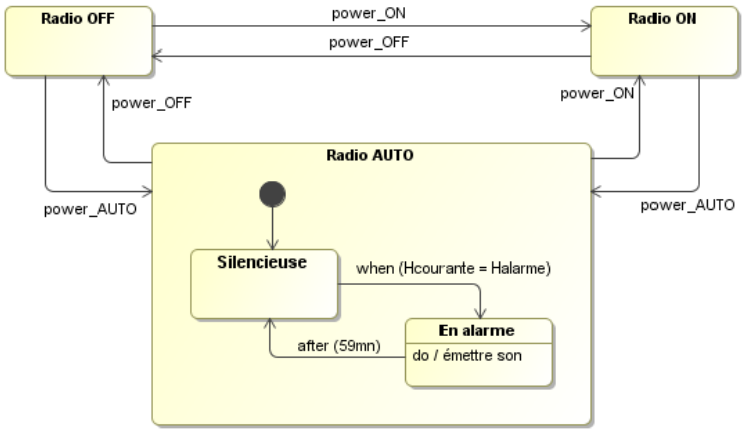
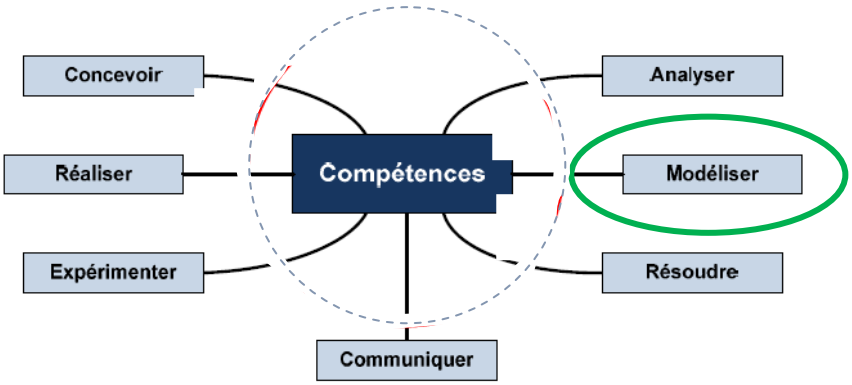
Cycle 8: Modélisation des systèmes à évènements discrets

Chapitre 3 : Diagrammes d'états



Compétences:

- Modéliser les systèmes à évènements discrets (diagrammes d'état)
- Traduire le comportement d'un système à évènements discrets



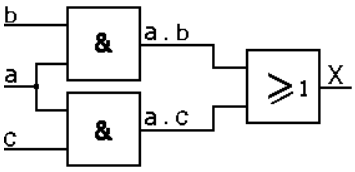
Sommaire

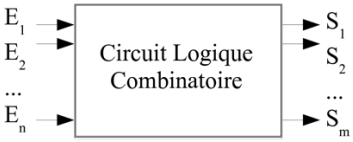
1. Rappels: système combinatoire - système séquentiel

1.1. Système combinatoire

Une **même cause produit toujours le même effet**. Un même état des entrées donne toujours le même état des sorties. un système est combinatoire si toutes les sorties sont de nature combinatoire.

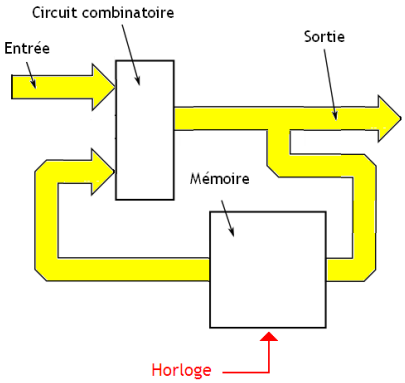
La logique combinatoire utilise **l'algèbre de Boole.**





1.2. Système séquentiel

Une **même cause peut produire des effets différents.** Un même état des entrées peut donner des états différents des sorties. On tient compte de l'état du système, c'est un **système à mémoire.**

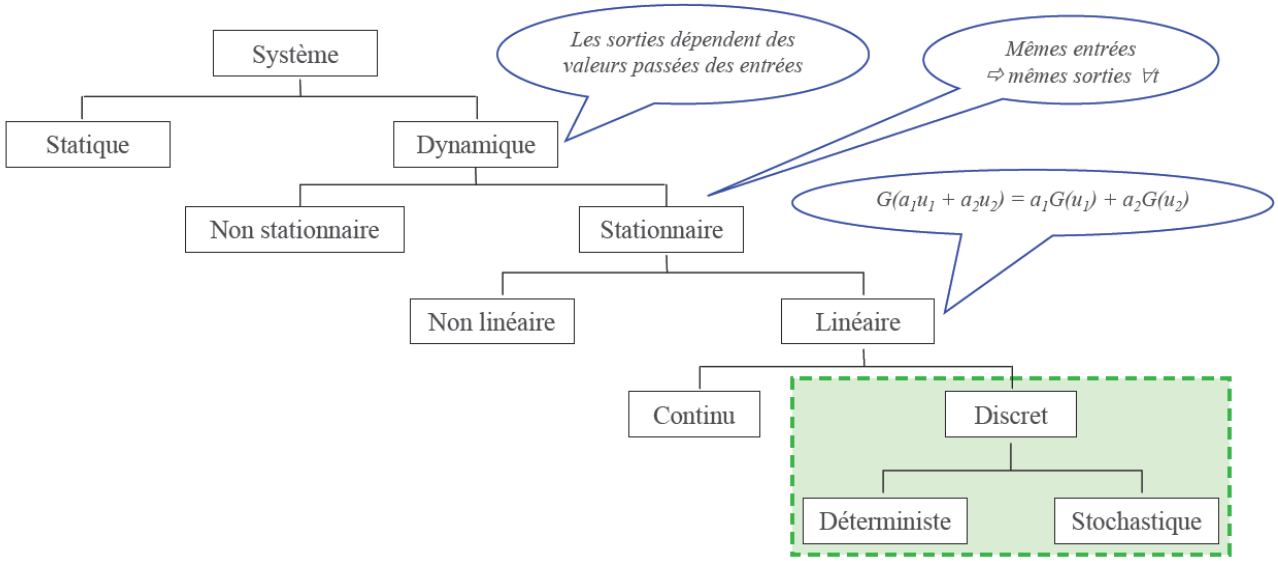


2. Problématique - système - modélisation comportementale

Problématique

Comment ***décrire et programmer le comportement séquentiel d'un système ?***

Système: *" Ensemble d'éléments en interaction mutuelle et en interaction avec l'environnement, organisés en fonction d'un même but pour parvenir à une même fin".*



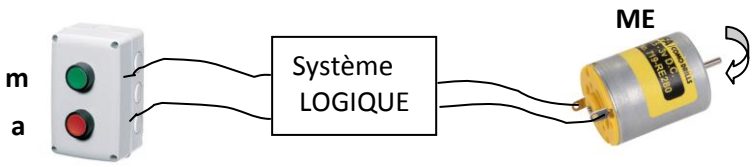
La description du comportement d'un système séquentiel peut être réalisé notamment par:

- **un chronogramme,**
- **un algorigramme,**
- **un diagramme de séquence SYSML,**
- **un diagramme d'états SYSML.**

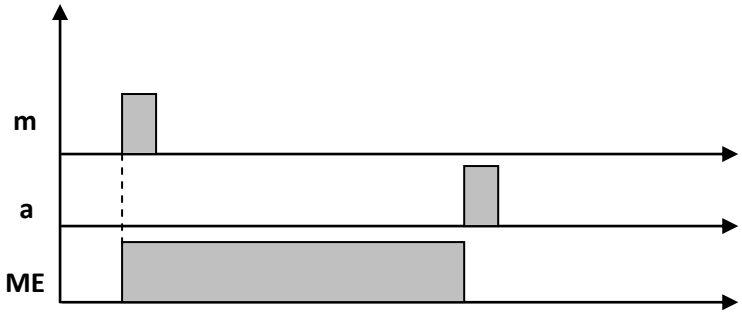
Ces outils sont, à la base, des ***outils de modélisation du comportement séquentiel.***

3. Exemple de système séquentiel

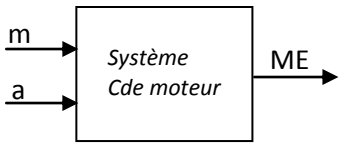
Soit un moteur électrique commandé par 2 boutons poussoirs.



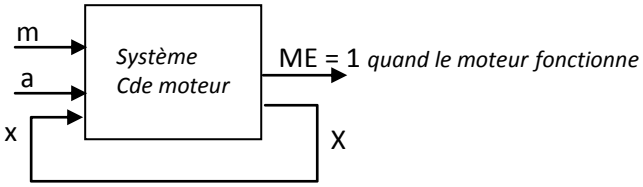
Le **chronogramme** de fonctionnement est le suivant:



Inventaire des E/S:



Lorsque les 2 entrées sont à 0, le moteur peut être en marche ou à l'arrêt. Le système est donc **séquentiel.** Nous allons utiliser une variable interne X pour caractériser l'état du système. on note "x" variable interne prise en compte comme entrée et "X" variable interne prise en compte comme sortie.



On peut représenter le fonctionnement par la **table de vérité suivante:**

m	a	x	ME
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	1

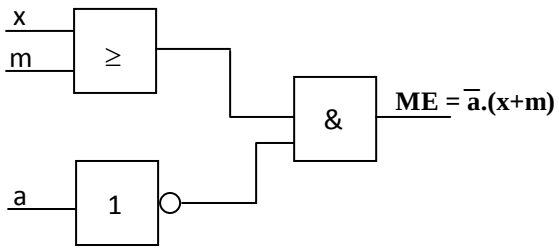
Rappels: $a.\overline{a}=0$ / $a+\overline{a}=1$ / $a+a=a$ / $a.a=a$

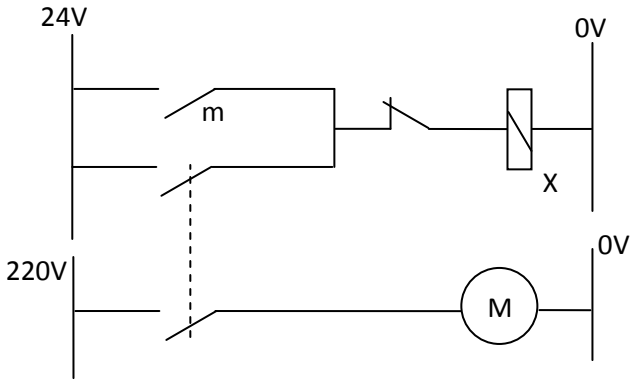
L'équation de fonctionnement est:

$$\begin{aligned}\mathbf{ME} &= \overline{m}.\overline{a}.x + m.\overline{a}.\overline{x} + m.\overline{a}.x \\ &= \overline{a}.x.(m+\overline{m}) + m.\overline{a}.\overline{x} \\ &= \overline{a}.x + m.\overline{a}.\overline{x} \\ &= \overline{a}.(x + m.\overline{x}) \\ &= \overline{a}.[(x+m).(x+\overline{x})] \\ &= \mathbf{\overline{a}.(x+m)}\end{aligned}$$

*On retrouve le comportement désiré + choix avec **arrêt prioritaire***

Exemple de schéma logique et de câblage:



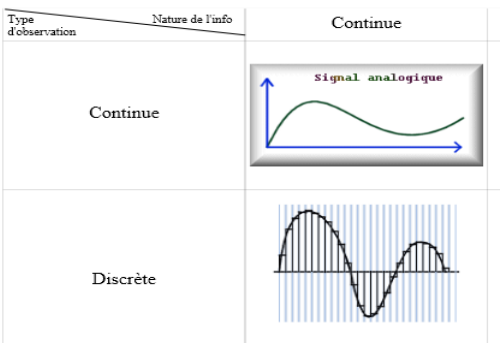


4. Les systèmes à événements discrets

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 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.

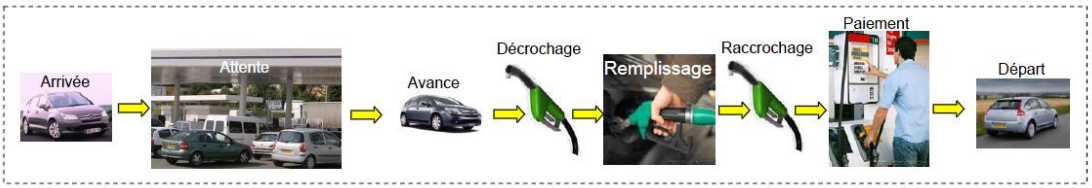
Les SED apparaissent de façon naturelle dans la modélisation des systèmes informatiques, des réseaux de télécommunications, des réseaux de transport ou des systèmes de production (lignes d'assemblage, ateliers flexibles).

- Dans un système continu, l'état du système change en permanence avec l'évolution du temps.
- Dans un **système à évènements discrets**, l'état change seulement à certains instants lors de l'occurrence d'évènements particuliers.



Remarques:

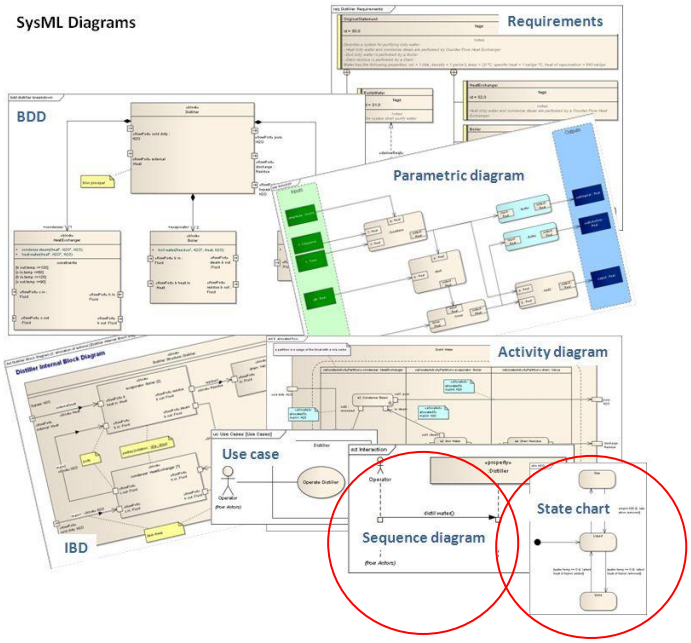
- *Le temps et les états du système réel évoluent de façon continue, mais on ne s'intéresse qu'à des instants particuliers*



- *Le signal de sortie est élaboré à partir d'un signal d'entrée logique (ou d'une combinaison de signaux) et doit prendre en compte une* ***chronologie pré-établie qui porte sur un nombre fini d'opérations****.*

5. Description du comportement d'un système avec les diagrammes SYSML

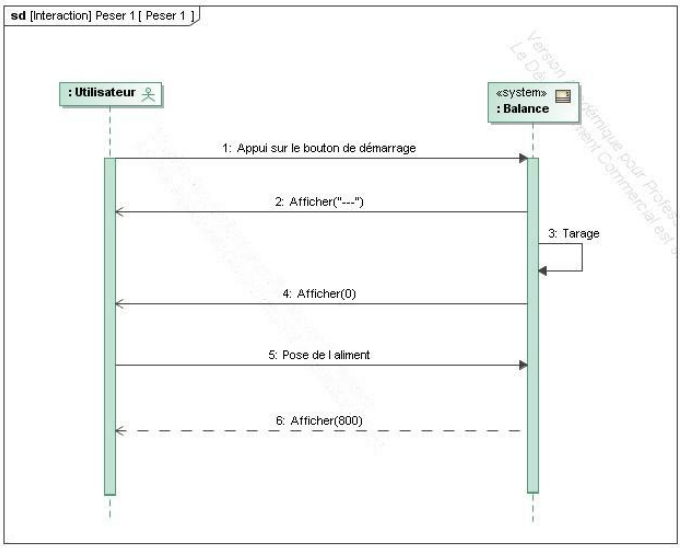
Les **diagramme de séquences et d'états** sont des diagrammes normalisés **SysML**. Ils permettent de décrire le comportement d'un système ou d'une de ses parties. Il fait partie des 9 diagrammes du langage SYSML.



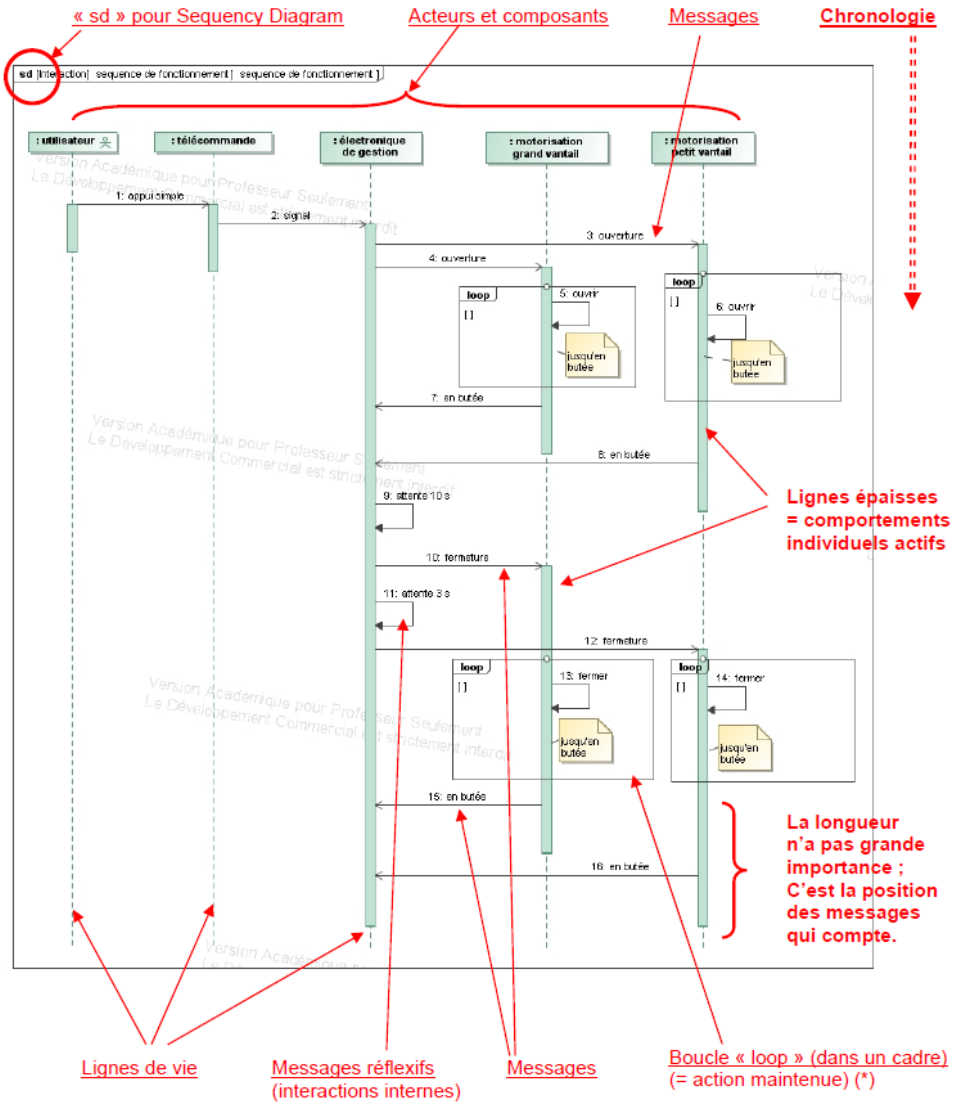
Rappel de cours sur le diagramme de séquence:

Il représente les **échanges de messages entre les acteurs et le système** ou entre des parties durant une séquence temporelle d'actions appelée **scénario.**
Il montre la séquence, représentation **verticale chronologique**, des messages passés entre blocs au sein d'une interaction.

exemple balance Halo



exemple portail automatisé



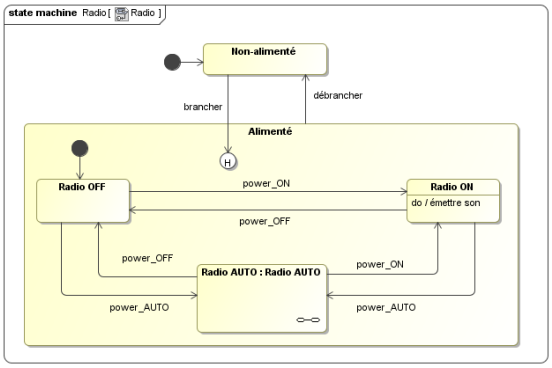
6. La machine à états

La **machine à nombre fini d'états** (FSM : Finit State Machine), est aussi appelée **automate fini**. Elle peut être **une entité matérielle** *(un microprocesseur ou automate programmable)*, 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.

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).

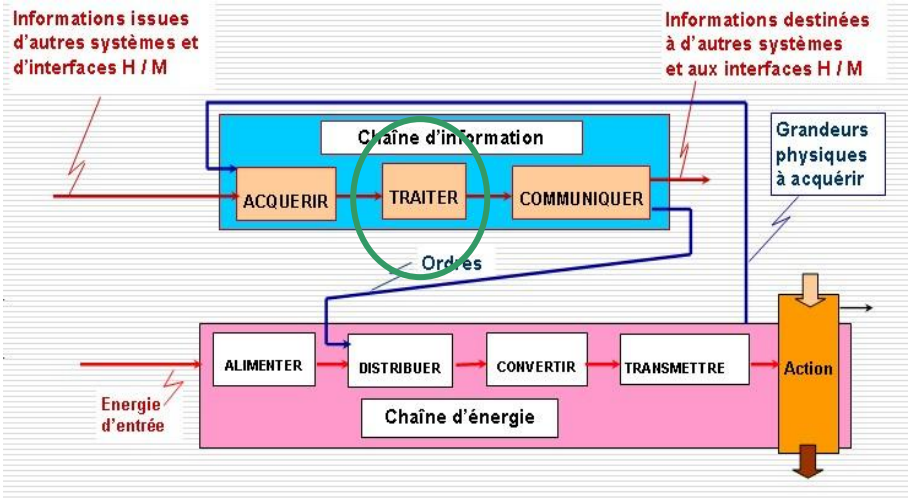
On conçoit les machines à états à l'aide **d'outils de modélisation**. Celui que nous allons développer ci après est issu du langage SYSML: **le diagramme d'états.**



Un diagramme d'états permet d'obtenir une machine à états réalisant la **partie commande** d'un système. Les états représentent toutes les valeurs que peuvent prendre au fil du temps les variables internes du circuit de logique séquentielle.

Le graphe d'états, comme l'algorigramme (cf. cours d'informatique), est essentiellement un outil graphique permettant de modéliser le comportement séquentiel, en termes de déroulement d'actions temporelles.

Mais il peut aussi servir **à programmer les composants réalisant la fonction "Traiter"** de la chaîne d'information *(microcontrôleur, microprocesseur, automate programmable, ...).* Les variables d'entrée de la fonction "Traiter" sont alors les informations fournies par la **fonction "Acquérir" (capteurs, …)** et les variables de sortie sont les ordres pour la **fonction "Distribuer"** de la chaîne d'énergie, éventuellement via la fonction "Communiquer".



7. Le diagramme d'états SYSML

SysML a repris le concept bien connu de machine à états finis, qui consiste à s'intéresser au **cycle de vie d'une instance générique d'un bloc particulier** *(qui peut être le système complet)* **au fil de ses interactions, dans tous les cas possibles**.

7.1. Définitions

Définition 3.6. Diagramme d'états - stm

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

Le diagramme d'états est rattaché à un bloc qui peut être le système, un sous-système ou un composant. Le comportement décrit par ce type de diagramme sert à montrer les différents états pris par le bloc en fonction des évènements qui lui arrivent.

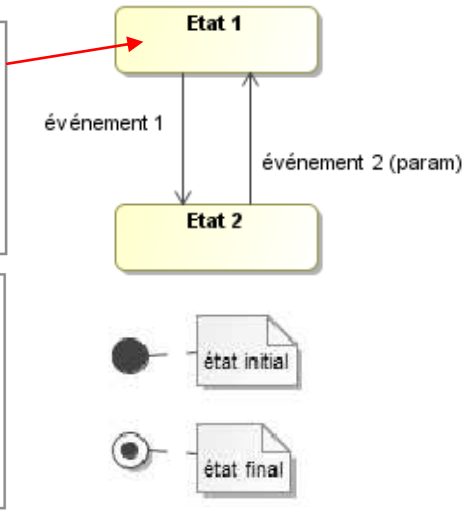
Un état représente une situation d'une durée finie durant laquelle un système exécute une activité, satisfait à une certaine condition ou bien est en attente d'un événement. Le passage d'un état à un autre se fait en franchissant une transition.

7.2. Syntaxe et notations de base

B.A.-BA État *: rectangle aux angles arrondis contenant son nom*

Un état représente une situation durant la vie d'un bloc pendant laquelle :
- il satisfait une certaine condition ;
- il exécute une certaine activité ;
- ou bien il attend un certain événement.

Un bloc passe par une succession d'états durant son existence. Un état a une durée finie, variable selon la vie du bloc, en particulier en fonction des événements qui lui arrivent.



B.A.-BA État initial, état final

En plus de la succession d'états « normaux » correspondant au cycle de vie d'un bloc, le diagramme d'états comprend également deux pseudo-états :
- l'état initial du diagramme d'états correspond à la création d'une instance ;
- l'état final du diagramme d'états correspond à la destruction de l'instance.
- Il est possible d'utiliser plusieurs états finals (ou finaux, les deux se disent...) pour distinguer par exemple la destruction normale de l'élément en fin de vie d'une destruction accidentelle.

A un état, on peut principalement rattacher **une activité (do), une action d'entrée (entry)** et **une action de sortie (exit)**. Les **actions sont supposées de courtes durées**. Attention, une **action ne peut pas être interrompue.**

État avec activités internes:
- ✓ Entry /action: action exécutée à l'entrée de l'état;
- ✓ Exit /action: action exécutée à la sortie de l'état ;
- ✓ Do /action: action récurrente exécutée dans l'état;

Exemple:

Composer un numéro

Entry/ *afficher tonalité ligne libre*
Do/ *mémoriser numéro composé*
Exit/ *appeler le numéro*

Exemples:
- *activité: ouvrir une porte, sortir vérin, rotation moteur...*
- *action: émettre ordre pré-actionneur, incrémenter variable...*

B.A.-BA Événement

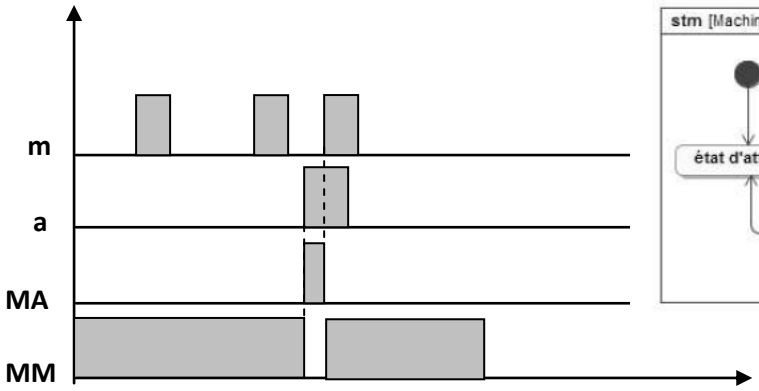
Spécification d'une occurrence qui peut déclencher une réaction sur un élément et qui possède une localisation dans le temps et l'espace. ***= front montant***

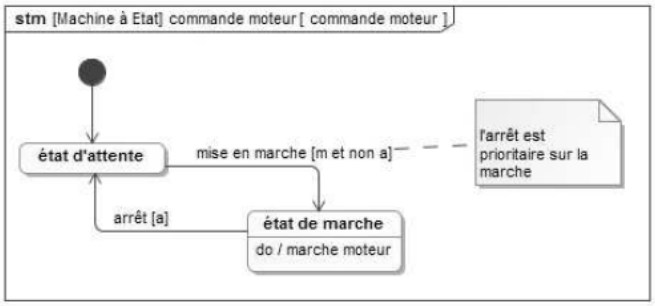
ex: Appui Bouton Poussoir

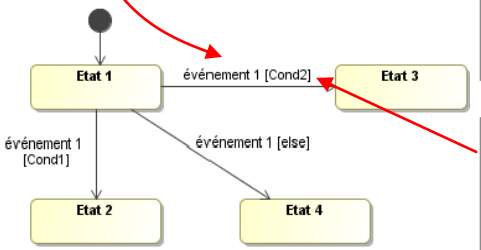
exemples: appuis BP, détection objet, comptage...

Méthode de travail pour comprendre les événements (ex de la commande moteur):

- On regarde sur le diagramme quel est l'état actif
- On en déduit l'évènement qui peut amener à sortir de cet état







B.A.-BA Transition ***: flèche orientée = lien entre 2 états***

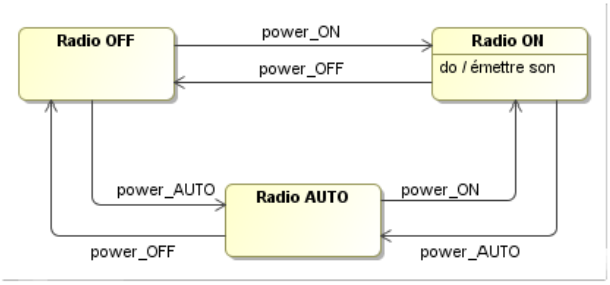
Une transition décrit la réaction d'un bloc lorsqu'un événement se produit (généralement le bloc change d'état, mais pas forcément).

B.A.-BA Condition ***: expression booléenne entre crochets***

Une condition (ou condition de garde) est une expression booléenne qui doit être vraie lorsque l'événement arrive pour que la transition soit déclenchée.
Elle peut concerner les valeurs du bloc concerné ainsi que les paramètres de l'événement déclencheur.
Plusieurs transitions avec le même événement doivent avoir des conditions de garde différentes.

7.3. Compléments et notations avancées : **exemple du radio réveil**

Commençons par déclarer trois états principaux correspondant aux **trois positions du bouton physique** permettant **d'allumer la radio, de l'éteindre, ou d'armer l'alarme du réveil**. Les événements de changement de position sont nommés power_ON, power_OFF, power_AUTO. Dans l'état Radio ON, nous avons déclaré une activité durable do/émettre son.



En fait, dans l'état Radio AUTO, la radio est silencieuse jusqu'à ce que l'heure courante devienne l'heure d'alarme : *when (Hcourante = Halarme).*
Ensuite, la radio s'éteint toute seule au bout de 59 mn : *after (59 mn).*

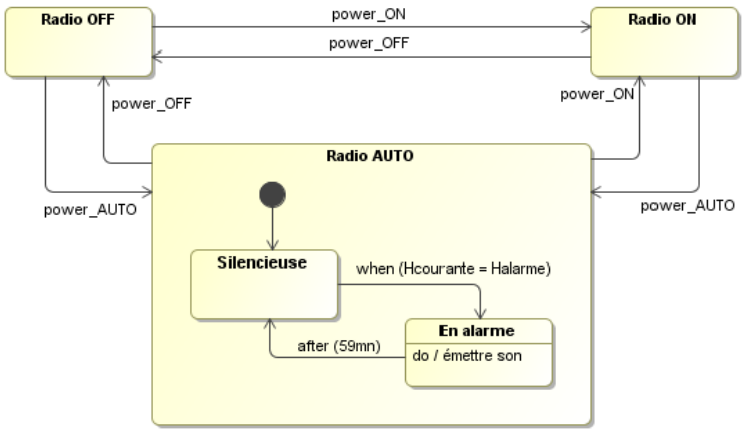
ATTENTION Événement interne

Les changements d'état internes(change event) se modélisent en utilisant le mot-clé `when` suivi d'une expression booléenne dont le passage de faux à vrai déclenche la transition.

ATTENTION Événement temporel

Lepassage du temps (time event) se modélise en utilisant le mot-clé `after`suivi d'une expression représentant une durée, décomptée à partir de l'entrée dans l'état courant.

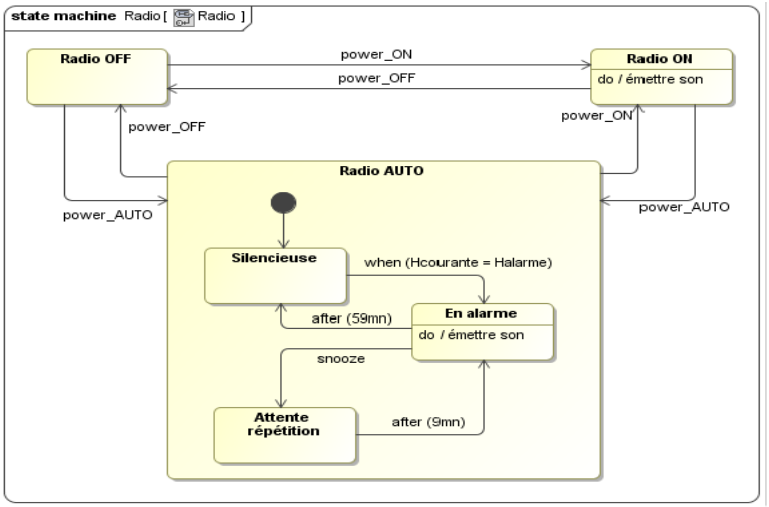
Pour modéliser ce comportement plus détaillé, une première solution consiste à transformer l'état Radio AUTO en **état composite** (encore appelé super-état), et à dessiner les nouveaux états à l'intérieur. Notez qu'il faut ajouter un sous-état initial pour indiquer que lorsque le bloc passe dans l'état Radio AUTO, il rentre en fait directement dans le sous-état Silencieuse.



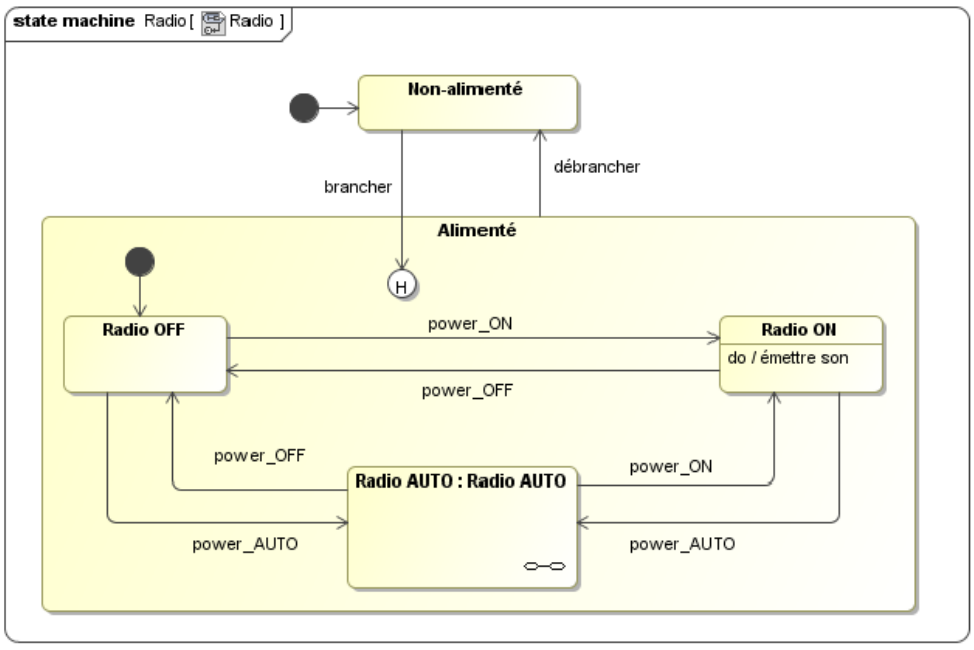
B.A.-BA État composite

Un état composite (aussi appelé super-état) permet d'englober plusieurs sous-états exclusifs. On peut ainsi factoriser des transitions déclenchées par le même événement et amenant vers le même état cible (comme power_ON ou power_OFF dans l'exemple), tout en spécifiant des transitions particulières entre les sous-états.

Ajoutons maintenant le comportement du **bouton « Snooze ».** Quand l'utilisateur appuie sur ce bouton, il interrompt provisoirement le son de la radio qui se réactive automatiquement après 9 mn.



Imaginons maintenant que nous souhaitons **modéliser la mémorisation de l'état de la radio après une coupure de courant intempestive**. Il faut déjà ajouter deux états : **Alimenté et Non-alimenté**, avec deux transitions provoquées par les événements brancher et débrancher. L'état Alimenté correspond en fait au super-état de tous ceux que nous avons modélisé jusqu'à présent. Au passage, nous avons opté pour faire de l'état Non-alimenté notre état initial.



Ce modèle est bon si l'on considère que lorsque l'on rebranche le radio-réveil après l'avoir débranché, il retourne toujours dans l'état Radio OFF (sous-état initial). Mais **si l'on souhaite un comportement un peu plus intelligent qui conserve l'état courant de la radio (ON, OFF ou AUTO)** en cas de débranchement intempestif (**avec pile de sauvegarde**), ce modèle n'est plus satisfaisant.

SYSML fournit dans ce cas une construction intéressante: le **pseudo-état History.**

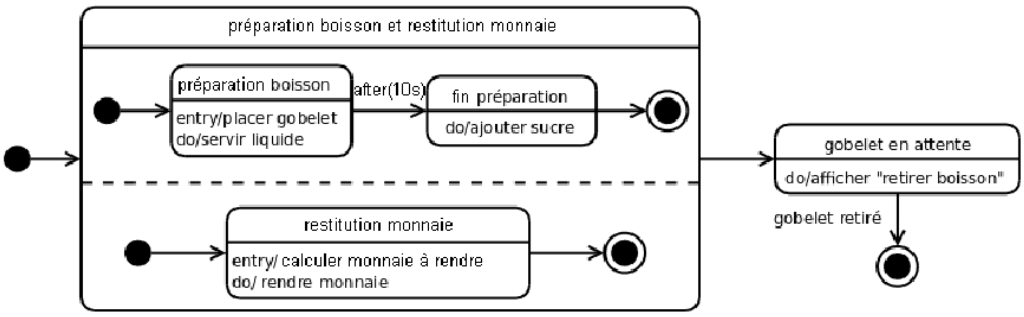
ATTENTION Pseudo-état History

L'activation du pseudo-état History permet à un super-état de se souvenir du dernier sous-état séquentiel qui était actif avant une transition sortante. Une transition vers l'état History rend à nouveau actif le dernier sous-état actif, au lieu de le ramener vers le sous-état initial.

7.4. Concurrence et synchronisation

Dans un état composite, plusieurs graphes d'états peuvent **évoluer** simultanément (en parallèle). On dit qu'il y a **concurrence de plusieurs états.**

Par exemple pour un distributeur de boissons :



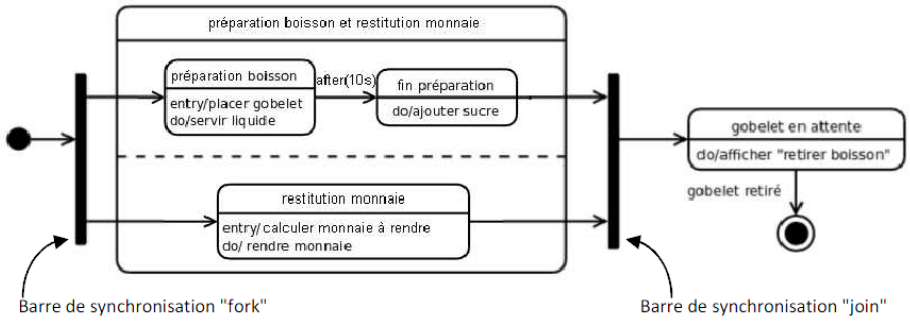
L'état composite est **dit orthogonal** car il comporte plus d'une région, chaque région représentant un flot d'exécution.

• *Graphiquement, dans un état orthogonal, les différentes régions sont séparées par un trait horizontal ou vertical en pointillés allant d'un bord à l'autre de l'état composite.*

• *Chaque région peut posséder un état initial et final. Une transition qui atteint la bordure d'un état composite orthogonal est équivalente à une transition qui atteint les états initiaux de toutes ses régions concurrentes.*

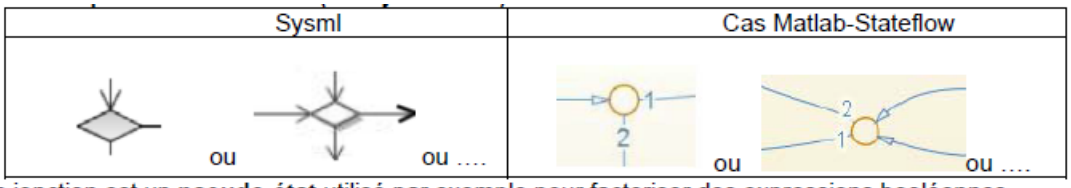
• ***Toutes les régions concurrentes** d'un état composite orthogonal **doivent atteindre leur état final** pour que l'état **composite soit considéré comme terminé**. La synchronisation est alors automatique et la transition de sortie de l'état composite est déclenchée.*

Il est également possible de représenter ce type de comportement au moyen de transitions concurrentes constituées de **barres de synchronisation "Fork" et "Join"** Le graphe ci-dessous est une représentation équivalente à la précédente :



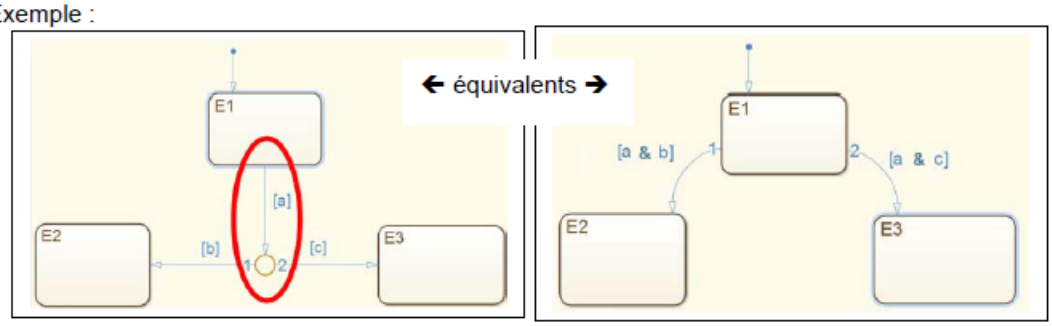
• Les transitions automatiques (sans évènement ou condition de garde) qui partent d'une barre de synchronisation "fork" se déclenchent en même temps. On ne franchit une barre de synchronisation "join" qu'après déclenchement de toutes les transitions qui s'y rattachent.

7.5. Transition avec point de décision ou jonction



la jonction est un **pseudo-état** utilisé par exemple pour factoriser des expressions booléennes associées à des conditions.

Exemple :

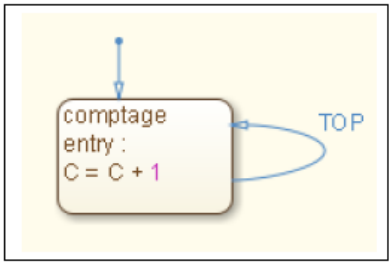


Explication : lorsque la condition [a] est vraie, la jonction devient active, mais E1 est désactivé seulement si [b] ou [c] deviennent vrais.

7.5. Transition réflective

l'état source et l'état pointé est le même ; il est désactivé puis aussitôt réactivé au pas de calcul suivant.

Exemple ci-contre : transition réflexive utilisée en comptage (de TOP) :



SYNTHESE sur la syntaxe: