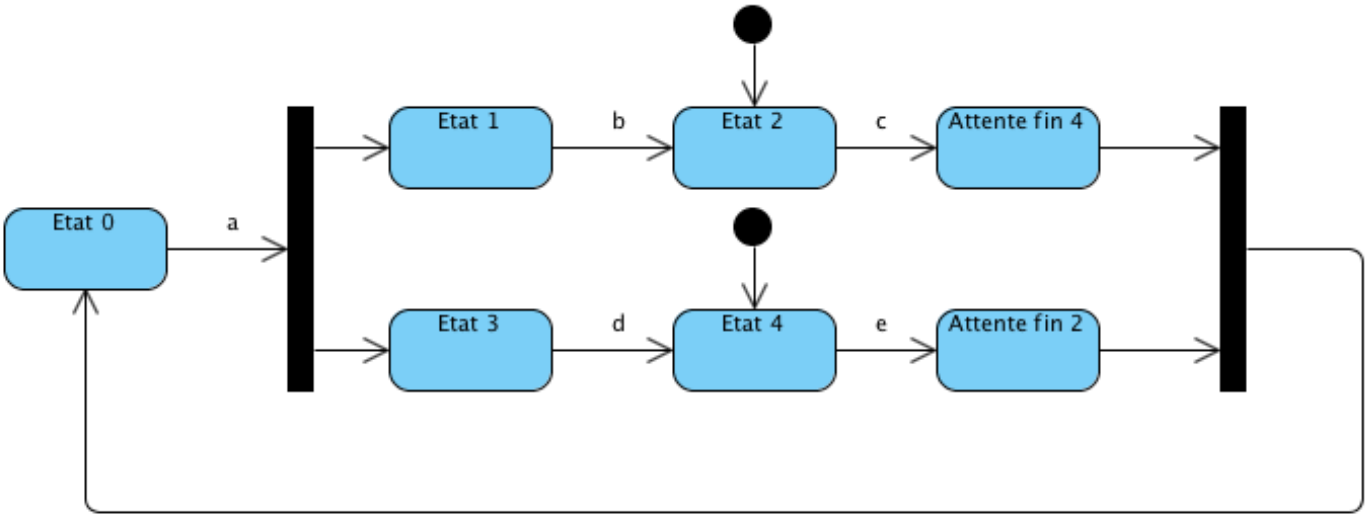


31 : Exemple n°2

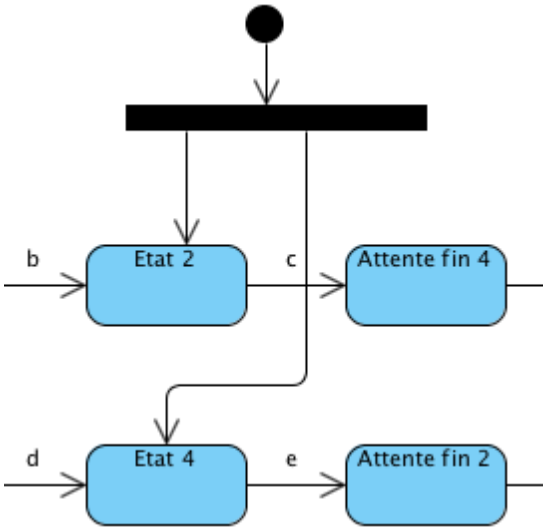
Cette fois, le fonctionnement est différent, puisque dès l'état initial les états 2 et 4 sont activés.

Les diagrammes précédents sont alors modifiés comme suit.

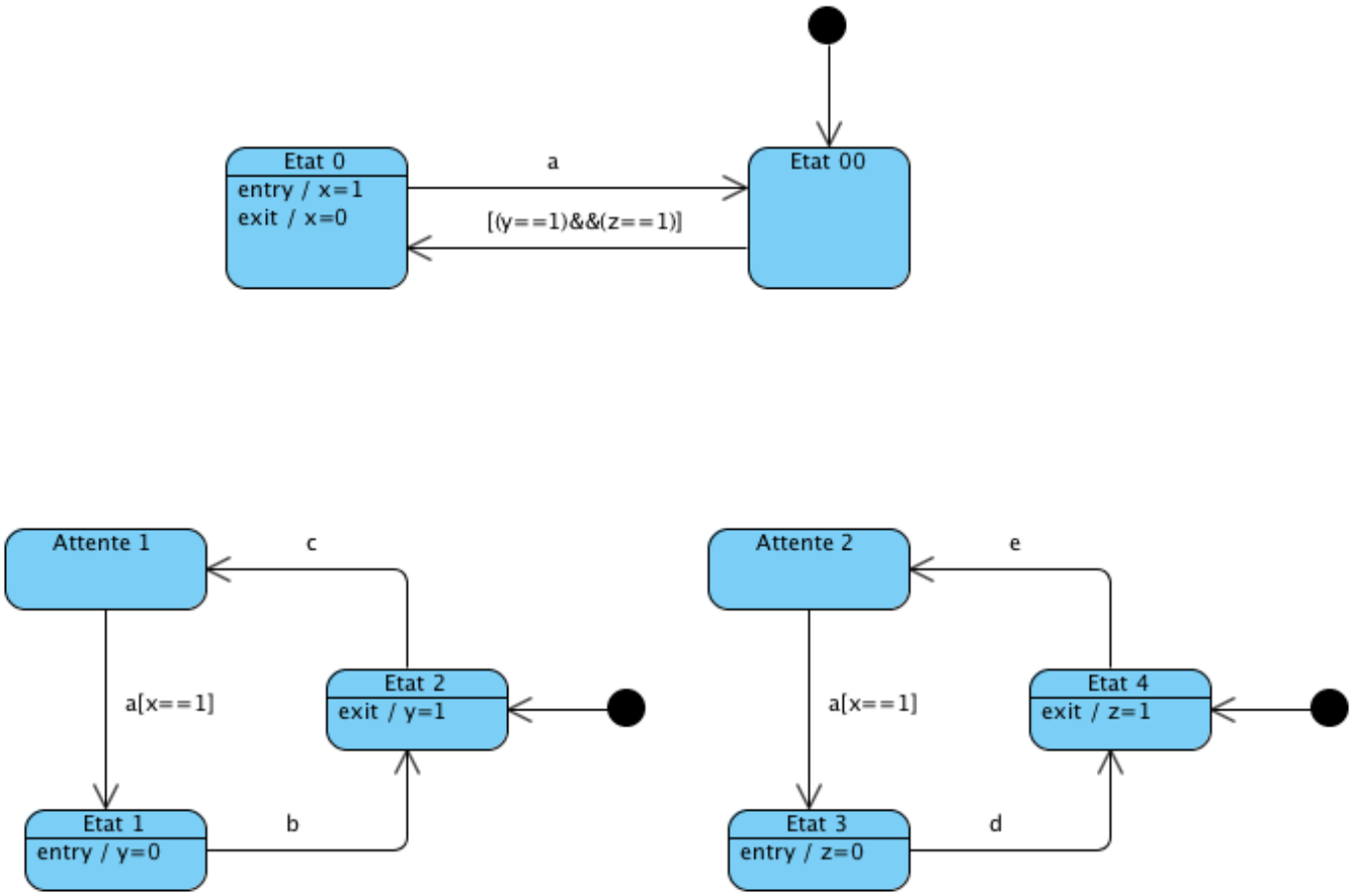
Proposition 1 : utilisation de fork et join



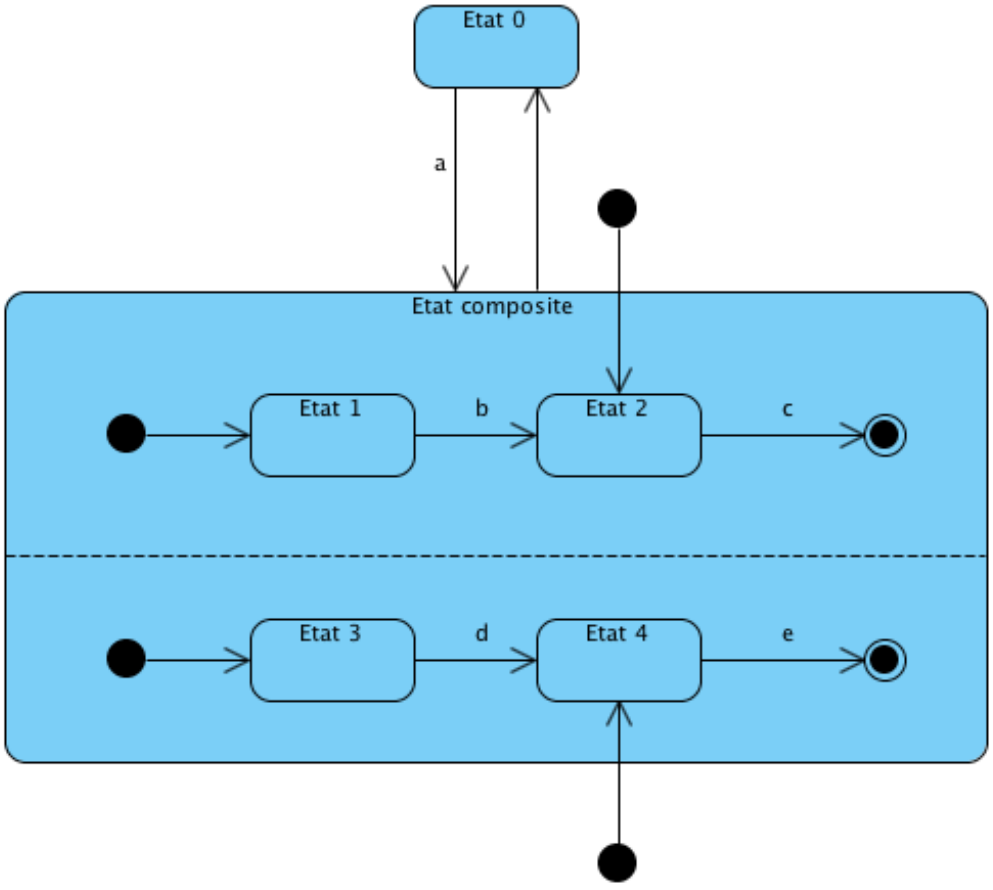
Question : a-t-on le droit d'avoir plusieurs états initiaux, où doit-on en mettre un seul, distribué par un fork ?



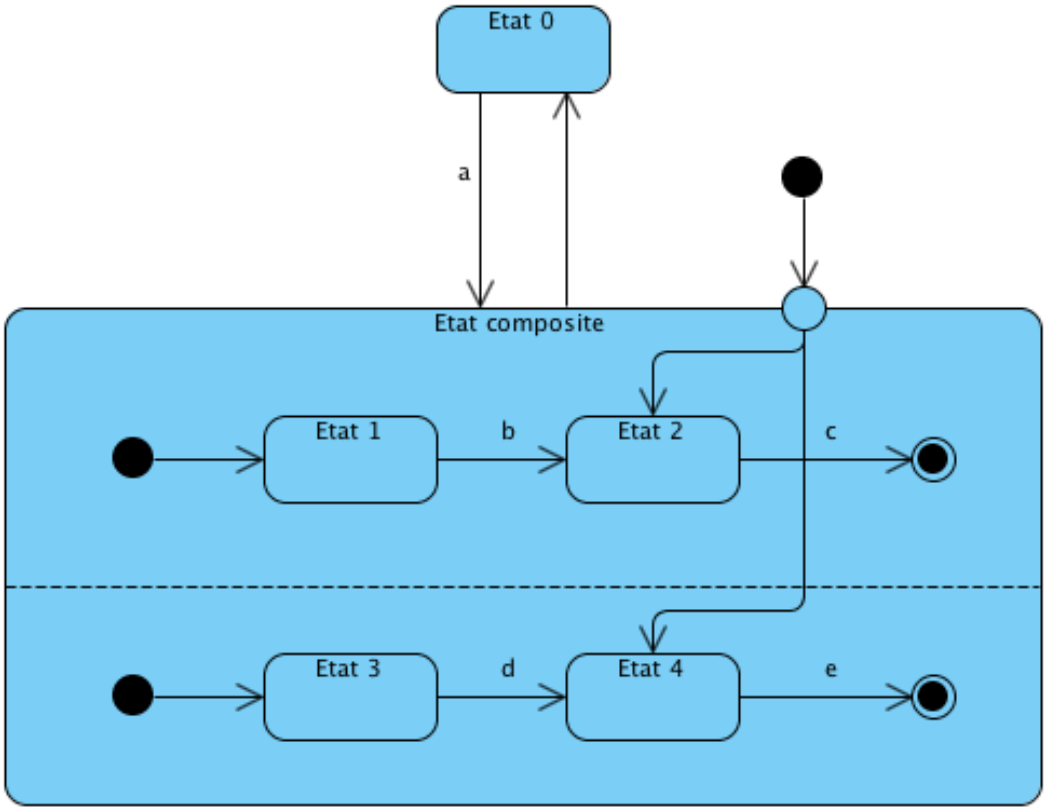
Proposition 2 : solution multi graphe avec création de variables de synchronisation.



Proposition 3 : utilisation d'un état composite avec deux régions orthogonales.



Question : a-t-on le droit d'avoir des transitions qui pointent vers des sous-états d'un état composite ? (même interrogation d'ailleurs en ce qui concerne des transitions dont les sources seraient des sous-états)
Doit-on utiliser obligatoirement utiliser des points de connexion ? La solution est-elle alors la suivante ?



Ou bien les points de connexion ne sont obligatoires que lorsque l'état composite n'est pas détaillé, comme ci-dessous ?