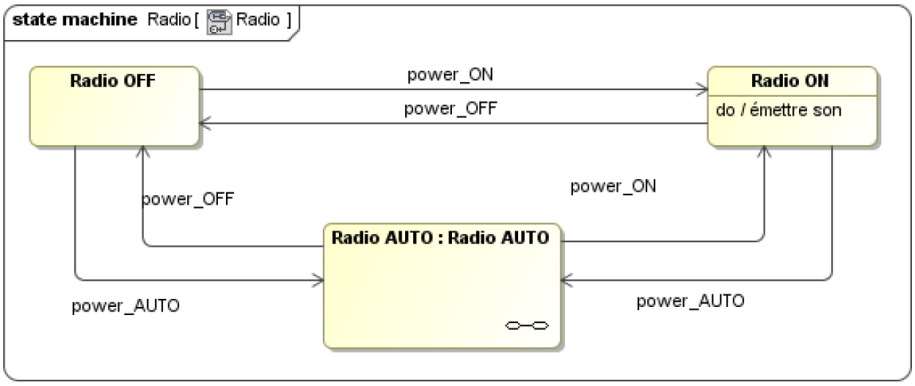
CI-6 Modéliser, prévoir et vérifier les performances des systèmes combinatoires et séquentiels.

CI-6-2 Prévoir, simuler et valider un système séquentiel



Objectifs — MODELISER-SIMULER-VALIDER-OPTIMISER

A l'issue de la séquence, l'élève doit être capable de:

- **B2 :** Proposer un modèle de connaissance et de comportement
  - Représenter tout ou partie de l'évolution temporelle
  - Décrire et compléter un algorithme représenté sous forme graphique
- **F2 :** Mettre en oeuvre une communication
  - Choisir l'outil de description adapté à l'objectif de la communication
  - Décrire le fonctionnement du système en utilisant un vocabulaire adéquat

Table des matières

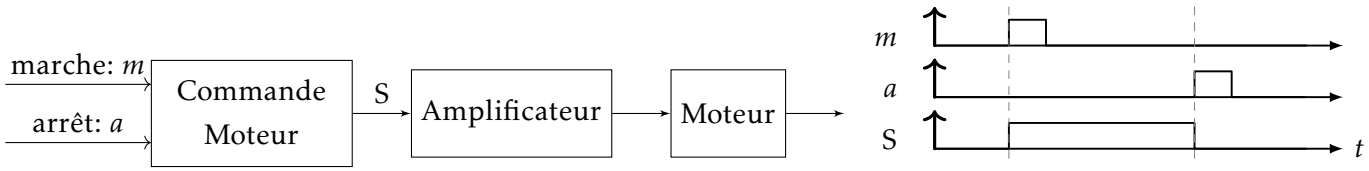
1 Système séquentiel

1.1 Définition

Définition : Système séquentiel

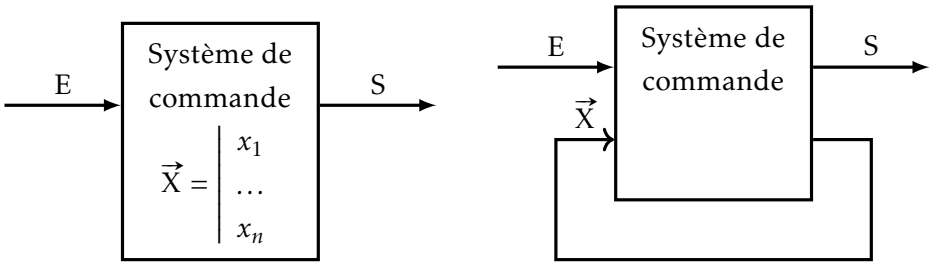
> *Système où l'état des sorties S_i dépend de l'état des entrées à l'instant présent, mais aussi de l'histoire de l'évolution des entrées-sorties.*

Exemple : Commande d'un moteur électrique par un système séquentiel. On remarque sur le chronogramme de droite que la sortie S peut présenter une valeur différente (0 ou 1) pour une configuration identique des entrées m et a.



Commande d'un moteur électrique

Le système est capable de mémoriser de l'information. Cette information mémorisée est l'état du système, qui peut être représenté par un vecteur d'état $\vec{X} = (x_1, x_2, \ldots, x_n)$.



Représentation d'un système séquentiel par un système combinatoire muni d'un vecteur d'état

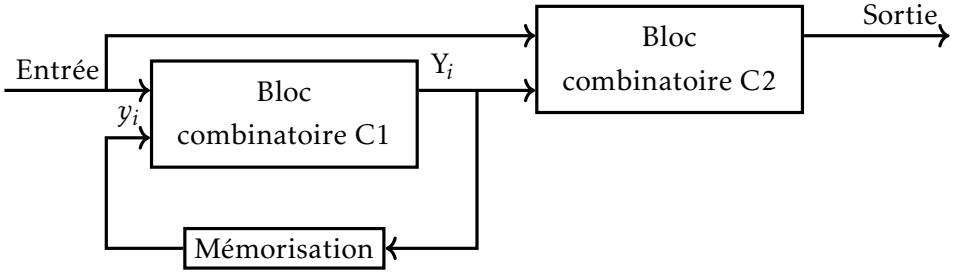
Connaissant $\vec{X}$ et $\vec{E}$, la sortie $\vec{S}$ peut être déterminée comme une fonction booléenne de $\vec{X}$ et de $\vec{E}$: $\vec{S} = f(\vec{X}, \vec{E})$. L'état interne $\vec{X}$ à l'instant t dépend de $\vec{E}(t)$ et de l'état interne immédiatement précédent : $\vec{X} = f\left(\vec{E}(t).\vec{X}(t-\Delta t)\right)$.

1.2 Structure d'un système séquentiel

La structure d'un système séquentiel fait apparaître deux blocs fonctionnels combinatoires.

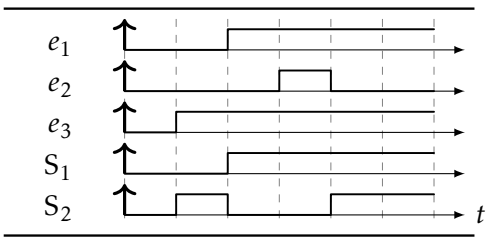
Un système séquentiel évolue à partir d'entrées logiques et à partir de son état caractérisé par un certain nombre de variables internes. Les variables internes évoluent à partir des entrées et de leurs propres valeurs mémorisées.

La mémorisation des variables internes Y_i (Y_{n+1}) est réalisée au moyen d'éléments spécifiques ou dans la construction même du système. Les temps de propagation des signaux dans le bloc C1 permettent de définir les valeurs actuelles des variables internes y_i (Y_n) et les futures valeurs Y_i (Y_{n+1}).



1.3 Chronogrammes ou diagrammes de Gantt

Afin de caractériser ces systèmes, on utilise alors les chronogrammes (aussi appelés diagrammes de Gantt). On représente l'évolution chronologique des entrées et sorties en considérant des changements d'états instantanés et simultanés.



Exemple de chronogramme:

1.4 Diagramme de séquence (sd)

1.4.1 Définitions

Le diagramme de séquence est un diagramme comportemental appelé Sequence Diagram (sd) dans le langage SysML.

Définition : Diagramme de séquence

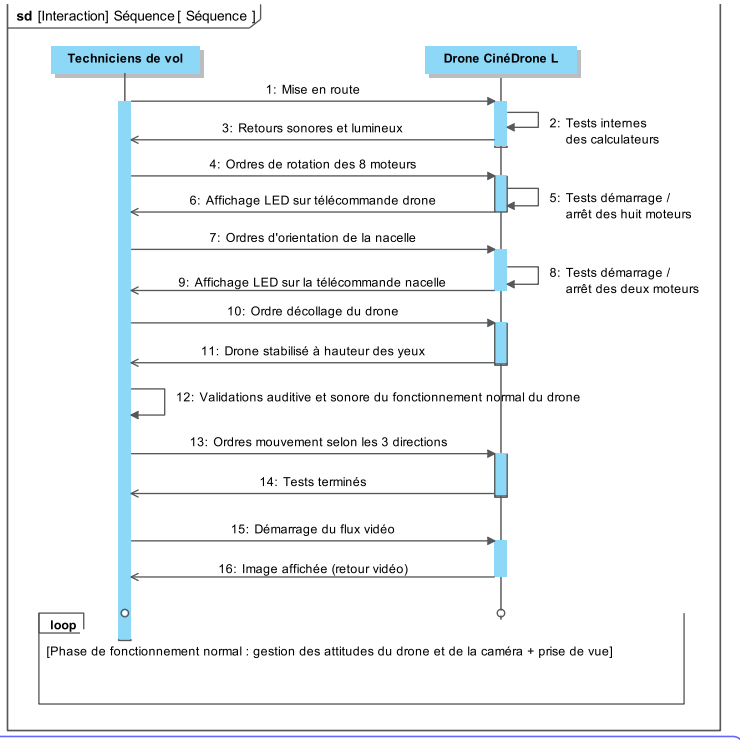
> *diagramme permettant de décrire les interactions existant entre plusieurs entités, celles-ci pouvant être des acteurs, le système ou ses sous-systèmes. Le diagramme ne montre donc que l'enchaînement séquentiel des différentes interactions.*

Un diagramme de séquence est rattaché à un cas d'utilisation et décrit ce dernier en entier ou en partie, ce qui correspond à un scénario de fonctionnement possible, défini dans un cadre précis : il peut donc aboutir tout aussi bien à des évolutions positives (fonctionnement normal) ou négatives (gestion des problèmes), en particulier dans la phase de démarrage avant le fonctionnement normal.

Le diagramme de séquence montre la séquence verticale des messages passés entre éléments (lignes de vie) au sein d'une interaction.

Définition : Ligne de vie

> *Représentation de l'existence d'un élément participant dans un diagramme de séquence. Une ligne de vie possède un nom et un type. Elle est représentée graphiquement par une ligne verticale en pointillés.*



1.4.2 Aspects graphiques

Pour les messages propres à un cas d'utilisation, les diagrammes de séquence "système" montrent non seulement les acteurs externes qui interagissent directement avec le système, mais également ce système (en tant que boîte noire) et les événements système déclenchés par les acteurs.

L'ordre chronologique se déroule vers le bas et l'ordre des messages doit suivre la séquence décrite dans le cas d'utilisation.

On représente un diagramme de séquence "système en plaçant de façon privilégiée l'acteur principal à gauche, puis une ligne de vie unique représentant le système en boîte noire, et, enfin, les éventuels acteurs secondaires sollicités durant le scénario à droite du système.

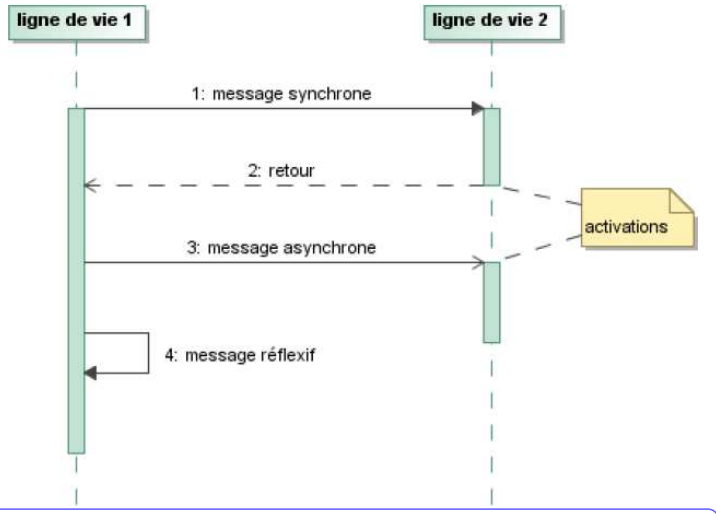
Définition : Message

> *Appel d'un comportement chez le destinataire. Cela peut être des signaux ou des opérations.*

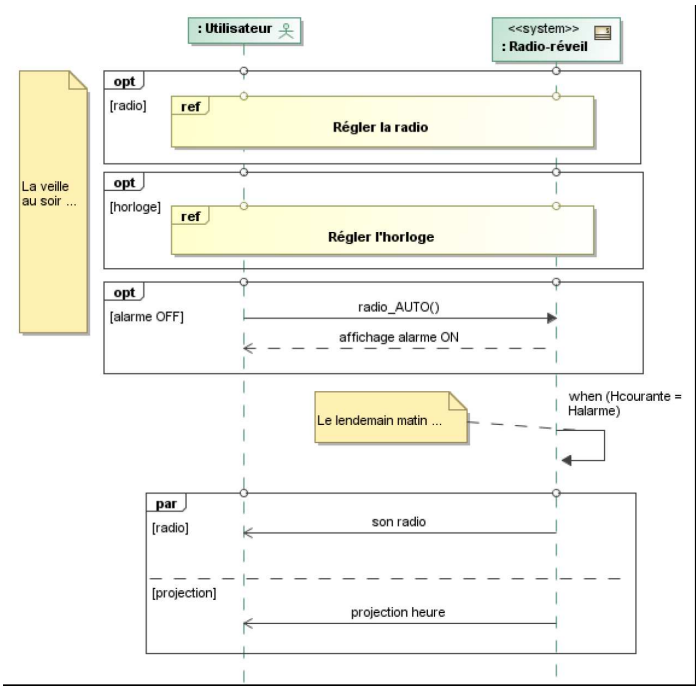
On distingue trois catégories de messages:

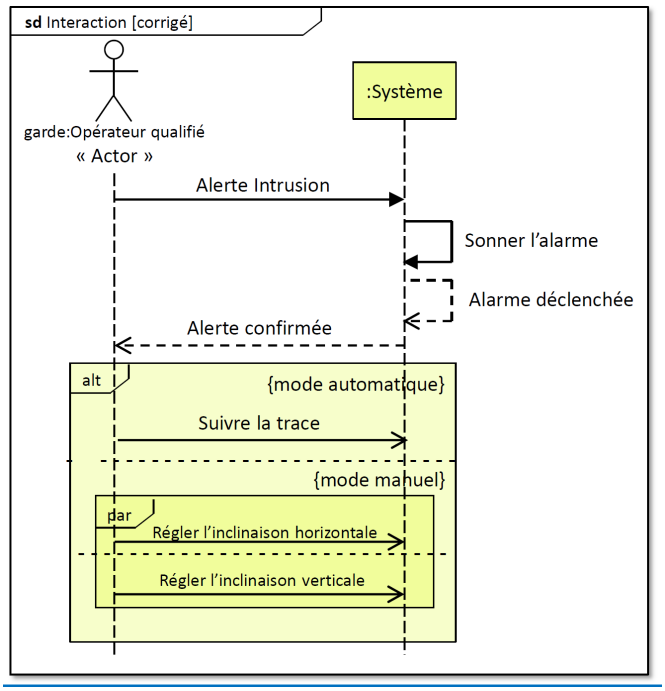
- **synchrones :** l'expéditeur attend une réponse pour poursuivre ——▶
- **asynchrones :** l'expéditeur n'attend rien en retour ——→
- **réponses :** - - - - - - - - - -›

Remarque : il exite aussi des messages réflexifs ⟲



1.4.3 Fragments combinés





SysML propose une notation très utile : le fragment combiné. Chaque fragment possède un opérateur et peut être divisé en opérandes. Les principaux opérateurs sont :

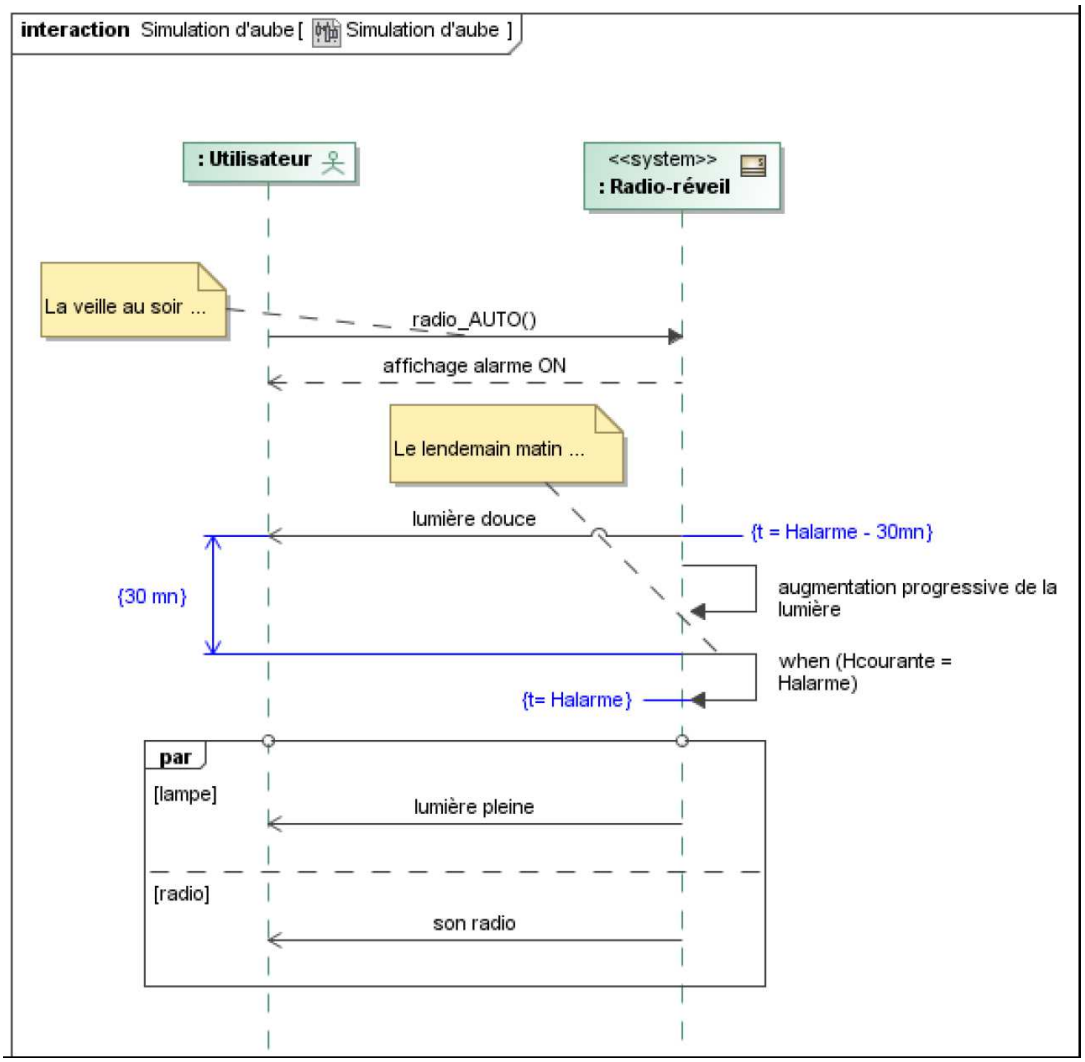
- **loop :** boucle. Le fragment peut s'exécuter plusieurs fois, et la condition de garde explicite l'itération ;
- **opt :** optionnel. Le fragment ne s'exécute que si la condition fournie est vraie ;

- **alt :** fragments alternatifs. Seul le fragment possédant la condition vraie s'exécutera.

1.4.4 Contraintes temporelles

SysML permet d'ajouter des contraintes temporelles sur le diagramme de séquence. Il existe deux types de contraintes :

- **la contrainte de durée** permet d'indiquer une contrainte sur la durée exacte, la durée minimale ou la durée maximale entre deux événements ;
- **la contrainte de temps** permet de positionner des labels associés à des instants dans le scénario au niveau de certains messages et de les relier ainsi entre eux.



Pour simuler l'aube, la lampe commence à émettre doucement trente minutes avant l'heure du réveil. Le message 3 : lumière douce modélise ce début d'éclairage.

La contrainte de durée est représentée par une double flèche en prolongement de ce message et du déclenchement de l'alarme (message 5), avec la durée entre accolades : {30 mn}.

La contrainte de temps, pour sa part, est représentée en associant une contrainte {t = Halarme} en prolongement du message 5, et une autre contrainte {t = Halarme - 30 mn} en prolongement du message 3.

2 Diagrammes d'états et d'activités

2.1 Introduction

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.

Définition : Etat

> *Représentation d'une situation d'une durée finie durant laquelle un système exécute une activité, satisfait à une certaine condition ou bien est en attente d'un événement.*

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'événements
- 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'événements.

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

2.2 Le diagramme d'états

2.2.1 Définition

Le diagramme d'états est un diagramme comportemental appelé StateMachine Diagram (stm) dans le langage SysML.

Définition : Diagramme d'états (stm)

> *Diagramme dont la description du comportement sert à montrer les différents états pris par le bloc en fonction des évènements qui lui arrivent.*

Le diagramme d'états est rattaché à un bloc (ou plutôt à son instance) qui peut être le système, un sous-système ou un composant.

2.2.2 Etat

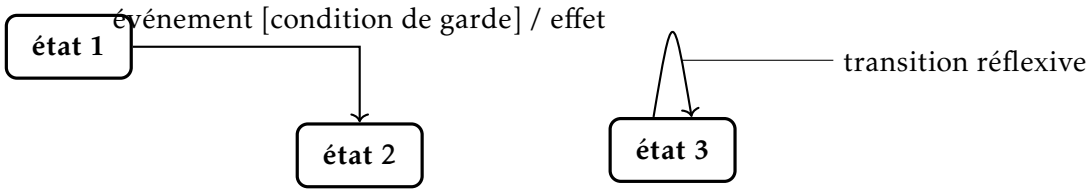
Les éléments graphiques utilisés dans ce diagramme sont principalement des rectangles aux coins arrondis pour représenter les états.

L'état initial ● correspond à la création de l'instance du bloc pour lequel le diagramme d'état est spécifié.

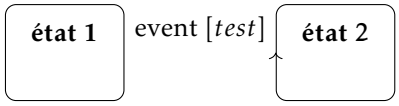
L'état final ◉ correspond à la destruction de cette instance de bloc. Il peut y en avoir plusieurs dans un diagramme d'états. En effet, plusieurs scénarios peuvent être possibles pour mettre fin à un comportement.

2.2.3 Transition

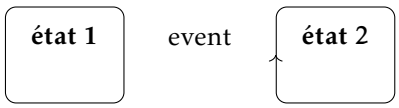
Une transition peut être associée à un événement, à une condition de garde et / ou à un effet (une action).



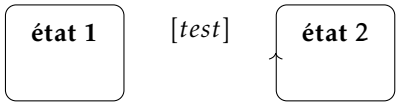
Le passage d'un état à un autre se fait en franchissant une transition :

- 

A l'occurrence de event, test est évalué et la transition est franchie uniquement si *test* est vrai. L'éventuelle activité est interrompue. Si *test* n'est pas vrai, event est *perdu* et il faut attendre une seconde occurrence de event pour éventuellement franchir la transition si cette fois *test* est vrai.

- 

A l'occurrence de event, la transition est franchie sans condition. L'éventuelle activité est interrompue.

- 

Si *test* est vrai, la transition est franchie uniquement dès la fin de l'éventuelle activité (qui doit donc être une activité finie). S'il n'y a pas d'activité associée à l'état 1, la transition est franchie immédiatement si *test* est vrai.

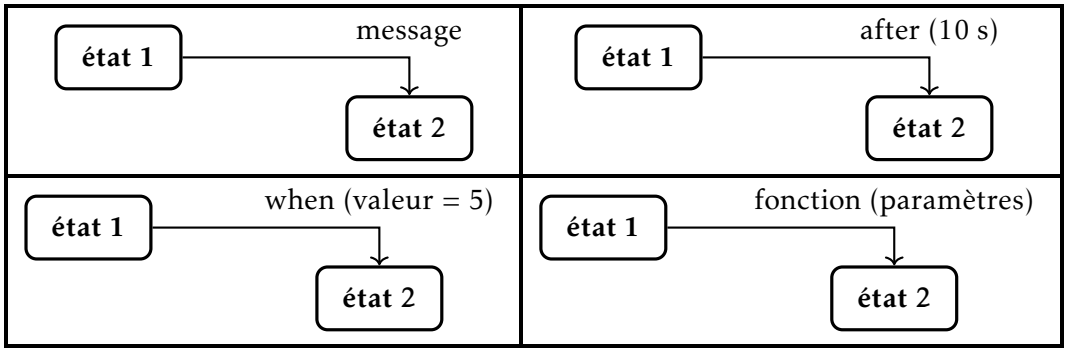
- 

Transition de complétion : est immédiatement franchie dès la fin de l'éventuelle activité. Equivaut à [1].

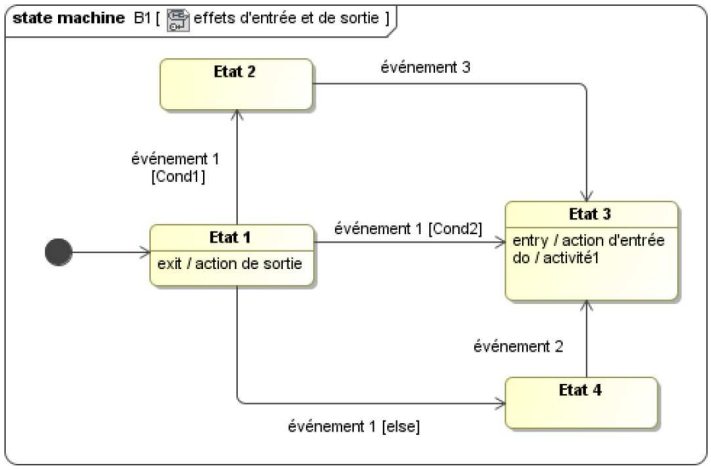
Une transition réflexive entraîne une sortie d'état puis un retour dans ce même état. Cela n'est donc pas sans conséquences selon les cas.

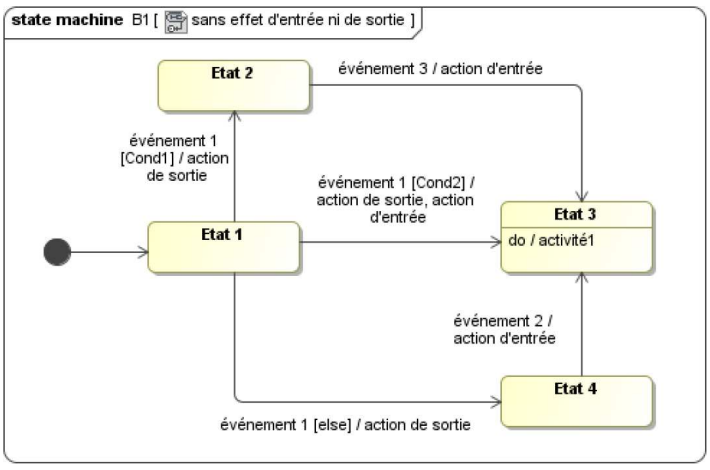
Il existe quatre types d'événements associés à une transition :

- **le message (signal event) :** un message asynchrone est arrivé,
- **l'événement 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'événement de changement (change event) :** une valeur a changé de telle sorte que la transition est franchie (mot clé when),
- **l'événement d'appel (call event) :** une requête de fonction (operation) du bloc a été effectuée. Un retour est attendu. Des arguments (paramètres) de fonction peuvent être nécessaires.



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.





2.2.4 Activité et action

A un état, on peut ainsi principalement rattacher une activité, une action d'entrée et une action de sortie.

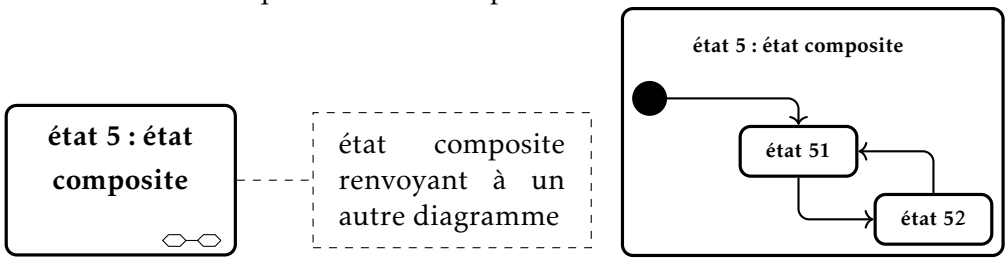
état
entry / action d'entrée
do / activité
exit / action de sortie

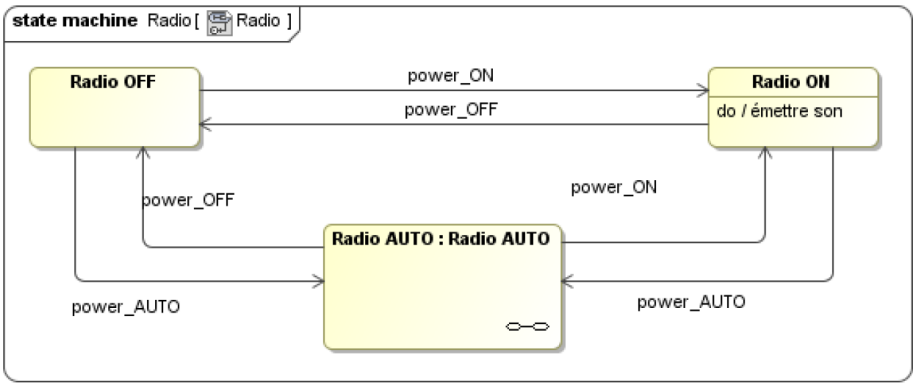
Une activité peut être considérée comme une unité de comportement. Elle prend du temps et peut être interrompue. On la trouve à l'intérieur des nœuds du diagramme (mot clé do).

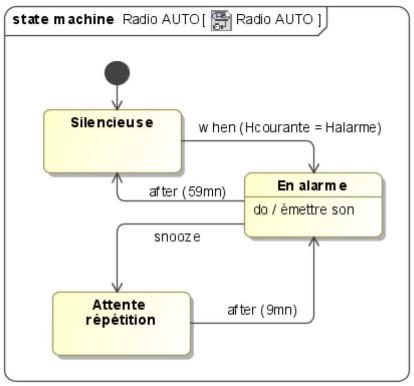
A contrario, une action ne prend pas de temps et ne peut pas être interrompue. Son exécution peut par exemple provoquer un changement d'état, l'émission d'un ordre pour un préactionneur ou un retour de valeur. On peut les trouver dans les transitions (effet) ou dans les états (mots clé entry ou exit).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

2.2.5 Etat composite (super-état)

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.



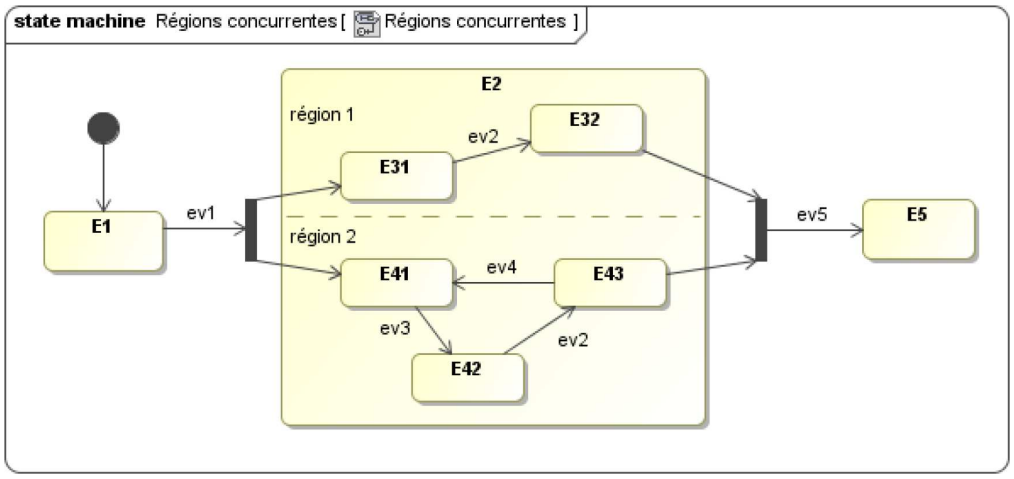




2.2.6 Régions orthogonales

Un état composite peut également contenir des régions concurrentes (ou orthogonales), il suffit graphiquement de le séparer par des traits pointillés.

Chaque région peut alors être nommée (optionnel). Elle contient ses propres états et ses propres transitions. Les régions sont dites concurrentes car elles peuvent évoluer en parallèle et indépendamment. L'état courant de l'élément concerné devient alors un vecteur à plusieurs lignes (autant que de régions).



Dans l'exemple précédent, à partir de l'état E1, quand l'événement ev1 arrive, l'élément passe dans l'état composite E2. Cela signifie qu'il est à la fois dans les états disjoints E31 et E41. Ensuite, suivant l'ordre d'arrivée des événements ev2, ev3 ou ev4, chaque région va évoluer indépendamment. Pour passer à l'état E5, il faudra que l'élément soit à la fois dans E32 et E43 quand ev5 arrivera

.

2.2.7 Les pseudo-états

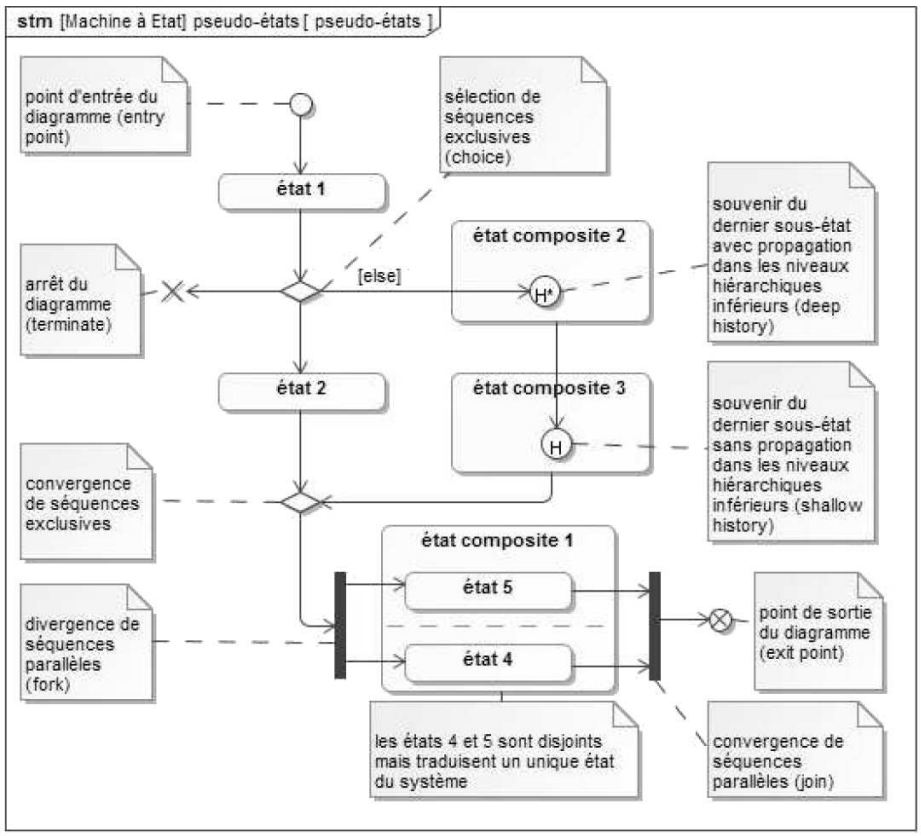
Définition : Pseudo-état

Elément 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 et merge** ◇ **:** sélection (choice) et convergence (merge) 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.



2.3 Le diagramme d'activités

2.3.1 Présentation

Le diagramme d'activités est un diagramme comportemental appelé Activity Diagram (act) dans le langage SysML.

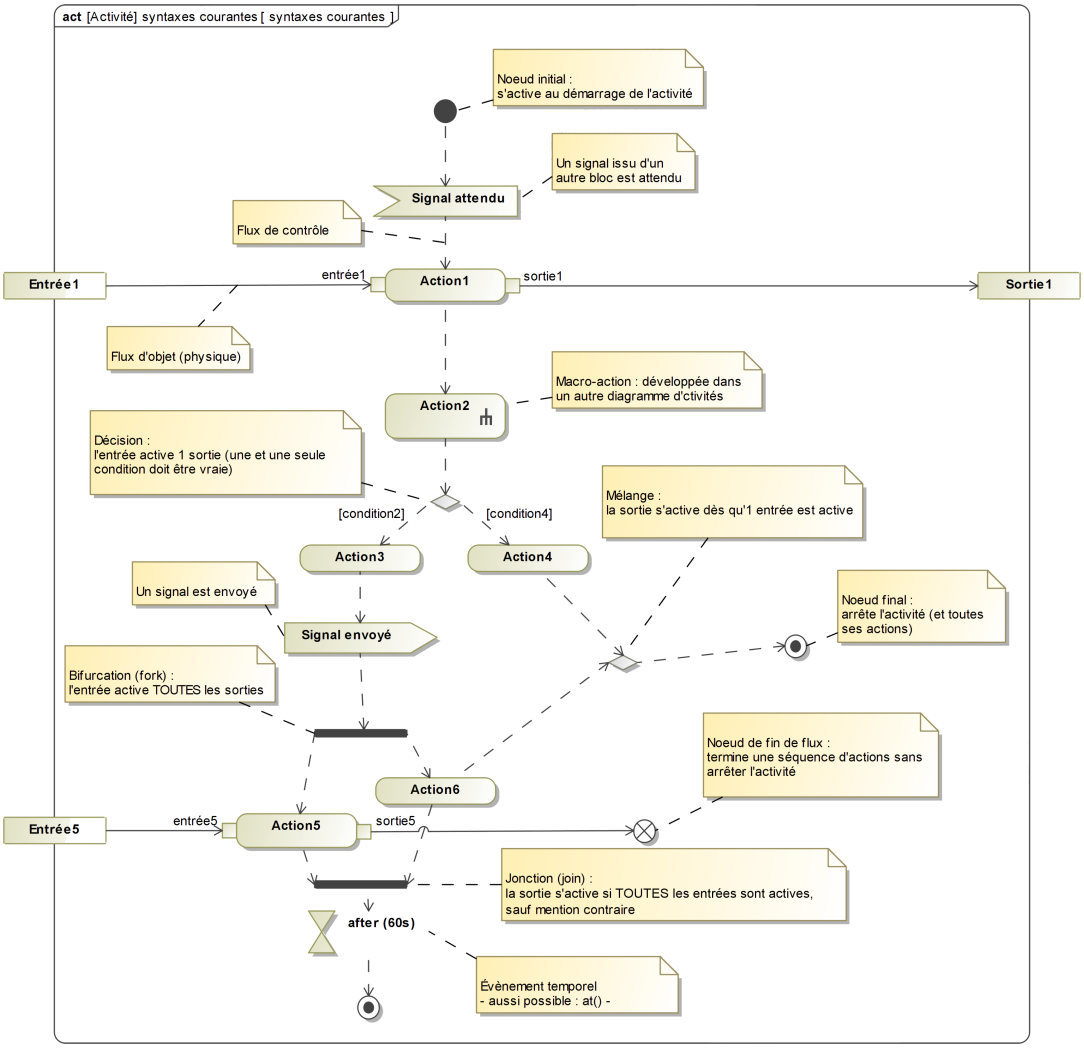
Définition : Diagramme d'activités (act)

> *Diagramme permettant de représenter le déroulement d'un processus sous la forme d'une activité correspondant à une décomposition séquentielle d'actions, aussi appelées tâches.*

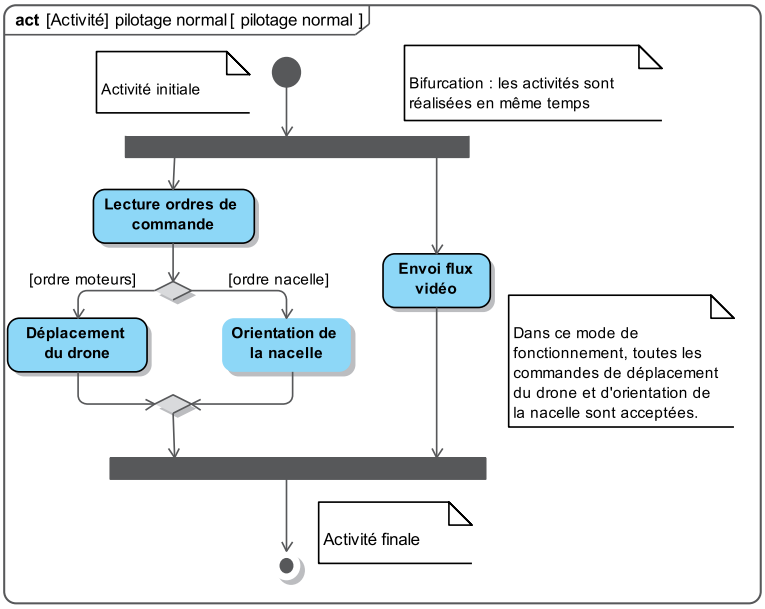
Il permet de décrire la transformation des flux d'entrées en flux de sorties (matières, énergies, informations) par le biais de séquences d'actions ou activité déclenchées par des flux de contrôle. Lorsqu'une tâche est terminée, la suivante commence.

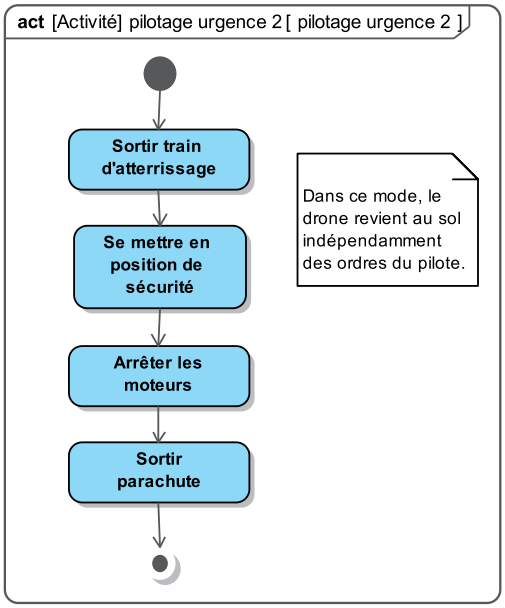
Dans sa forme la plus restreinte, ce diagramme représente un algorigramme, c'est-à-dire un flux de contrôle.

Remarque : ce flux n'a rien à voir avec ceux présents dans le diagramme de blocs internes : il ne faut donc pas les confondre...



2.3.2 Exemples





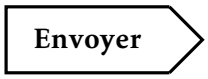
2.3.3 Signaux et événements

En plus de consommer et de produire des paramètres, une activité peut recevoir et émettre des signaux.

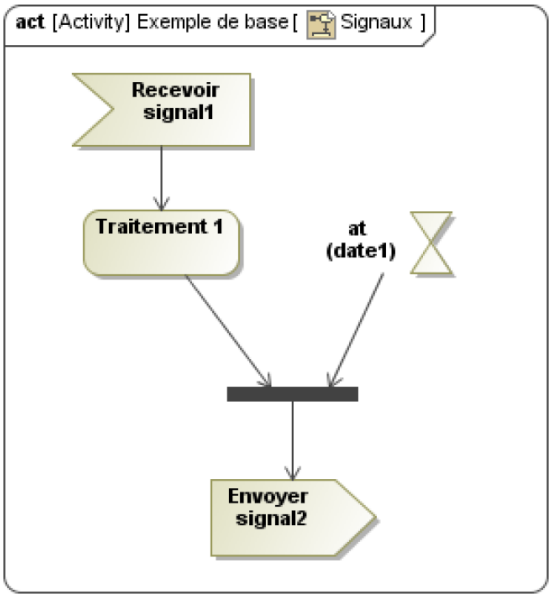
L'idée forte est de permettre à des activités de communiquer en incluant dans une activité l'émission d'un signal et dans une autre la réception d'événements.

Il faut utiliser pour cela des types d'action particuliers, possédant chacun une représentation graphique spécifique :

- **accept event action :** 

- **send signal action :** 

- **accept time event :** 



Remarque : Ce diagramme étant hors programme, un niveau de description supplémentaire ne sera pas envisagé avant la spé.

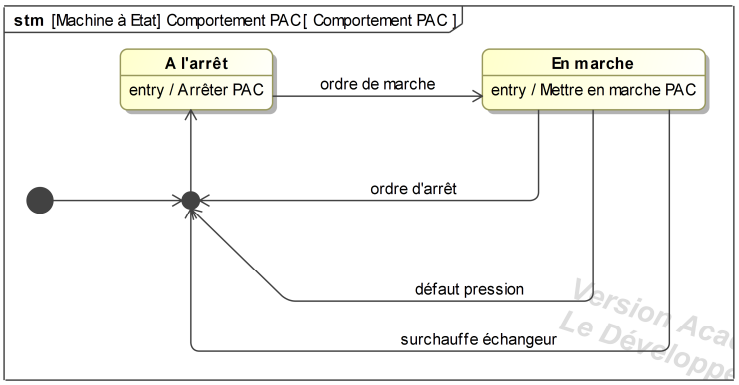
2.4 Synthèse sur le positionnement relatif de ces deux diagrammes

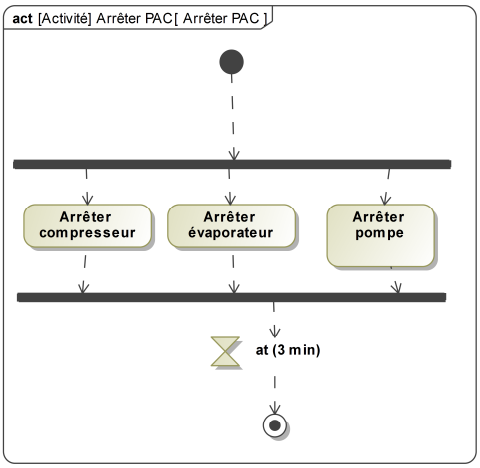
Les deux types de diagrammes sont différents car :

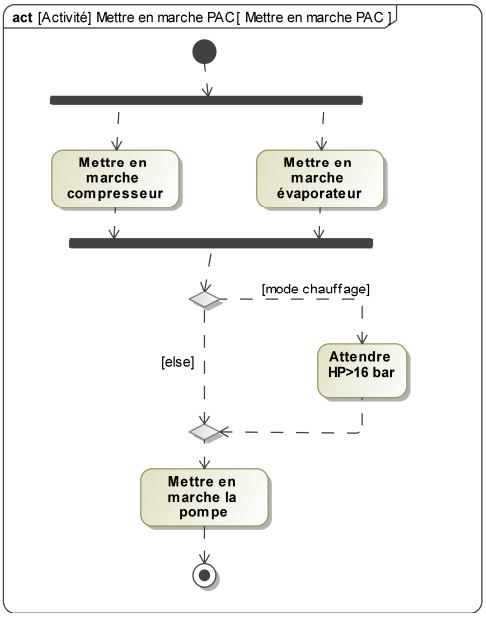
- Le diagramme d'états montre les événements déclenchant le passage d'un mode à un autre et il y aura quasiment toujours un événement associé à une transition.
- Le diagramme d'activités ne possède aucun événement associé aux transitions entre actions : la fin d'une action implique automatiquement le passage à la suivante, donc dans un ordre déterminé d'actions menant à un résultat. Lorsque le processus est enclenché, il va à son terme selon un ordre précis.

En conclusion :

- Le diagramme d'états ne se rattache qu'à un bloc, alors que le diagramme d'activités peut être supporté par plusieurs blocs.
- Un pilotage par des événements se traduit par un diagramme d'états : il ne doit donc pas devenir un diagramme d'activités.
- Dans un processus décrit par un diagramme d'activités, il est possible de mettre en évidence l'élément associé à la tâche. Avec le diagramme d'états, la question ne se pose pas car il est associé à un seul bloc.



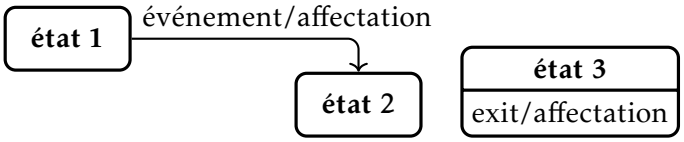




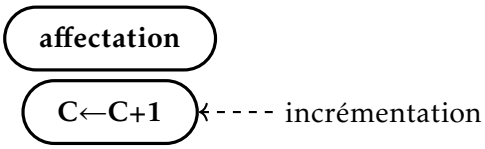
3 Les structures algorithmiques de base

3.1 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.

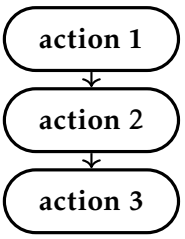


formalisme du diagramme d'états



formalisme du diagramme d'activité

3.2 Le groupe ou bloc d'instructions



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

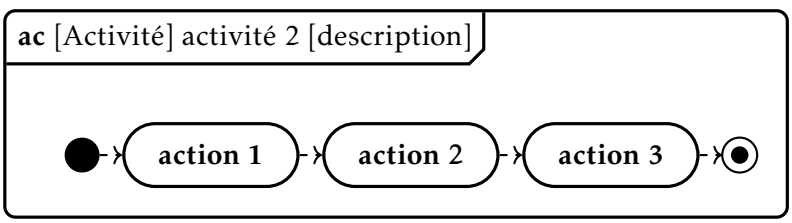
Cela correspond à une succession d'actions et / ou d'activités.

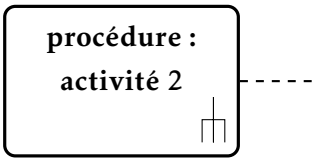
3.3 Fonctions et procédures

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.

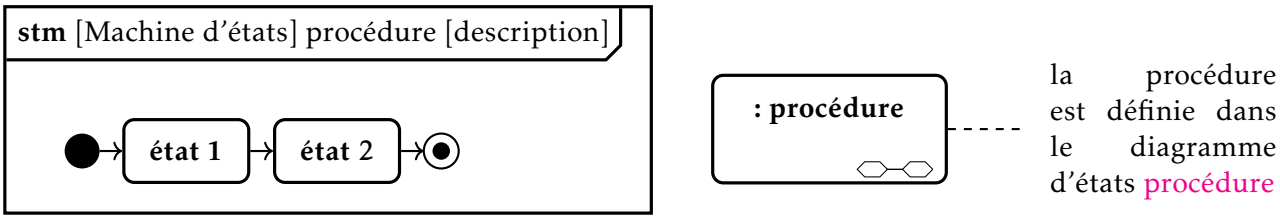
Une procédure comporte une succession d'instructions mais ne renvoie rien.



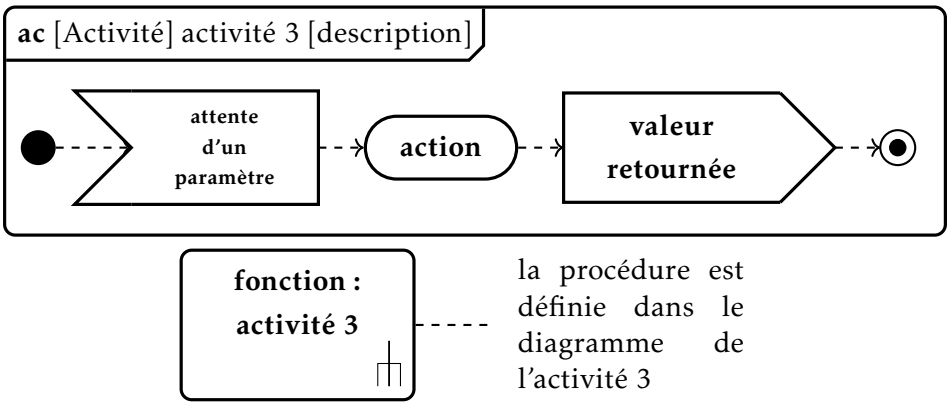


la procédure est définie dans le diagramme de l'activité 2

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



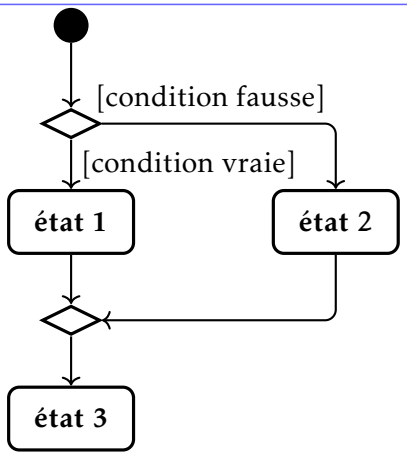
A la fin de l'exécution d'une fonction, il y a le retour d'une valeur, d'une liste, d'un objet, etc.



3.4 La structure alternative (conditionnelle)

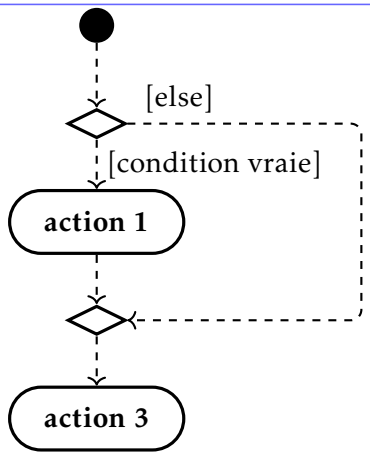
Si . . . , alors faire . . . , sinon faire . . .

3.4.1 Structure alternative complète



formalisme du diagramme d'états

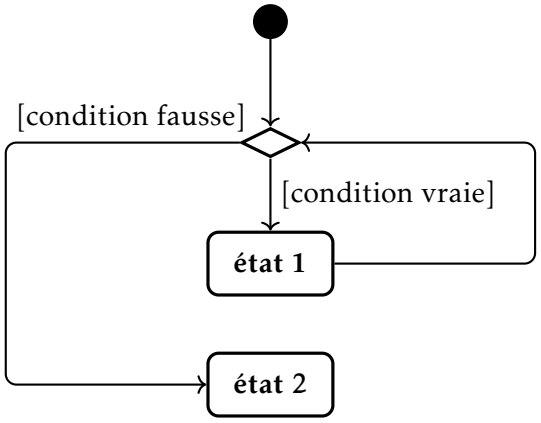
3.4.2 Structure alternative avec saut



formalisme du diagramme d'activité

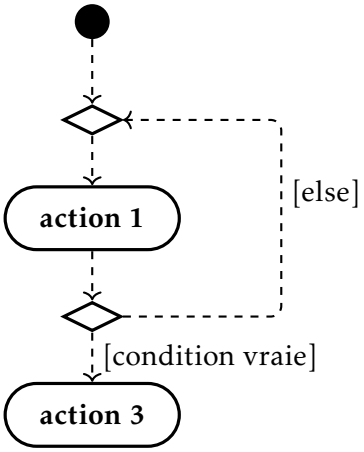
3.5 Les structures répétitives (itératives)

Tant que condition vraie, faire ...



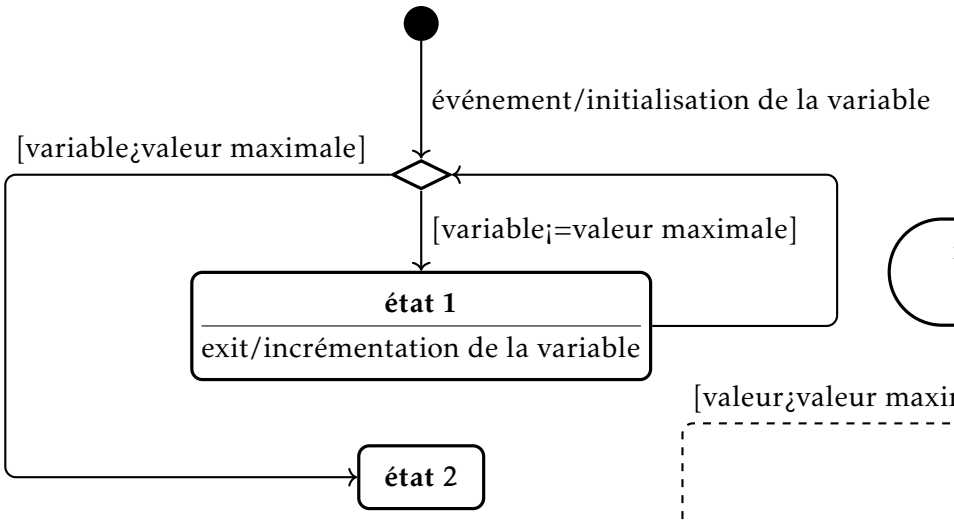
formalisme du diagramme d'états

Répéter...jusqu'à condition vraie

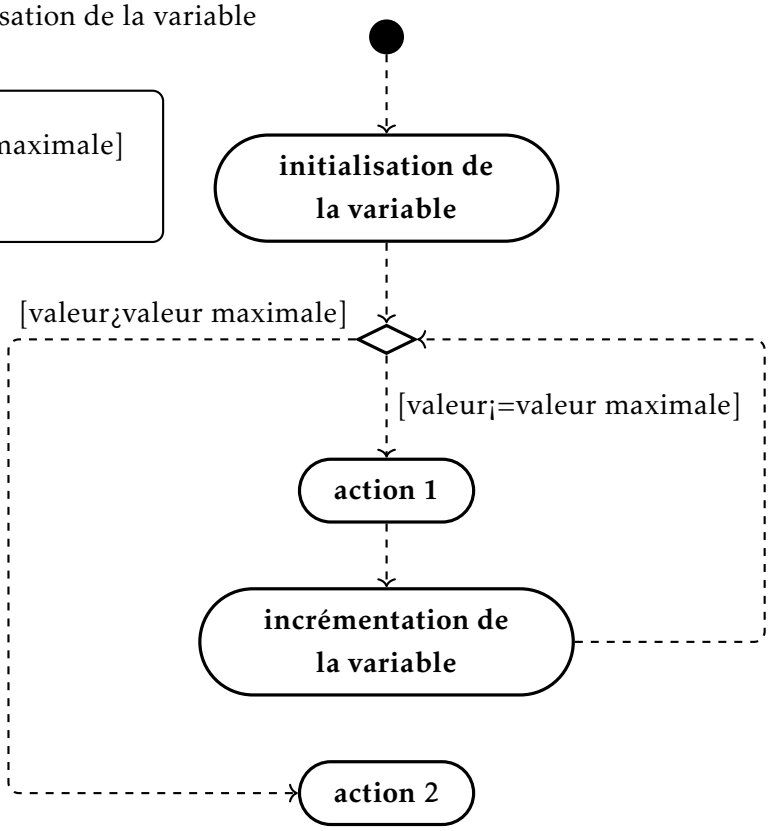


formalisme du diagramme d'activité

Pour variable = valeur initiale, jusqu'à valeur maximale, faire...



formalisme du diagramme d'états



formalisme du diagramme d'activité