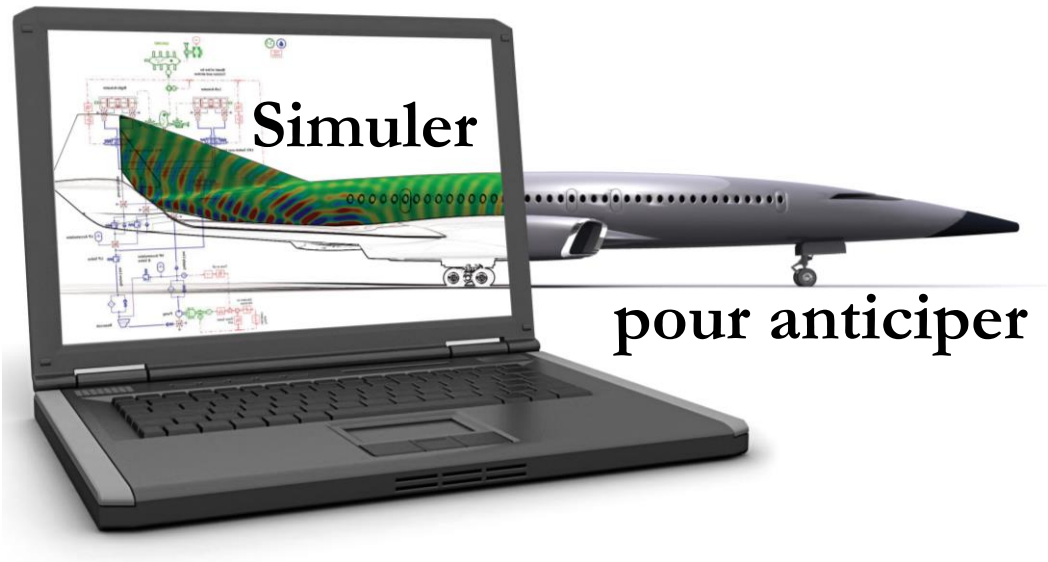
SYSTEMES à évènements discrets SEQUENTIELS

Objectifs - extraits du référentiel de 1ère et 2nde année (MP et PSI) pour les systèmes à évènements discrets

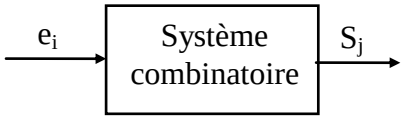
Compétence visée	*Savoir-faire associé*
Appréhender les analyses fonctionnelle et structurelle	*Interpréter tout ou partie de l'évolution temporelle d'un système à partir des diagrammes de séquence et d'états*
Proposer un modèle de connaissance et de comportement	*Exprimer un fonctionnement par des équations logiques (ET, OU, NON)*
	Représenter tout ou partie de l'évolution temporelle sous forme de chronogramme
	Décrire et compléter un algorithme représenté sous forme graphique



Sciences Industrielles de l'Ingénieur
1ère année de CPGE
Lycée Claude Fauriel

1 - Définitions

Dans un système combinatoire, à chaque combinaison de valeurs des variables d'entrée est associée une et une seule combinaison de valeurs des variables de sorties :



$S_j = f(e_i)$

Considérons l'exemple du portail automatisé. Le moteur M est mis en marche par appui sur un bouton poussoir m, le fonctionnement peut être décrit par le chronogramme suivant :

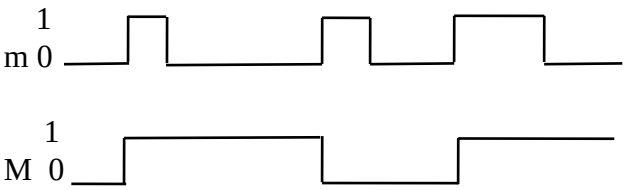
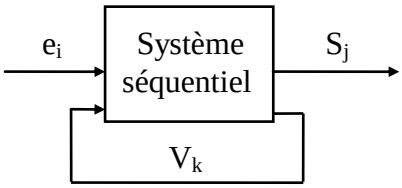


Table de vérité :

m	M
0	?
1	?

Ce système est-il combinatoire ? Non, car à un état de la variable d'entrée m correspondent deux états possibles de la sortie M. Le comportement du système ne dépend pas seulement des entrées à l'instant considéré mais de l'évolution antérieure du système. Il faut donc pouvoir caractériser le passé du système avec un certain nombre de variables qui devront être **mémorisées** à l'intérieur de ce système: on les appelle les **variables internes V_k** et elles rendent compte de l'état interne du système à un instant donné.

On peut schématiser un système à événements discrets (appelé aussi automatisme séquentiel) de la façon suivante :



$S_j = f(e_i, V_k)$

Voici ci-dessous l'ibd qui représente le fonctionnement d'un système à événements discrets :

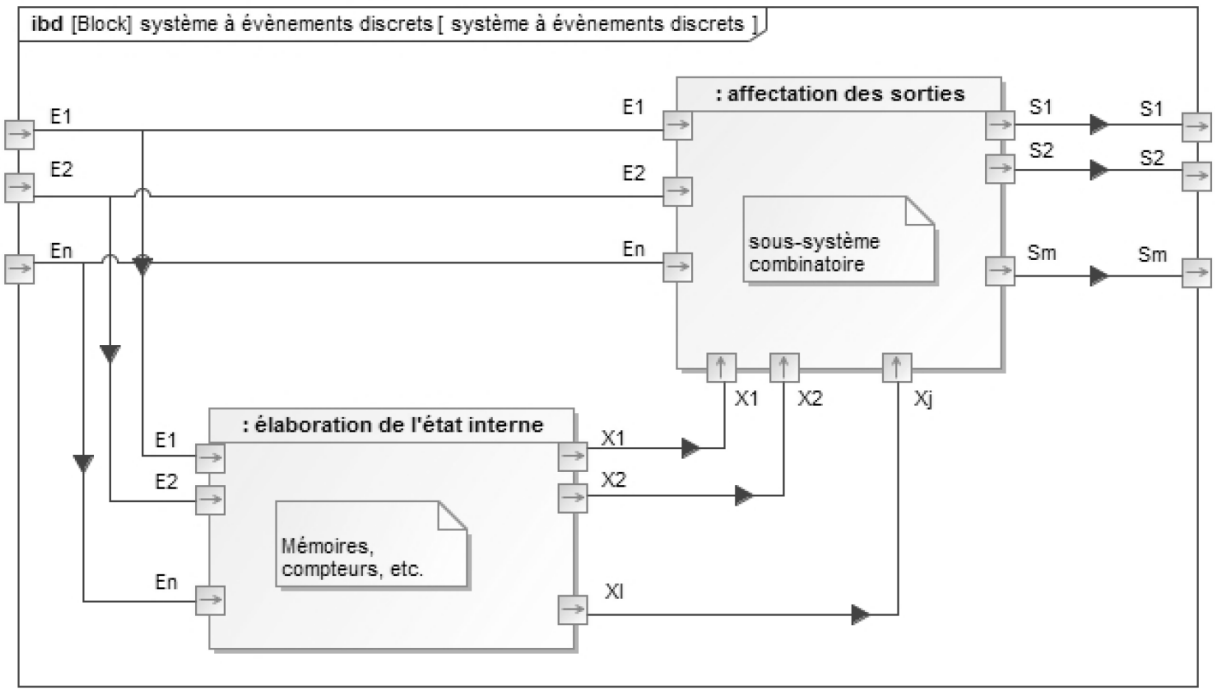
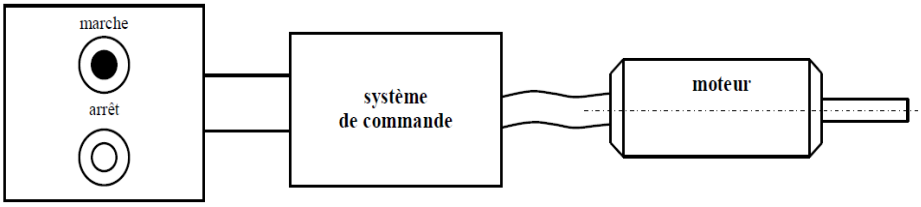


Illustration : commande tout ou rien (TOR) d'un moteur

2 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.

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 ; 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.

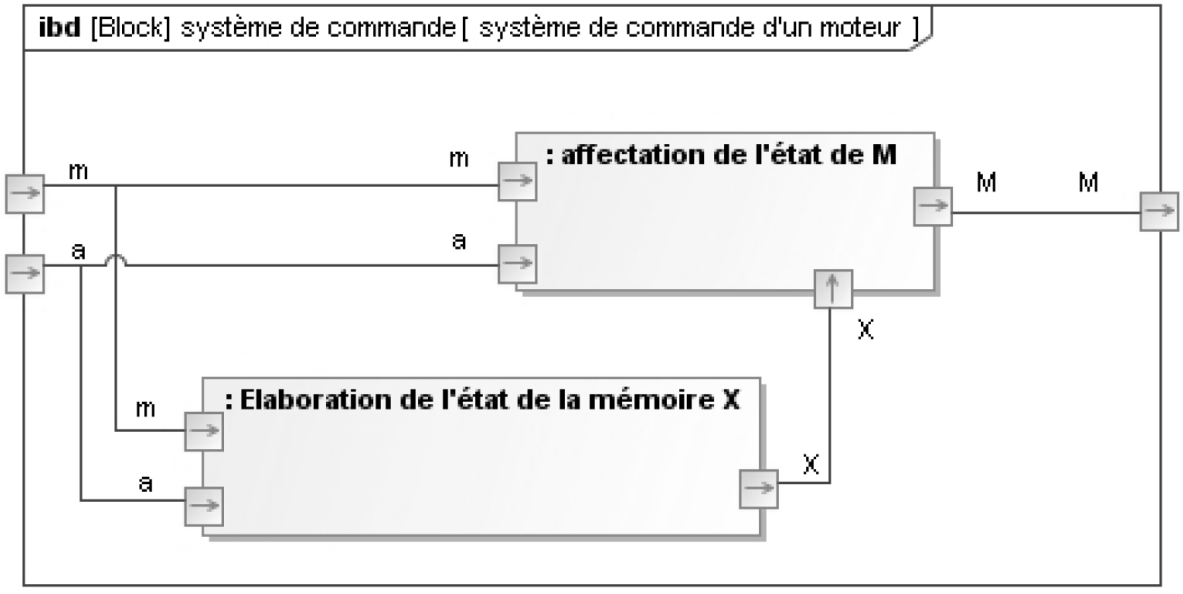
La table de vérité du sous-système « affectation de l'état de M » est alors la suivante :

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

d'où l'équation logique de M :

$$M = (m + X).\overline{a}$$

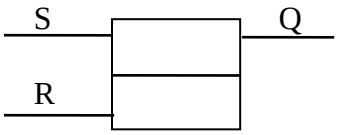
En langage SysML, le diagramme de blocs internes s'écrit :



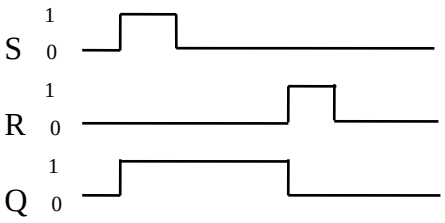
2 - Fonction mémoire

2.1 - La bascule RS

C'est le composant élémentaire ; c'est un circuit qui a pour propriété de conserver la valeur binaire précédemment inscrite en l'absence de toute sollicitation extérieure. Il comporte deux entrées R (comme *Reset* ou "remise à 0" de la sortie) et S (comme *Set* ou "mise à 1" de la sortie) et une sortie Q (la sortie complémentée $\overline{Q}$ est parfois disponible dans certaines technologies).



* Symbole de la bascule RS (logigramme)



* Chronogramme de la bascule RS

Le changement d'état des sorties est uniquement activé lorsque l'une des entrées R ou S commute de 0 à 1. Leur retour à 0 est sans effet.

*Table de vérité

S	R	Q^+
0	0	Q^-
0	1	0
1	0	1
1	1	**

Les notations Q^- et Q^+ correspondent à l'état de la sortie Q avant et après une modification des entrées.

← cet état dépend du fonctionnement de la bascule:
- 0 si bascule à **déclenchement** (ou arrêt) **prioritaire**
- 1 si bascule à **enclenchement** (ou marche) **prioritaire**
- Q^- si bascule sans priorité

***** D'où les équations logiques :

- mode à arrêt prioritaire : $Q^+ = Q^- \overline{S}\ \overline{R} + S\ \overline{R} = \overline{R}\ (S + Q^-)$

R \ S	0	1
0	Q^-	1
1	0	0

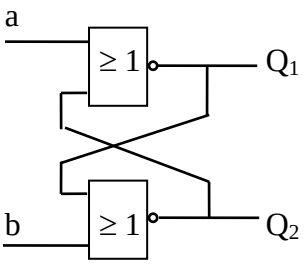
- mode à marche prioritaire : $Q^+ = Q^- \overline{S}\ \overline{R} + S\ \overline{R} + R\ S = \overline{R}\, Q^- + S$

R \ S	0	1
0	Q^-	1
1	0	1

Remarque :

Les outils vus pour la représentation des systèmes combinatoires (table de vérité, fonctions logiques, tableau de Karnaugh, logigramme, …) peuvent également être utilisés pour la représentation des systèmes à événements discrets ; cependant cet usage se limitera à des cas simples.

2.2 - Réalisation à l'aide d'opérateurs NOR



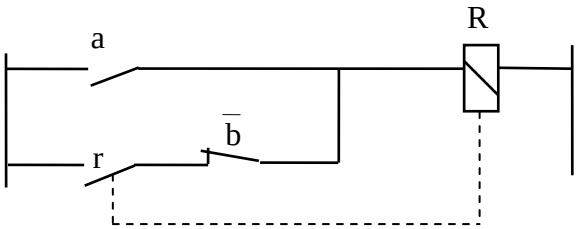
$Q_1 = \overline{a + Q_2} = \bar{a}\ (b + Q_1)$

$Q_2 = \overline{b + Q_1} = \bar{b}\ (a + Q_2)$

b	a	Q_1	Q_2
0	0	$\bar{Q_1}$	$\bar{Q_2}$
0	1	0	1
1	0	1	0
1	1	0	0

On a (table de vérité) une bascule à déclenchement prioritaire avec $Q_2 = \overline{Q_1}$ (attention : seulement pour les entrées (0,1) et (1,0)).

2.3 - Mémoire câblée électromagnétique

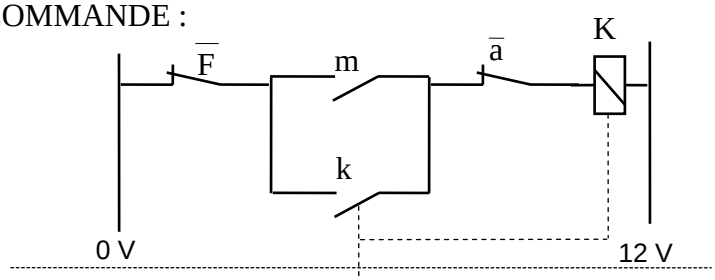


a = bouton poussoir
à contact normalement ouvert
b = bouton poussoir
à contact normalement fermé
R = relais commandant un contact r
normalement ouvert

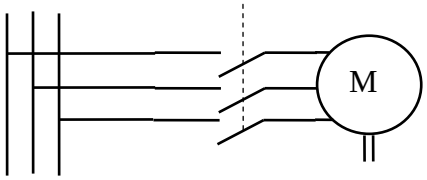
équation : $R = a + R^{-}.\bar{b} \Rightarrow$ marche prioritaire (set = a et reset = b)

Application : commande manuelle d'un moteur électrique

COMMANDE :



PUISSANCE :



Le moteur triphasé est commandé à l'aide de deux boutons poussoirs :
- m = mise en marche
- a = arrêt

En cas de commande simultanée de marche et arrêt, le moteur doit s'arrêter.

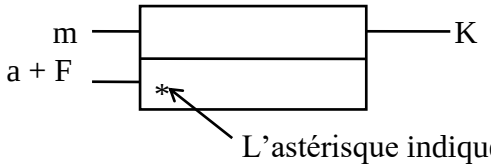
Une protection en cas de surintensité est assurée par le relais thermique F qui voit son contact normalement fermé s'ouvrir, entraînant la coupure de l'alimentation de K, donc celle du moteur.

Équation logique de K : $K^{+} = \bar{F}\ \bar{a}\ (m + K^{-})$

La mise en marche du moteur se fait par action sur m, ce qui alimente le relais K et ferme les contacts k correspondants. L'action de a ou F arrête le moteur.

Si a = m = 1, alors K = 0 : on a bien un arrêt prioritaire.

On peut représenter cette commande de la façon suivante :



L'astérisque indique que la mémoire est à arrêt prioritaire.

3 – La machine d'état

La **machine à nombre fini d'états** (FSM : Finite State Machine), est aussi appelée **automate fini**. Elle peut être une entité matérielle ou une entité conceptuelle comme un algorithme. Elle comporte un nombre fini d'états.

La machine d'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.

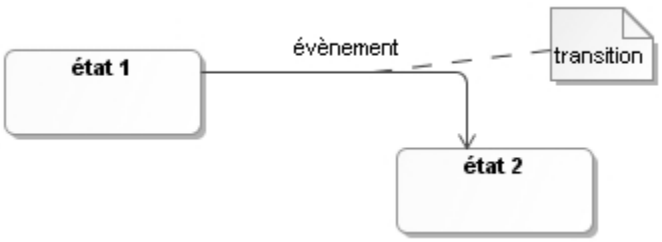
Un **évènement** est une description d'occurrence qui conduit à une évolution du comportement du système. On l'appelle aussi un déclencheur (trigger).

4 – Le diagramme d'états

4.1 – Présentation et formalisme

Le diagramme d'états (state machine diagram ou **stm**) est un diagramme normalisé SysML, il 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.

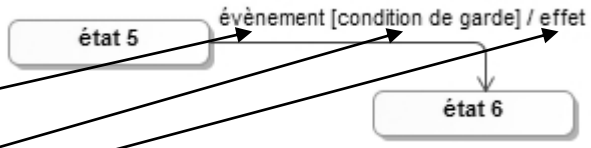


A un instant donné, le système ne peut être que dans **un seul état**.

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.
Les cas d'évolution du système entre les états sont identifiés par les transitions. On trace alors un lien entre les nœuds correspondants. Chaque transition peut être qualifiée par un évènement, une condition de garde et un effet.

4.2 – Les transitions

A une transition peuvent être associés :



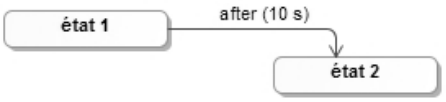
un évènement,
et/ou une condition de garde,
et/ou un effet (une action).

4.2.1 – L'événement

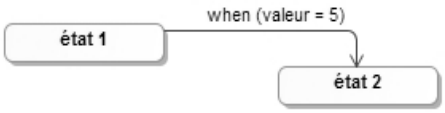
Il existe quatre types d'évènements associés à une transition :

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

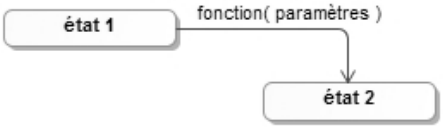
- l'**événement temporel** (time event) : un intervalle de temps s'est écoulé depuis l'entrée dans un état (mot clé « after ») ou un temps absolu a été atteint (mot clé « at »),



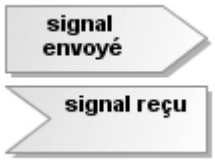
- l'**évènement de changement** (change event) : une valeur a changé de telle sorte que la transition est franchie (mot clé « when »),



- l'**évènement d'appel** (call event) : une requête de fonction (operation) 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 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 (recieve signal action).



4.2.2 – La condition de garde

La condition de garde est une expression booléenne (ou fonction logique) faisant intervenir des entrées et / ou des variables internes.

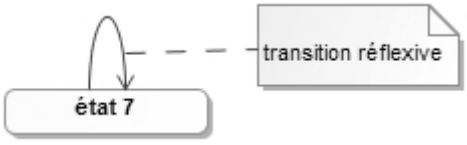
Elle autorise le passage d'un état à un autre.
Il est possible d'utiliser les notations non booléennes de front montant (↑) et front descendant (↓).

4.2.3 – L'effet

L'effet associé à une transition est effectué lorsque la transition est franchie, il ne prend pas de temps et ne peut pas être interrompue.
Son exécution peut par exemple provoquer un changement d'état ou l'émission d'un ordre.

4.2.4 – La transition réflexive

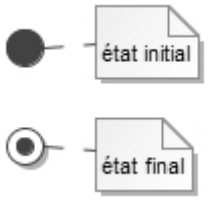
Une transition réflexive entraîne une sortie d'état puis un retour dans ce même état. Elle n'est pas sans conséquence.



4.3 – Les états

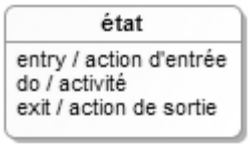
4.3.1 – L'état initial / l'état final

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.



4.3.2 – Activité et actions associées à l'état

A un état, on peut principalement rattacher :
- une action d'entrée,
- une activité,
- 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 »).

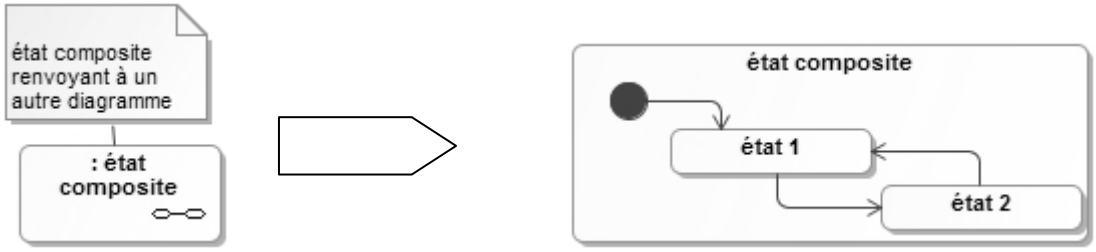
A contrario, une **action** ne prend pas de temps et ne peut pas être interrompue. Son exécution peut par exemple provoquer un changement d'état, l'émission d'un ordre pour un pré-actionneur ou un retour de valeur. On peut les trouver dans les transitions (effet) ou dans les états (mots clé « entry » ou « exit »).

Dans le langage SysML, les activités et actions sont directement liées aux fonctions (operations) définies dans les propriétés de bloc.

4.3.3 – L'état composite

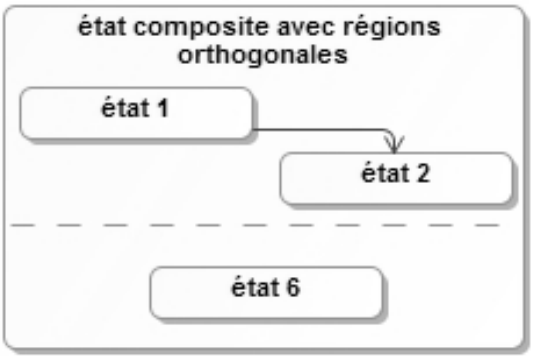
Un **état composite** est constitué de sous-états, il « appelle » donc un autre diagramme d'états.

Cela permet d'introduire la notion d'état de niveau hiérarchique inférieur et supérieur dans les diagrammes d'états : pour alléger l'écriture (et donc la lecture) d'un diagramme, un état dit composite regroupera plusieurs sous-états qui seront détaillés sur un autre diagramme (analyse descendante).



Un état composite peut contenir des régions dites orthogonales délimitées par un trait pointillé ;

dans chacune des régions, un seul état est actif.



4.4 – Synthèse sur le formalisme basique

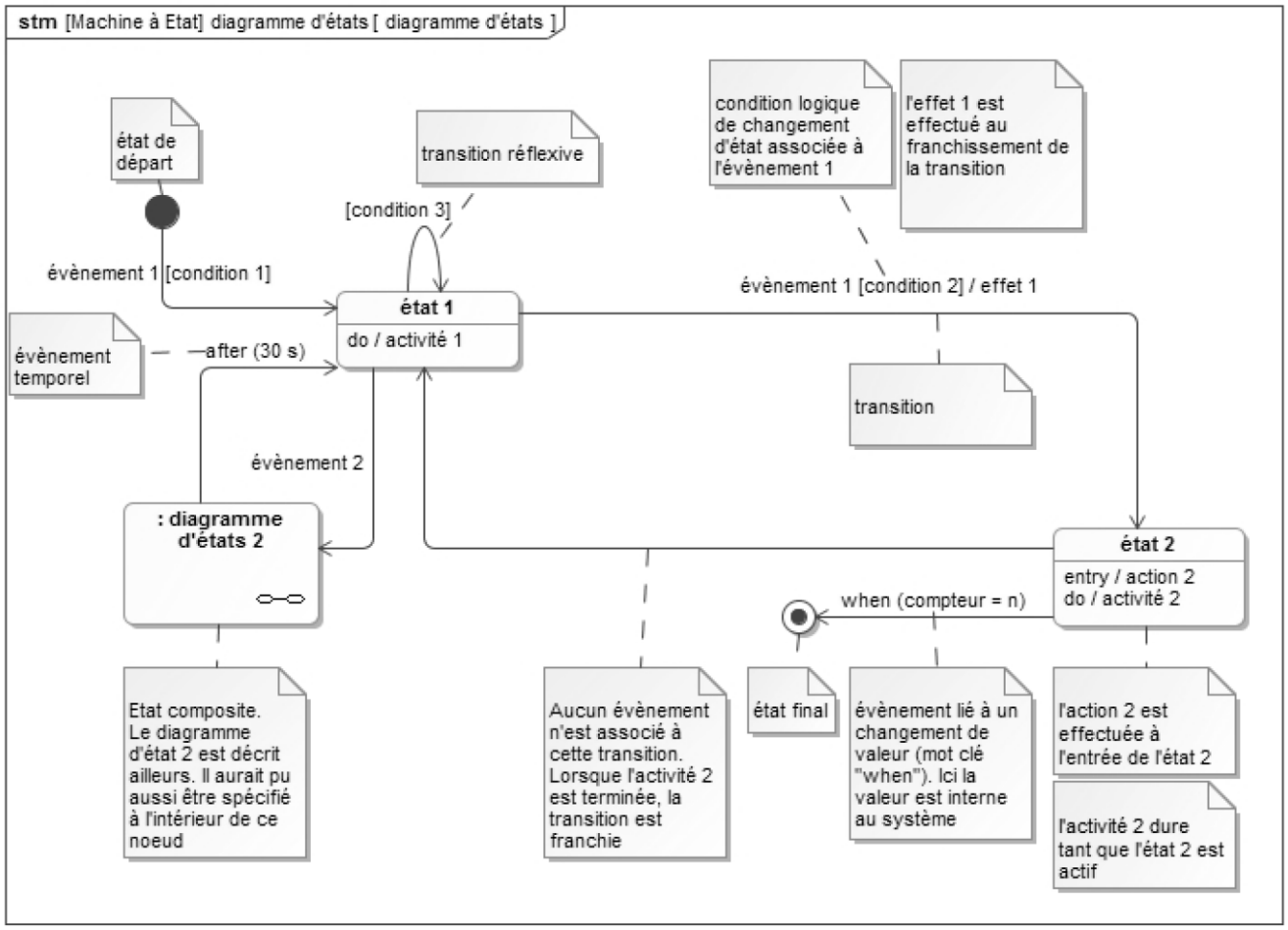
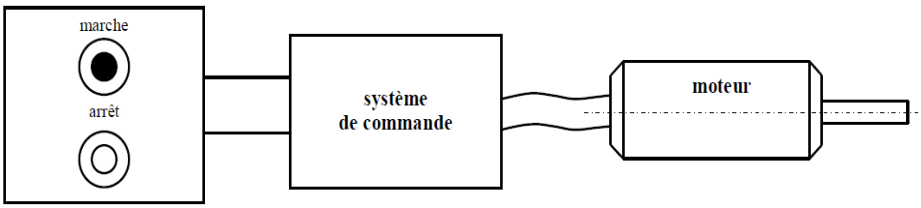


Illustration : commande tout ou rien (TOR) d'un moteur

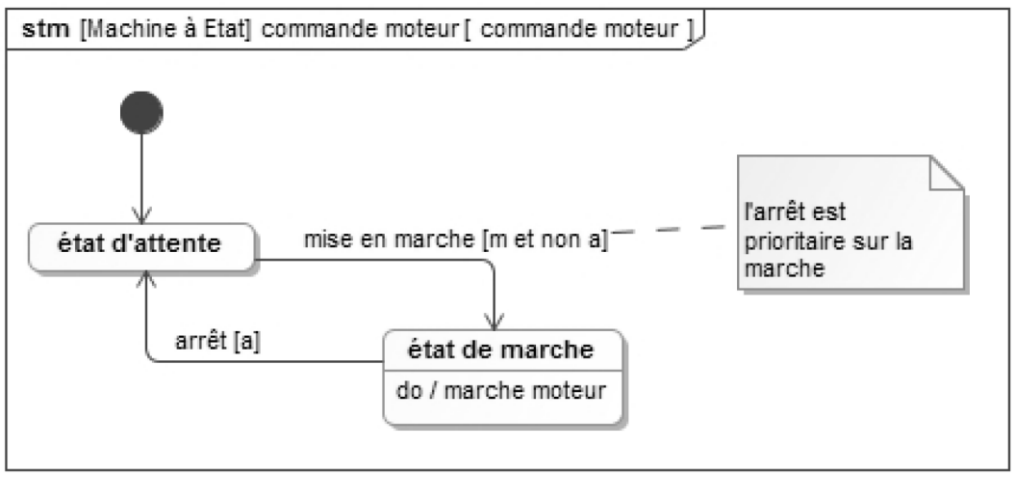
2 boutons poussoir m (marche) et a (arrêt) commandent le fonctionnement d'un moteur que l'on caractérise par une variable logique de sortie M.



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 diagramme d'état s'écrit :



4.5 – Les pseudo-états

Un **pseudo-état** est un élément de commande (sous forme de symboles spécifiques, en général entre les états) qui permet d'enrichir les diagrammes d'états.

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

Le formalisme SysML admet 9 pseudo-états :

« **entry point** » permet de créer un point d'entrée du diagramme.

« **exit point** » permet de créer un point de sortie vers un autre diagramme.

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

« **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.

« **fork** » et « **join** » divergence et convergence de séquences parallèles.

« **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.

Synthèse sur les pseudo-états :

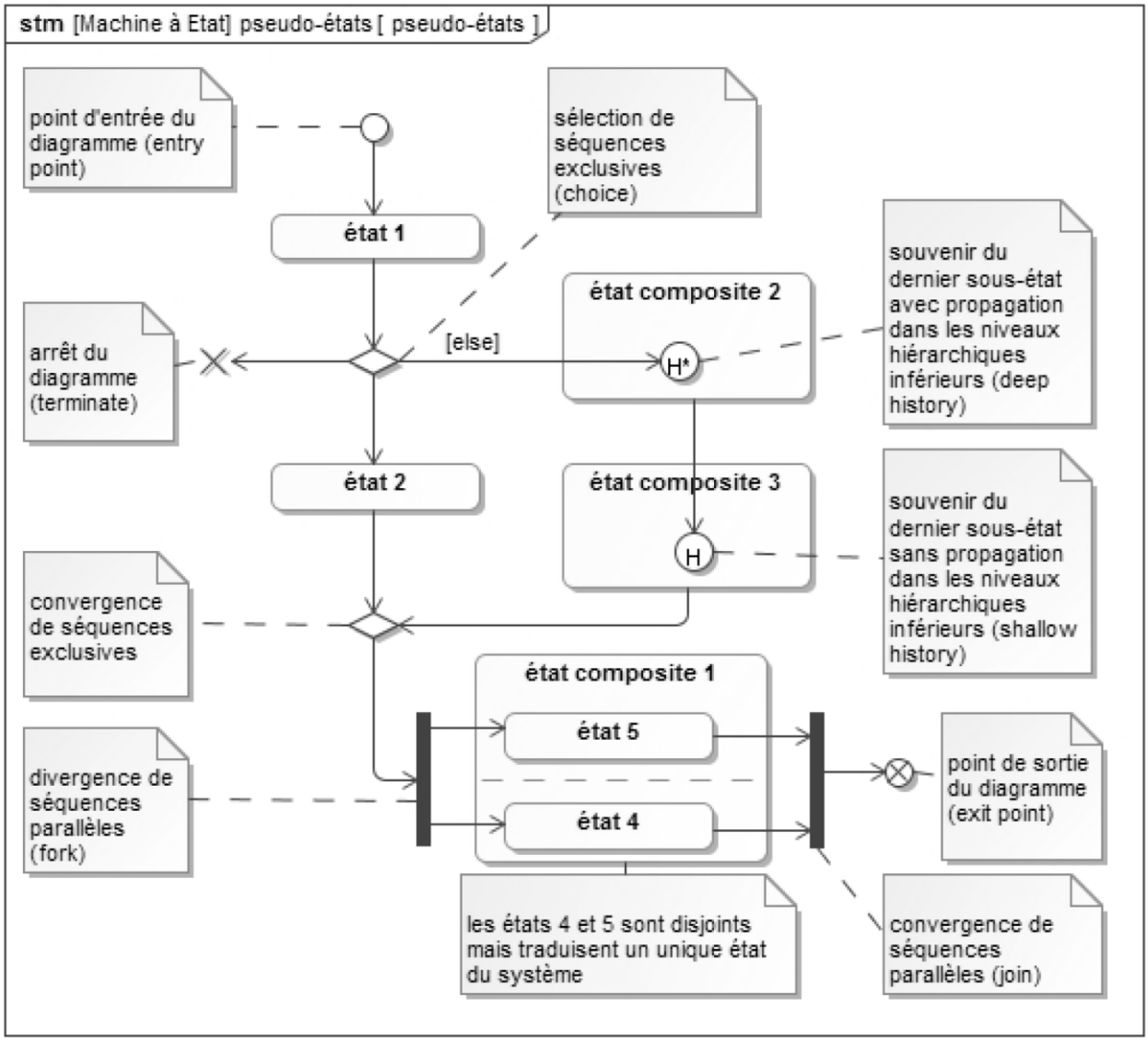


Illustration : commande tout ou rien (TOR) d'un moteur

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

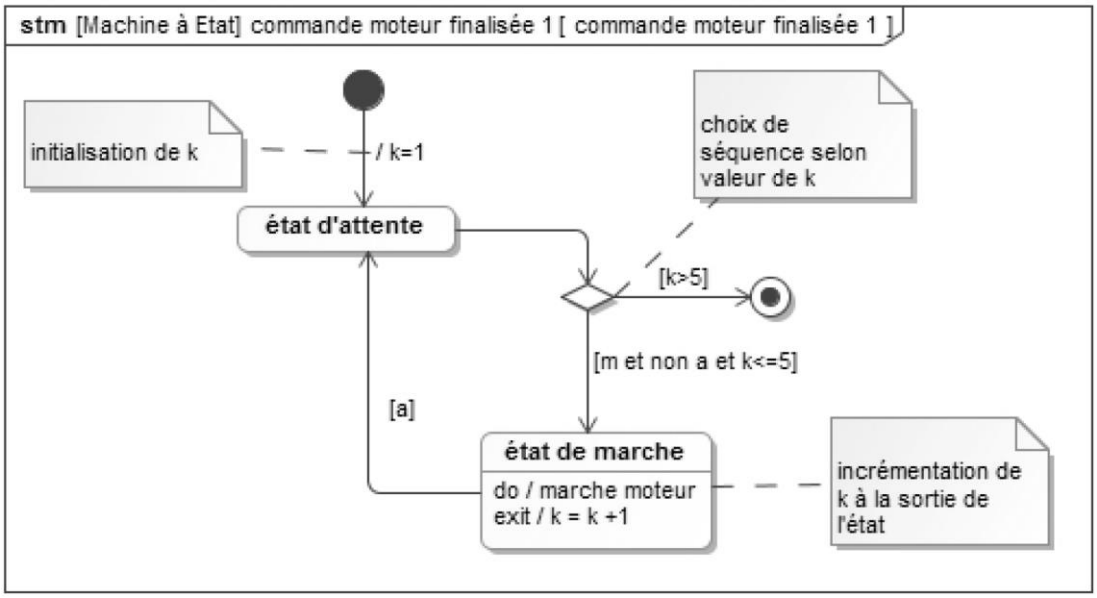
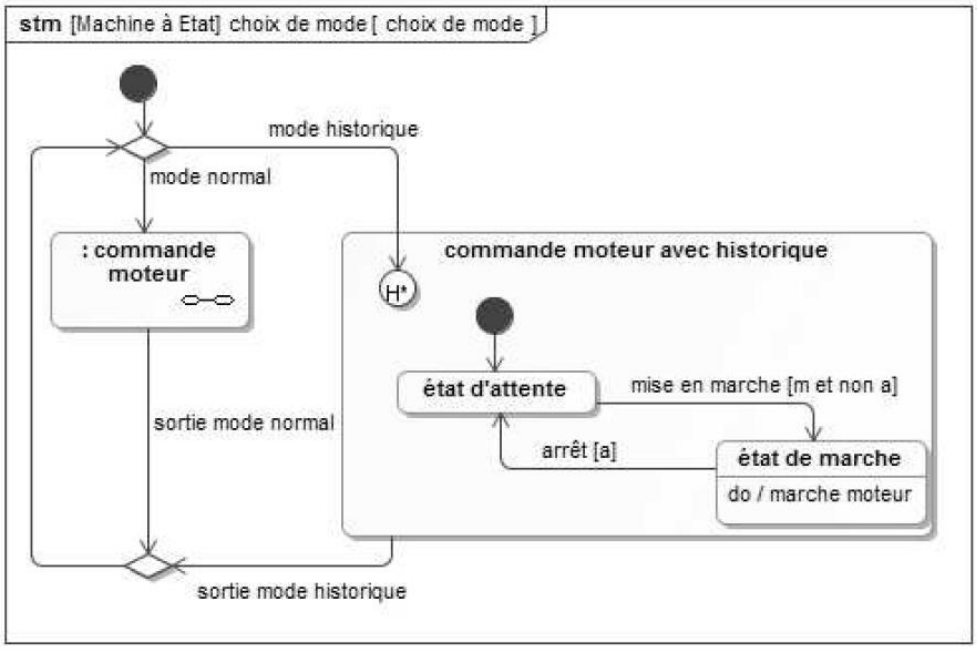


Illustration : commande tout ou rien (TOR) d'un moteur

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 »).



5 – Application aux structures algorithmiques

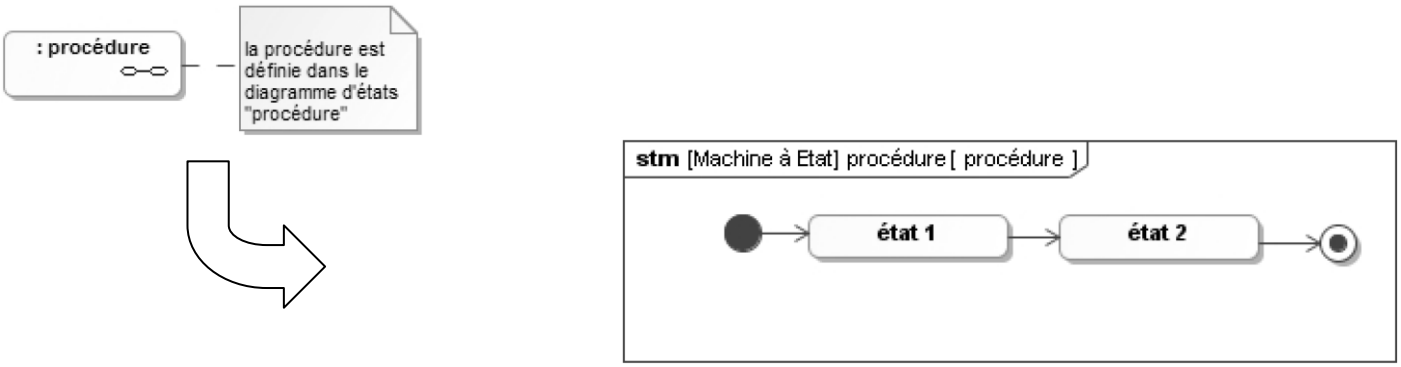
5.1 – Fonctions et procédures

La décomposition d'un algorithme en fonctions et procédures permet :

- de scinder une problématique générale en plusieurs problématiques élémentaires,
- de pouvoir réutiliser des sous-programmes réalisant des tâches élémentaires.

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

Procédure réalisée en diagramme d'états : (avec un état composite)

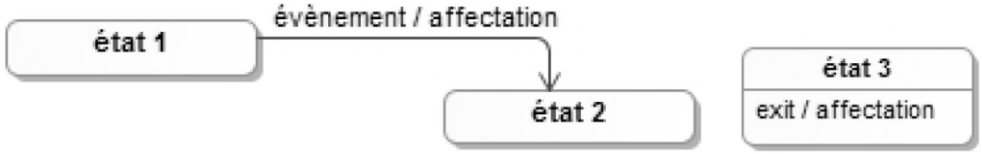


A la fin de l'exécution d'une **fonction**, il y a le retour d'une valeur, d'une liste, d'un objet, etc.

5.2 – Illustrations

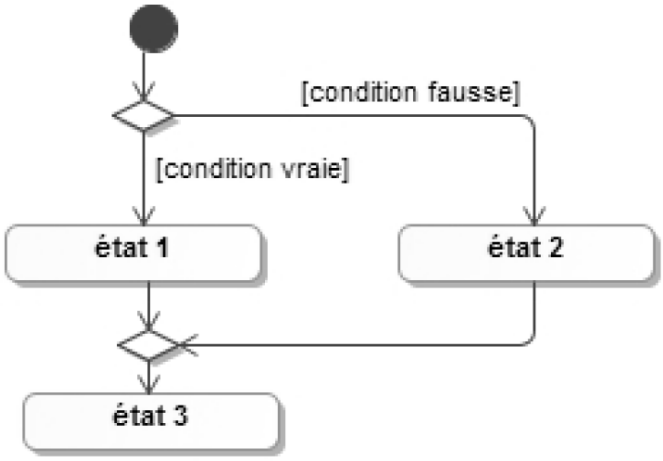
Affectation

d'une valeur à une variable : cette action se fait en un temps nul.



Structure alternative conditionnelle

si …, alors faire …, sinon faire …



Structure répétitives itératives

pour variable = valeur initiale, jusqu'à valeur maximale, faire …