Logique Séquentielle

Compétences attendues :

- ✓ Interpréter tout ou partie de l'évolution temporelle d'un système séquentiel.
- ✓ Décrire le comportement d'un système séquentiel.

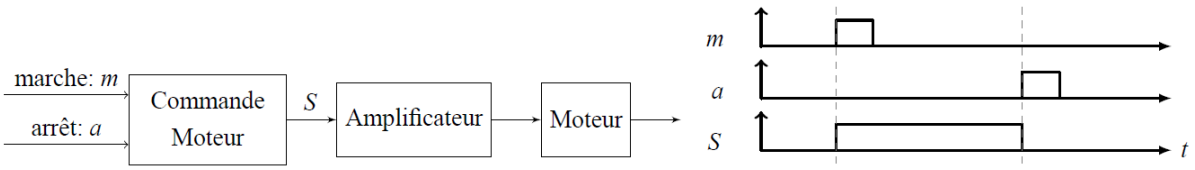
1. Système séquentiel

1.1. Définition

Un système séquentiel, ou à évènements discrets, est un 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.

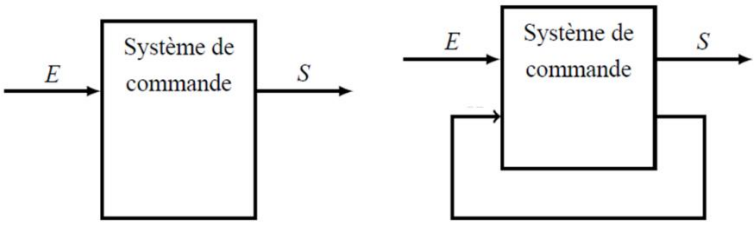
Exemple :

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.



Représentation d'un système séquentiel par un système combinatoire muni d'un retour d'information sur l'état du système

Remarque : Précision, rapidité, stabilité :

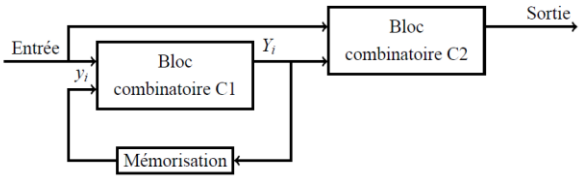
On peut décliner les performances d'un système continu (précision, rapidité et stabilité) pour un système séquentiel, pour cela :

- Le système respecte **précisément** tous les aspects du cahier des charges fonctionnel.
- Le système est **rapide** en réduisant au minimum les temps de cycle. Le calcul du temps de cycle se fait en estimant le temps nécessaire à chaque opération du processus le plus critique.
- Le système est **stable** : la partie commande ne doit jamais aboutir à une situation non prévue et conduisant à un blocage ou à un fonctionnement dégradé.

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. On notera ici Y_i la variable interne (qui sera mémorisée).



2. Diagramme de séquence (*sd*)

Les parties suivantes présentent les différents diagrammes du langage SysML sur l'exemple particulier d'un drone multi-rotors utilisé pour la prise de vue aérienne lors de la réalisation de films ou de reportages.

2.1. Un drone pour le cinéma

Afin de réaliser des prises de vue aériennes en haute définition lors de la réalisation de reportages ou de films, des drones sont fréquemment utilisés. Vous trouverez une vidéo de démonstration ici : http://cine-drone.fr/video.html

Diagramme de contexte

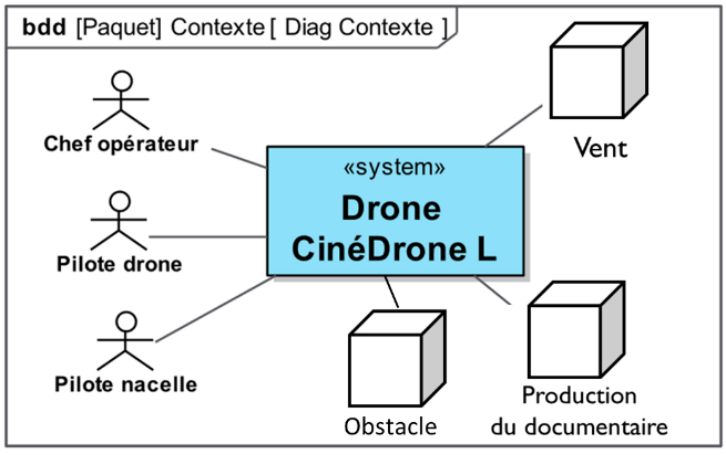


Diagramme des cas d'utilisation

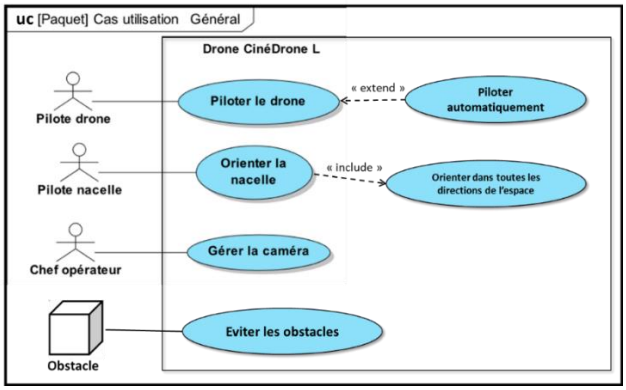
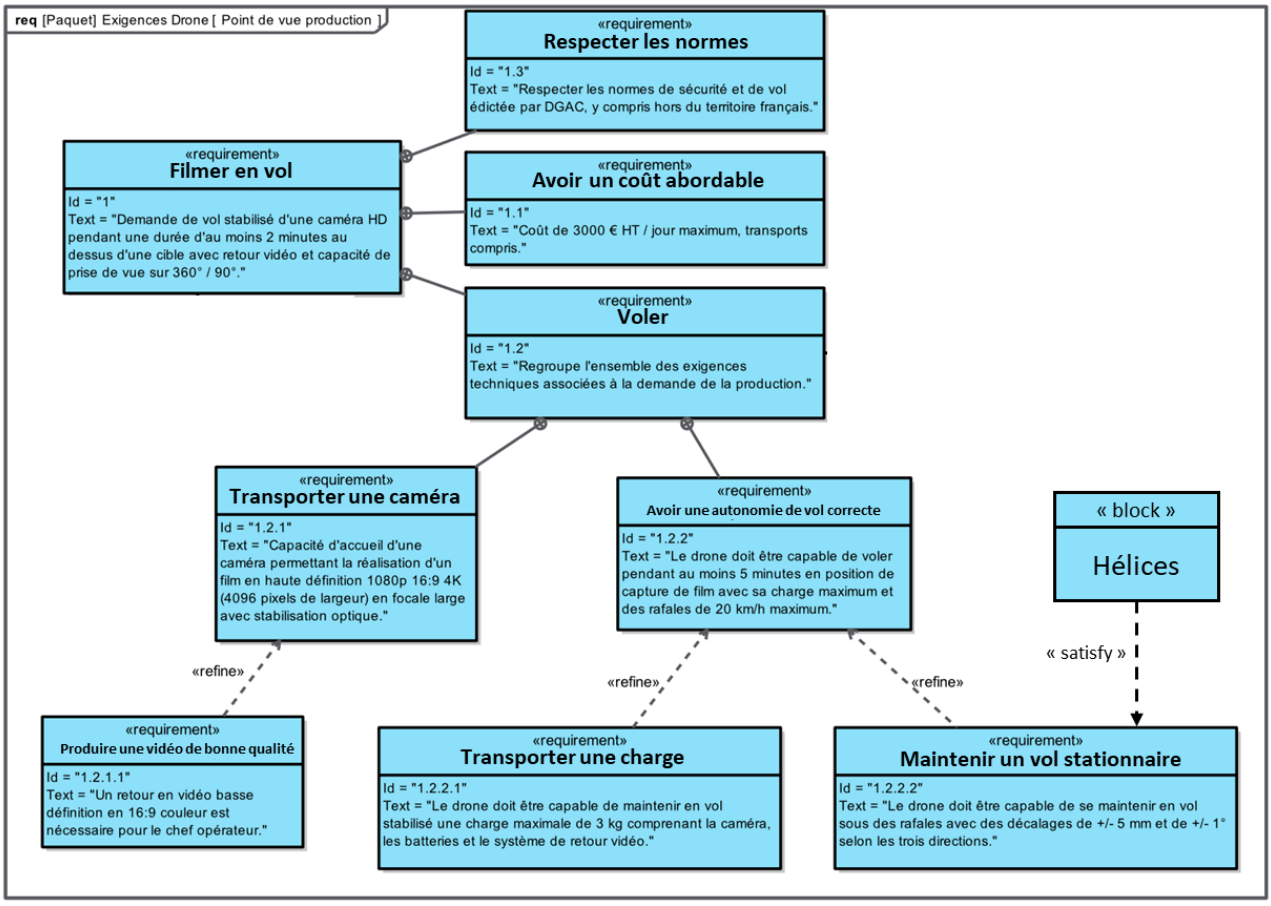


Diagramme d'exigences



Chaîne d'information et de puissance

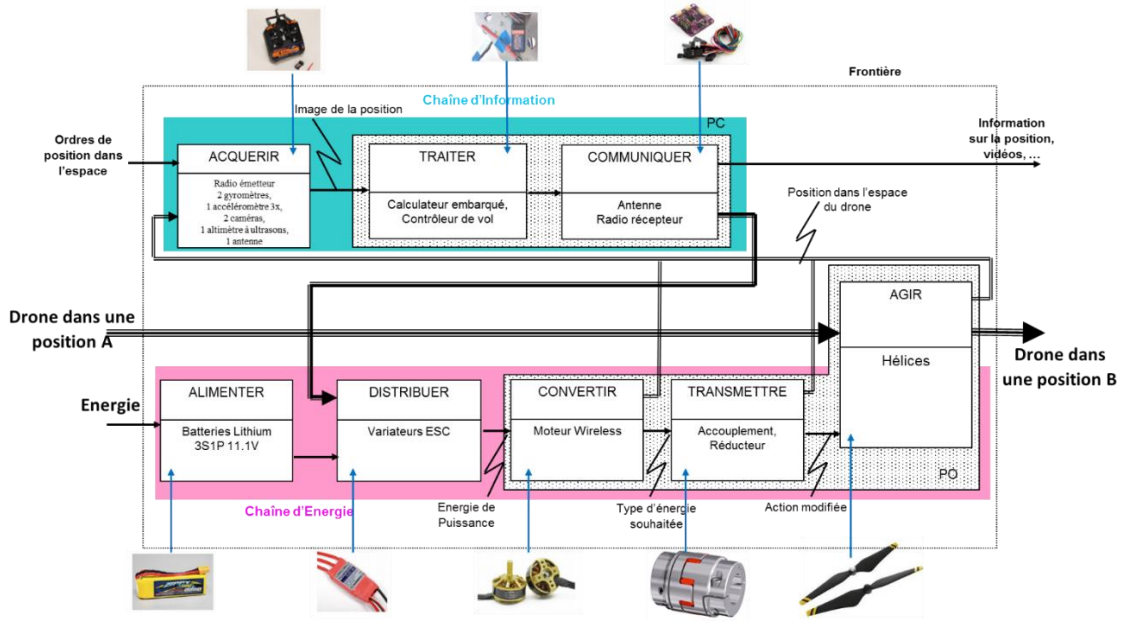


Diagramme de définition de blocs

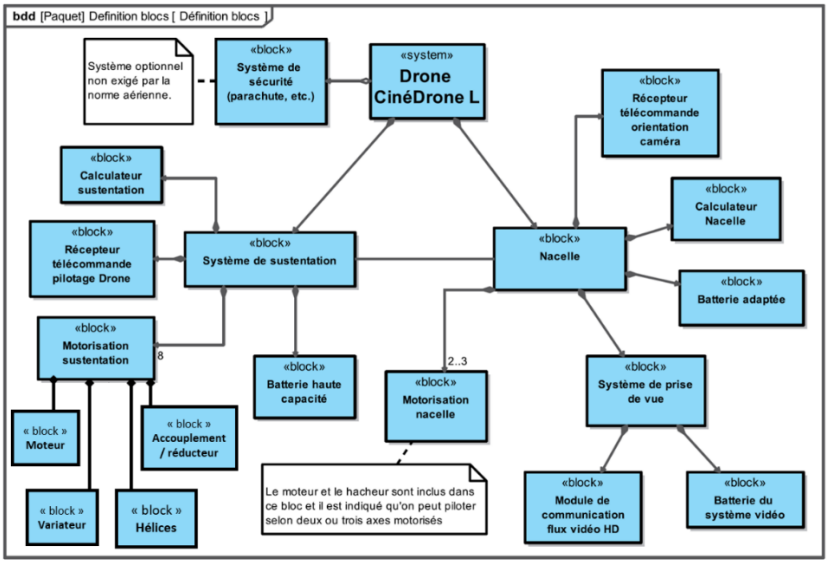
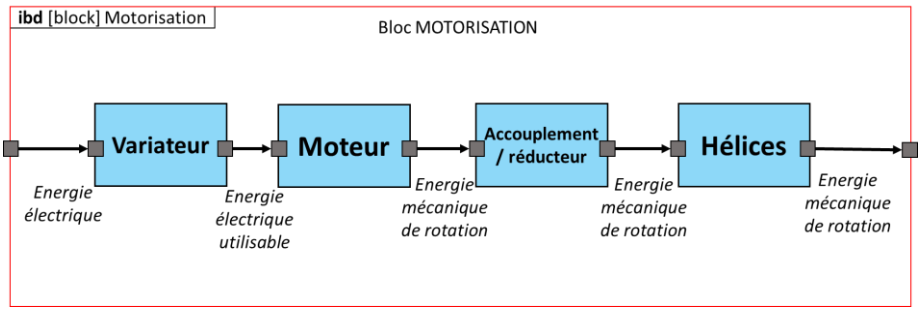


Diagramme de blocs internes



2.1.1. Diagramme des séquences (sd)

Le diagramme de séquence noté **sd** (*SysML Sequence Diagram*) permet de décrire les scénarios correspondant aux cas d'utilisation.

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

« Comment est réalisé ce cas d'utilisation ? »

A tout cas d'utilisation correspond au moins un diagramme de séquence

Les éléments graphiques utilisés dans ce diagramme sont principalement :

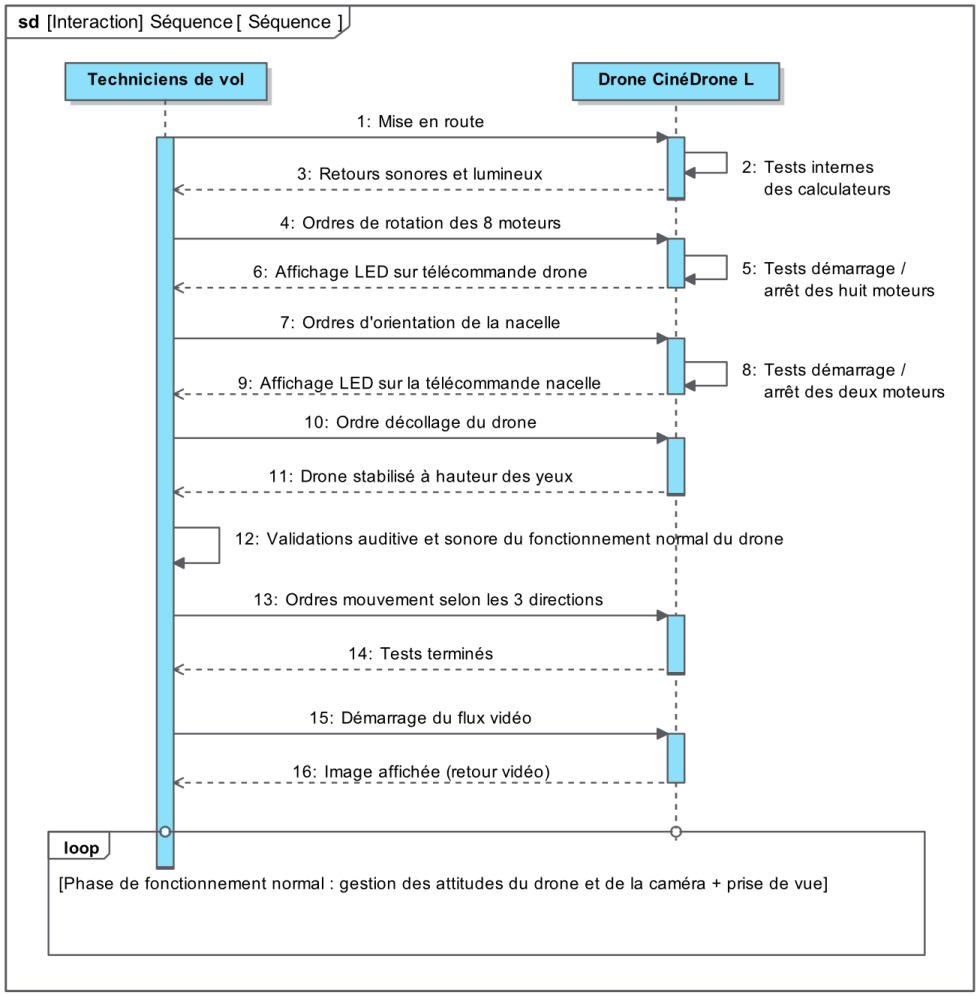
- Des traits verticaux en pointillés appelés « **lignes de vie** » avec l'indication des propriétaires (en général des acteurs, le système et tout ou partie de ses sous-systèmes) sur la partie supérieure. **Le temps se déroule du haut vers le bas, sans échelle particulière**. Une **ligne de vie** est la représentation de l'existence d'un élément participant dans un diagramme de séquence.

- Des flèches horizontales, avec différentes syntaxes, indiquant l'envoi et la réception de « **messages** », notion qui doit être prise au sens large car ils matérialisent les interactions qu'il peut y avoir entre un émetteur et un récepteur. Par exemple, l'appui sur un bouton peut être considéré comme le message envoyé (représentant dans ce cas un événement) et l'affichage d'une image sur un écran comme la réponse à cette sollicitation.

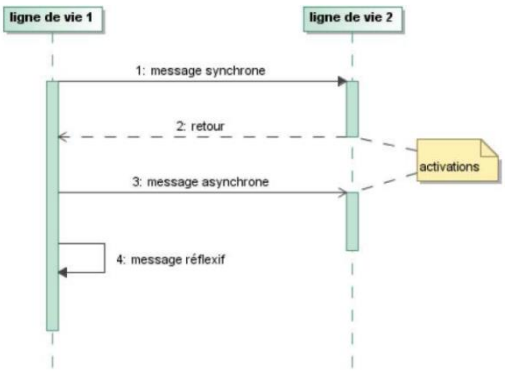
- Les bandes verticales le long d'une ligne de vie représentent des périodes d'« **activation** ». Elles sont optionnelles mais permettent de mieux comprendre la flèche pointillée du message de retour.

Illustration :

Pour le système étudié, il est possible de mettre en place le diagramme présenté ci-dessous qui décrit l'évolution séquentielle des échanges entre les deux techniciens de vol (le pilote du drone et le pilote de la nacelle supportant la caméra) lors de la phase de préparation au vol : cette description répond à une exigence portant sur la norme et ne représente donc pas un scénario complet car seule la phase de test préalable à l'utilisation du système est représentée, ce qui correspond à une partie des cas d'utilisation « Piloter le drone » et « Orienter la nacelle ».



Les différents échanges :



- Message **asynchrone** : pas de réponse attendue
- Messages **synchrones** : réponse attendue

Le message qui suit un message synchrone est forcément la réponse associée

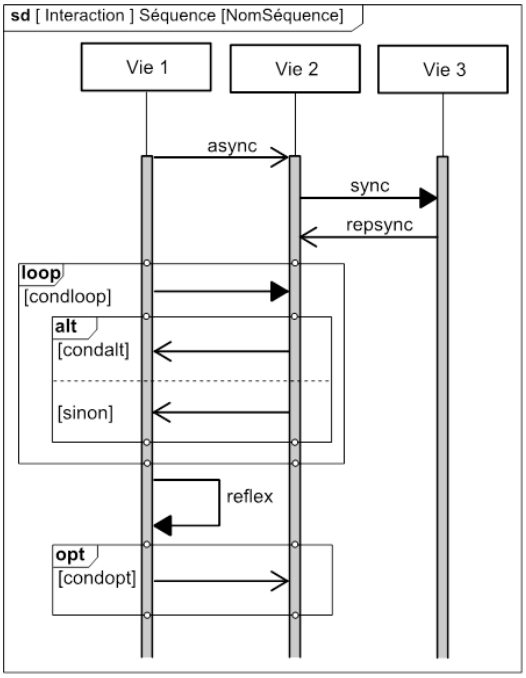
Message **réflexif** : synchrone par essence.

Il est possible d'imbriquer des diagrammes de séquences en utilisant des **fragments combinés**. Un fragment est indiqué par un « rectangle » qui contient un cartouche donnant sa nature ainsi que les **[conditions]** de son déroulement.

Fragments combinés :

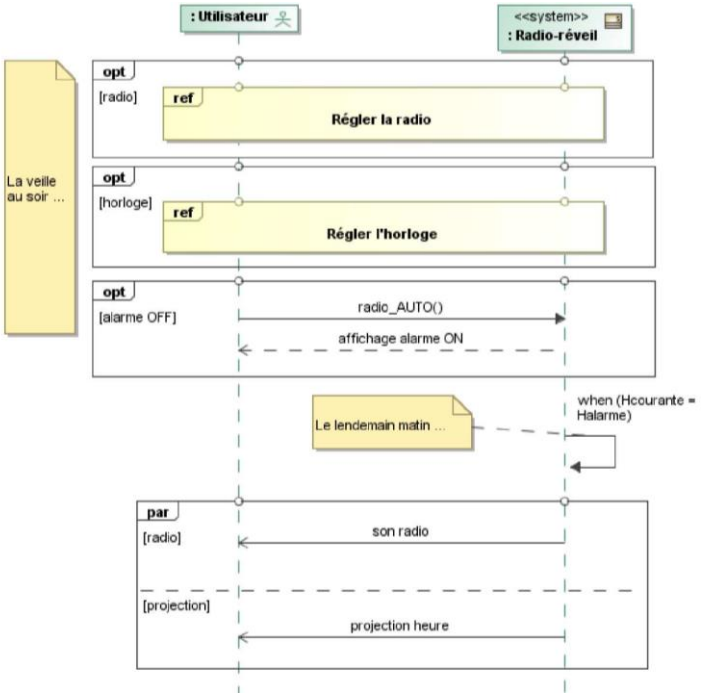
Un fragment combiné peut être divisé en différentes opérandes :

- **loop** (*boucle*) : le fragment s'exécutera plusieurs fois, la **condition de garde** explicitant jusqu'à quand.
- **opt** (*option*) : le fragment ne se déroule que si la condition est vraie.
- **alt** (*alternative*) : seul le fragment dont la condition est vraie se déroule.



Remarque : L'utilisation de la notation « **ref** » dans un « fragment combiné » permet de faire appel à un sous-graphique qui définit l'action associée de manière détaillée, ceci dans un souci de lisibilité.

Exemple : Réglage du Radio Réveil :



2.1.2. Diagramme d'états (stm)

Le diagramme d'états noté **stm** (*SysML State Machine Diagram*) permet de décrire le fonctionnement d'un programme sous forme de machine d'états. **Il montre les différents états pris par le système (ou un sous-système) en fonction des interactions.**

« Comment représenter les différents états du système ? »

Les éléments graphiques utilisés dans ce diagramme sont principalement des rectangles aux coins arrondis représentant les **états**. Les **transitions** entre les états sont représentées avec des **flèches** orientées et un texte les décrit.

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

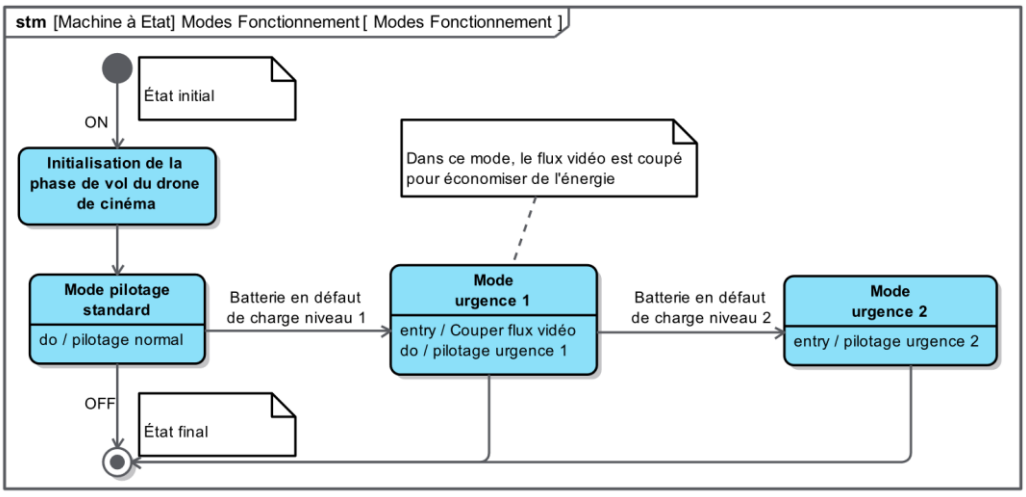
Les transitions sont liées aux évènements et sont réalisées lorsque les évènements associés ont lieu.

Le point de départ est un point noir extérieur aux états ●, le point de fin est un point noir entouré d'un cercle noir ◉.

Remarque : Dans un état, un système peut être en activité ou en attente.

Illustration :

Pour le système étudié, il est possible de mettre en place le diagramme d'états (*stm*) présenté ci-dessous qui permet de représenter les différents modes de fonctionnement du système, ainsi que les évènements qui permettent de passer de l'un à l'autre.

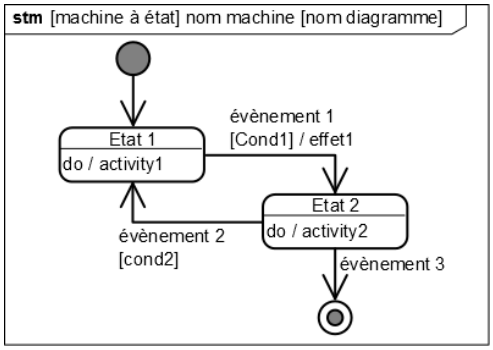


L'initialisation est un mode temporaire dans lequel le système effectue tous ses tests avant de donner la main aux pilotes. La transition est ensuite franchie automatiquement

car il n'y a pas d'évènement particulier à marquer (c'est la fin de l'initialisation). Viennent ensuite le mode de fonctionnement normal puis les modes d'urgence en cas de niveau de charge insuffisant.

Le comportement lors de l'entrée et de la sortie de chaque état est décrit : par exemple, l'entrée dans le mode urgence 1 (évènement interne *entry*) provoque automatiquement la coupure du flux vidéo pour économiser l'énergie et un mode de pilotage dégradé (pilotage à vue) où le drone ne peut que redescendre.

Diagrammes d'état / transition :



Le diagramme de **machine d'état** décrit les **transitions** entre les **états** (et les actions) que le système (ou ses parties) réalise en réponse **aux événements**.

Les machines d'état ne sont pas des algorithmes.

Un état représente une situation **dans la vie d'un bloc** durant laquelle :

- Il satisfait une certaine condition,
- Il exécute une certaine activité,
- Il attend un certain événement.

La vie d'un bloc est donc une succession d'états de durées finies.

Une transition décrit la réaction d'un bloc lorsqu'un évènement se produit (souvent un changement d'état). Une transition n'est examinée que si l'état qui la précède est actif.

En général une transition possède :

- Un évènement déclencheur,
- Une condition de garde,
- Un effet.

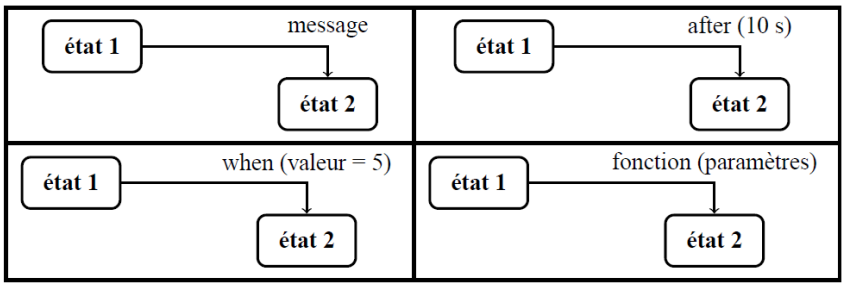
Il existe quatre types d'événements associés à une transition :

• **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 (opération) du bloc a été effectuée. Un retour est attendu. Des arguments (paramètres) de fonction peuvent être nécessaires.

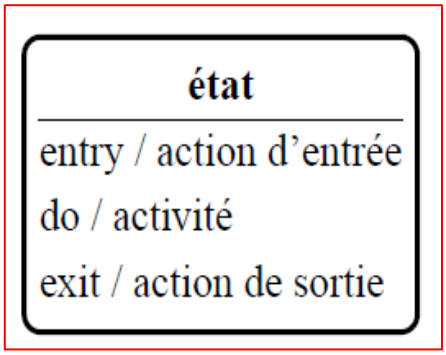


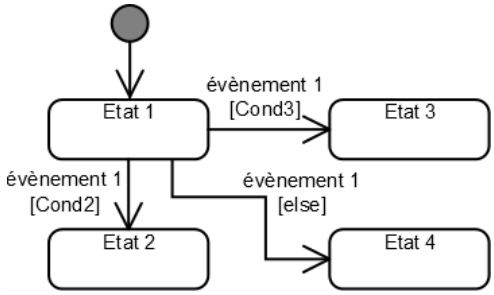
L'entrée dans l'état déclenche l'activité associée à *entry*.

Lorsque cette activité est terminée, celle associée à *do* commence.

Lors de la sortie, l'activité associée à *do* s'arrête, celle associée à *exit* commence (désactivation avant de sortir par exemple).

Il est évidemment possible de n'indiquer que l'activité associée à *do* le cas échéant (pas d'action en entrée ou en sortie).





Un changement d'état est provoqué par le franchissement **instantané** d'une transition entre un état source et un état cible.

La condition de franchissement est indiquée sur le lien : **événement [garde] / effet.**

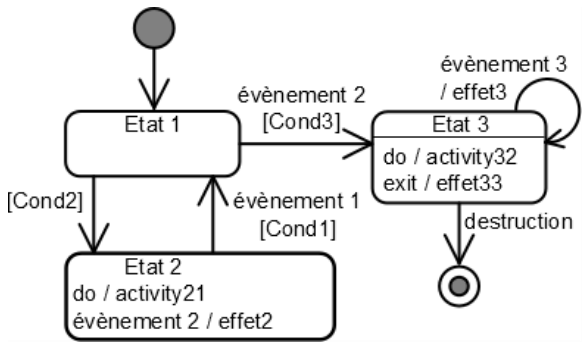
La transition n'est franchie que si l'évènement ET la condition sont vrais.

Attention : Il faut veiller à ce que les transitions issues d'un même état source soient « exclusives » : il ne doit y avoir qu'une seule possibilité d'évolution à partir d'un état.

On rappelle qu'une condition de garde est une expression **booléenne** faisant intervenir des **entrées** et/ou des **variables internes**.

Il est possible d'utiliser les notations non booléennes de front ↑↓.

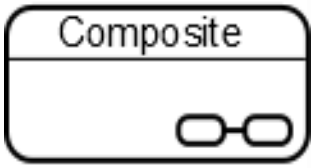
L'effet associé à une transition est effectué lorsque la transition est franchie.



L'évènement 3 est une transition réflexive (dite aussi propre) et elle provoque une sortie de l'état 3 et un retour immédiat dans celui-ci.

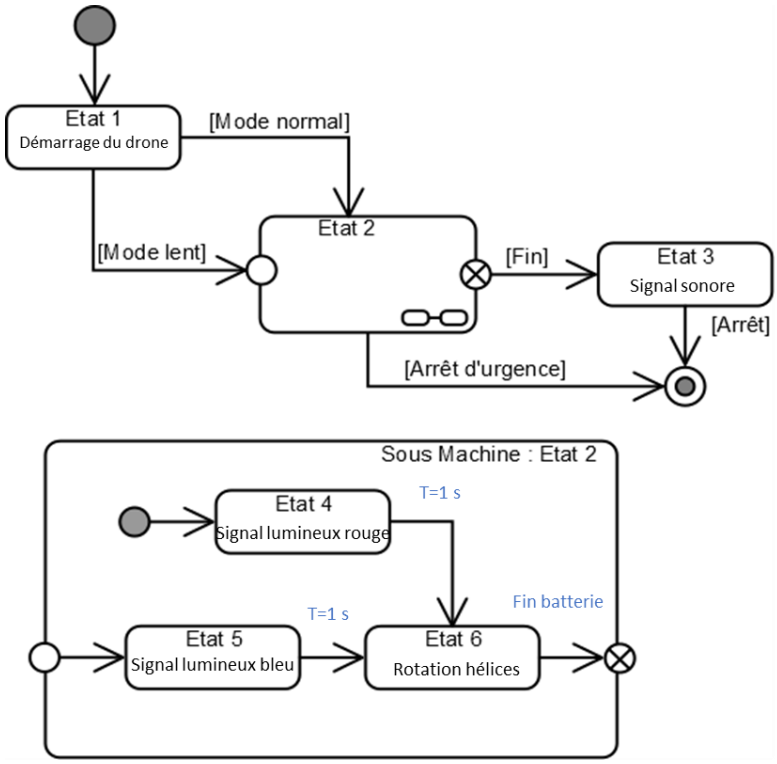
Ce qui a pour effet l'interruption de l'activité32, la réalisation de l'effet33, celle de l'effet3 et enfin le redémarrage de l'activité32.

Etat composite (super-état) :

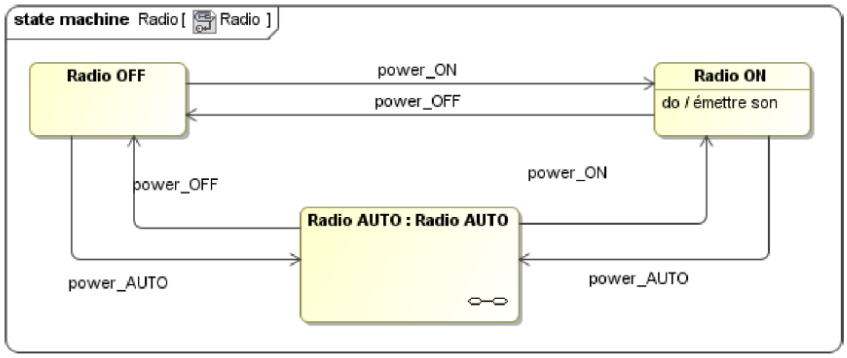


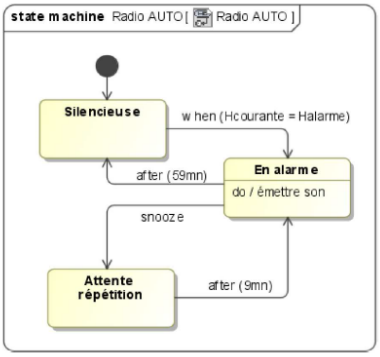
Un état composite (lunettes) peut être vu comme un macro- état. Il décrit les évolutions internes avec un autre diagramme d'état (il contient une machine d'état).

Illustration d'un état composite associé au fonctionnement du drone.

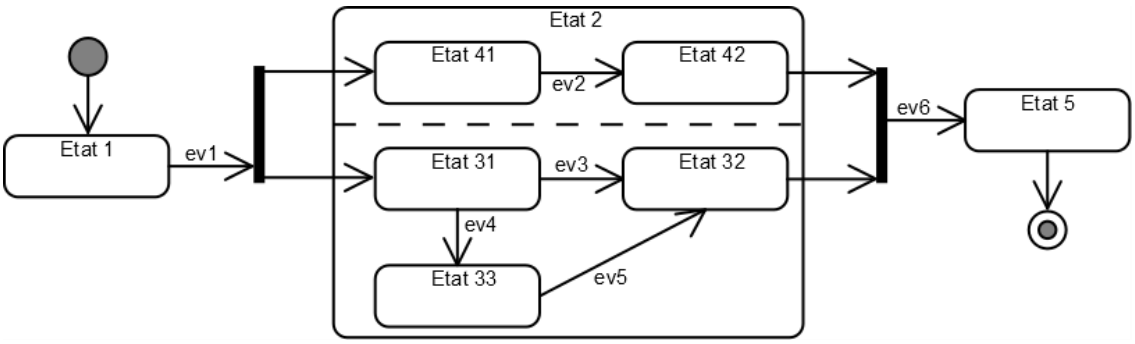


Exemple : Radio Réveil

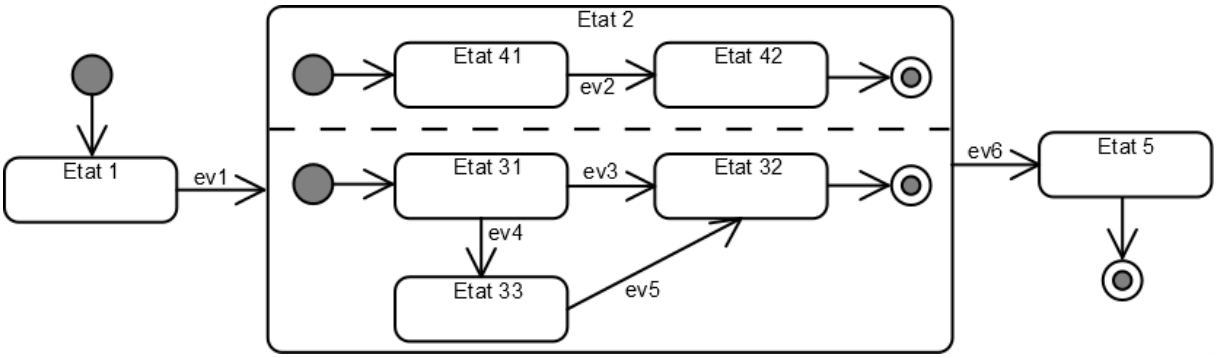




Divergence et convergence de séquences parallèles :



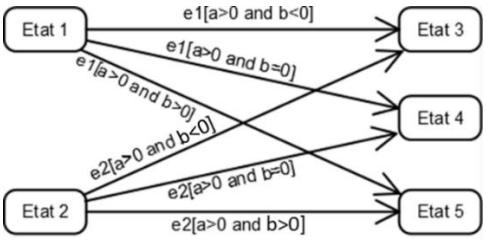
Transitions dans un état composite :

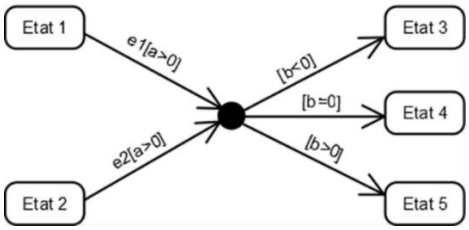


Une transition qui atteint la bordure d'un état composite (Etat2) est équivalente à une transition qui atteint toutes les étapes initiales.

Remarque : Les deux diagrammes d'états (divergence/convergence et composite) ci-dessus sont identiques.

Points de jonction :





Les points de jonctions sont des pseudo états, ils permettent de simplifier certains graphes.

2.1.3. Diagramme d'activités (act)

Le diagramme d'activité noté **act** (*SysML Activity Diagram*) permet de représenter le déroulement d'un processus sous la forme d'une activité correspondant à une décomposition séquentielle d'actions, aussi appelées tâches.

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

Les éléments graphiques utilisés dans ce diagramme ressemblent fortement à ceux du diagramme d'états : chaque tâche est représentée par un rectangle aux coins arrondis et est ensuite reliée à une autre tâche par des transitions représentées par de simples flèches (voir exemple). Lorsqu'une tâche est terminée, la suivante commence.

Remarque : Si le diagramme d'activités ressemble au diagramme d'états, leurs domaines d'utilisation sont sensiblement différents et il est donc fondamental de ne pas les confondre.

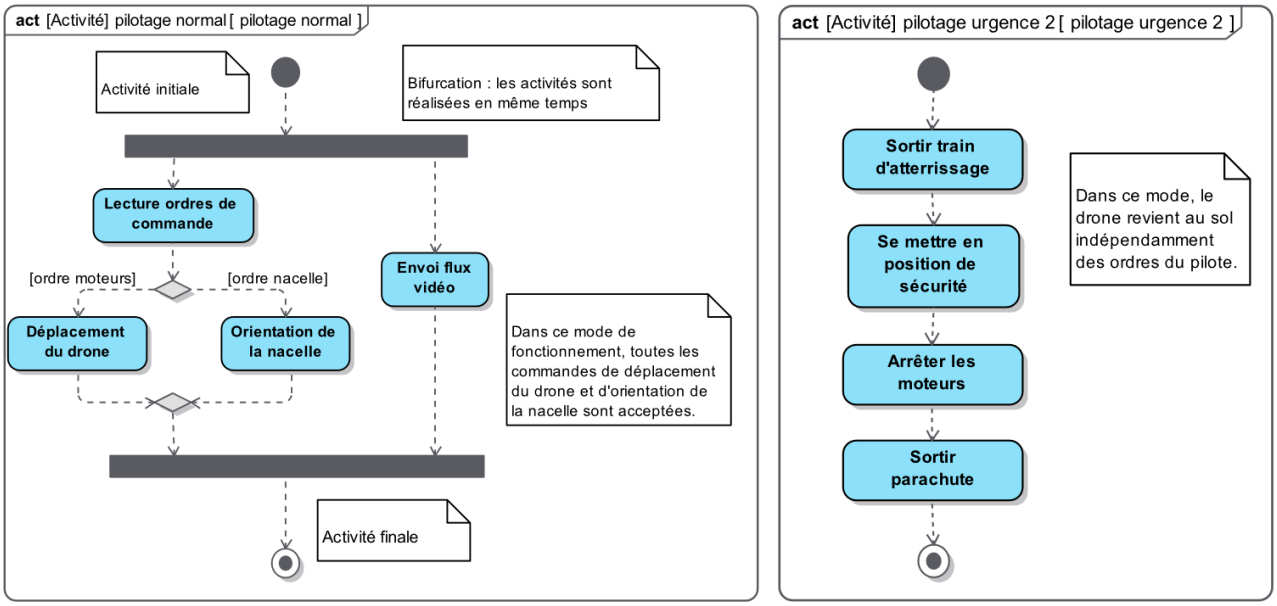
Remarque : Le diagramme d'activités n'est pas explicitement au programme des CPGE. Son utilisation est cependant pratique pour décrire la structure des programmes implantés dans les microcontrôleurs, notion qui elle est explicitement au programme.

Illustration :

Dans le diagramme d'états précédent, ce qui se passe dans chaque mode n'est pas décrit : pour ce faire, il est possible de choisir des nouvelles machines d'états ou des activités.

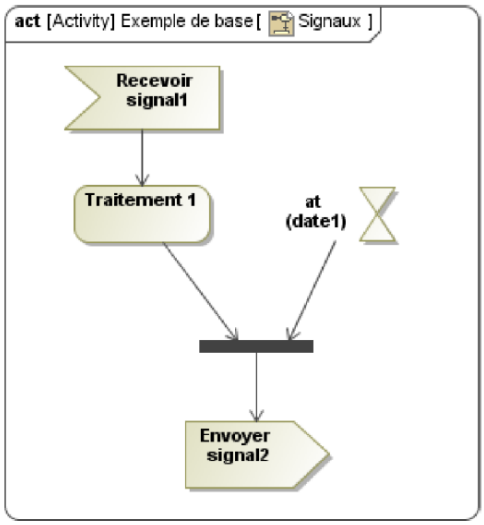
Par choix, la description de chaque mode a été réalisée uniquement par des diagrammes d'activités :

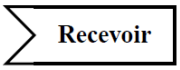
- Le mode normal correspond à un pilotage complet du drone avec en parallèle l'envoi du flux vidéo (les deux barres noires marquent le début et la fin de l'envoi des deux flux).
- Dans le mode d'urgence 2, la procédure pour atterrir automatique est décrite, la note précisant les limitations de ce mode.

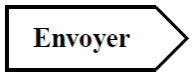


Signaux et évènements :

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 :** 

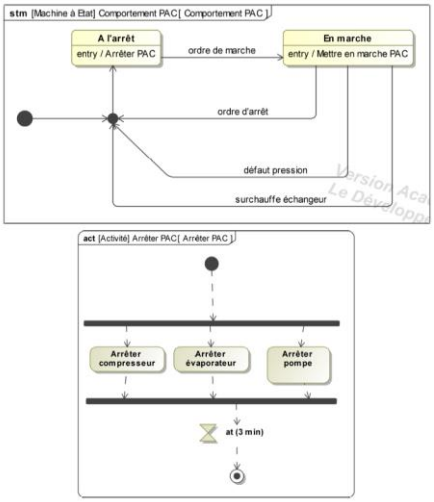
- **accept time event :** 

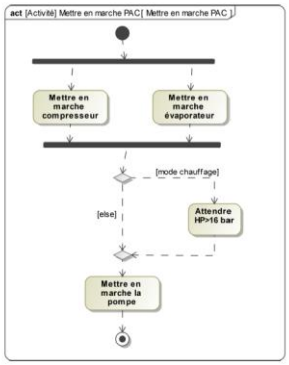
2.1.4. Différence entre le diagramme d'activités et le diagramme d'états

Les deux types de diagrammes sont différents :

- Le diagramme d'états montre les évènements 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 évènement 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.
- 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 évènements 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.

Exemples :

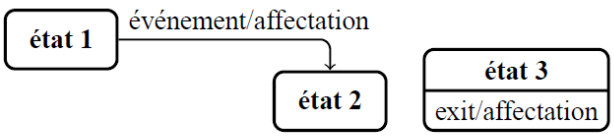




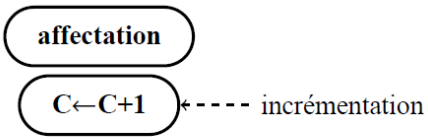
3. Les structures algorithmiques de base

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.



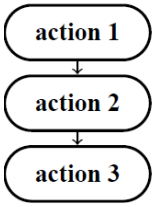
formalisme du diagramme d'états



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.



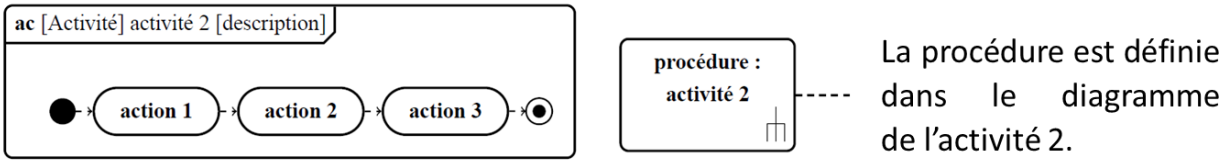
3.3. Fonctions et procédures

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

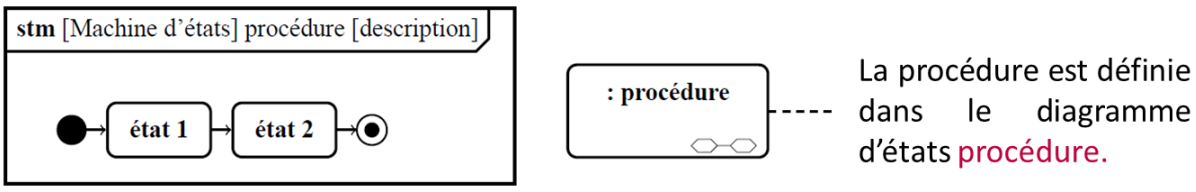
• d'une part, de scinder une problématique générale en plusieurs problématiques élémentaires,

• d'autre part, de pouvoir réutiliser des sous-programmes réalisant des tâches élémentaires.

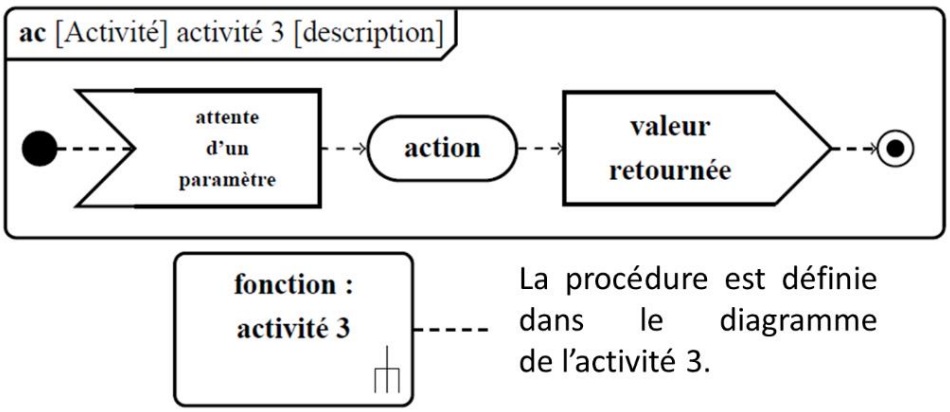
Une procédure comporte une succession d'instructions mais ne renvoie rien.



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



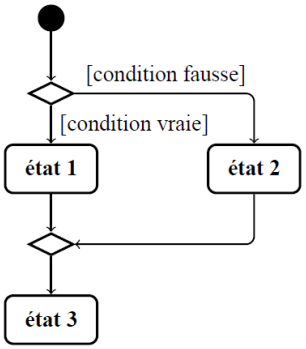
A la fin de l'exécution d'une fonction, il y a le retour d'une valeur, d'une liste, d'un objet, etc.



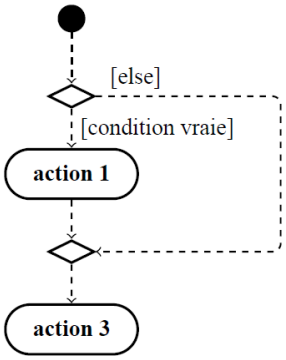
3.4. Les structures conditionnelles et bouclées

Structure alternative avec saut

Si …… , alors faire …… , sinon faire ……



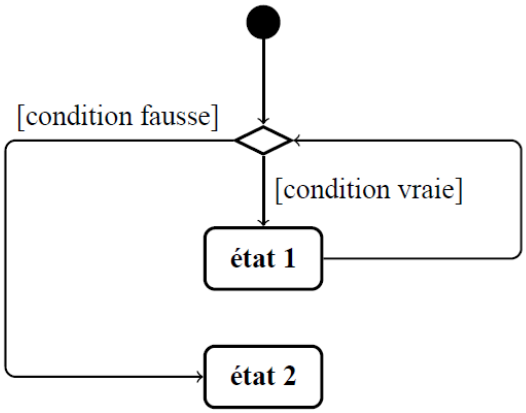
Formalisme du diagramme d'états



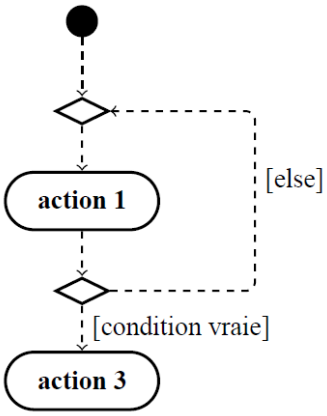
Formalisme du diagramme d'activités

Structures répétitives (itératives)

Tant que condition vraie, faire … / Répéter … jusqu'à condition vraie

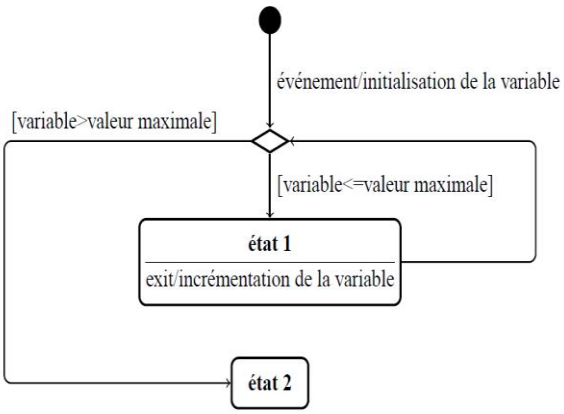


Formalisme du diagramme d'états

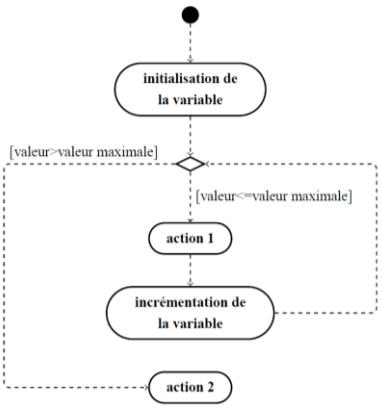


Formalisme du diagramme d'activités

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



Formalisme du diagramme d'états



Formalisme du diagramme d'activités

4. Algorigramme et pseudo-code

Définition : Un algorithme est une série d'actes ou d'opérations élémentaires qu'il faut exécuter en séquence pour accomplir une tâche quelconque, en suivant un enchaînement strict.

Remarque : Lorsqu'il sera demandé d'élaborer un algorithme, la méthode pour atteindre cet objectif sera de rédiger en français la succession des opérations élémentaires (phases courtes et précises) puis de passer à une écriture conventionnelle appelée pseudo-code.

4.1. Le pseudo-code

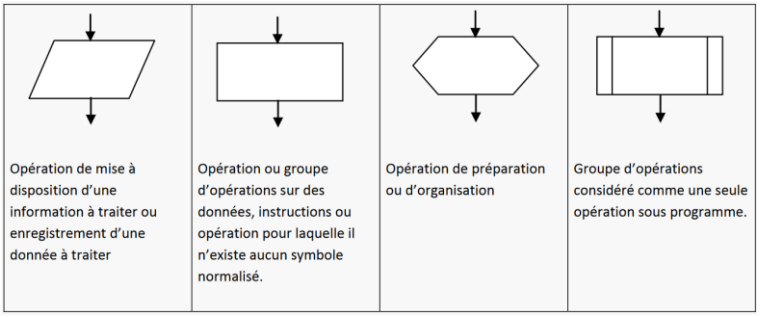
Ce mode de représentation consiste à exprimer en langage naturel, mais selon une disposition particulière et des mots choisis, les différentes opérations constituant l'algorithme, conformément au code donné dans le tableau suivant :

Mots et symboles du pseudo-code	Opérations réalisées
Début	Début de l'algorithme, permet de le nommer
Fin	Fin de l'algorithme
Faire	Exécution d'une opération
Entrer	Acquisition ou chargement d'une donnée
Sortir	Edition ou sauvegarde d'un résultat
←	Affectation d'une valeur à une variable
Symboles d'opérateur	Opérations arithmétiques ou logiques
Aller à	Branchement inconditionnel
Si…alors…[sinon]	Branchement inconditionnel
Selon cas…[autrement]	Branchement conditionnel généralisé
Itérer…sortir si…	} Répétition conditionnelle
Tant que…faire…	
Répéter…jusqu'à ce que…	
Pour…de…à…	Répétition contrôlée

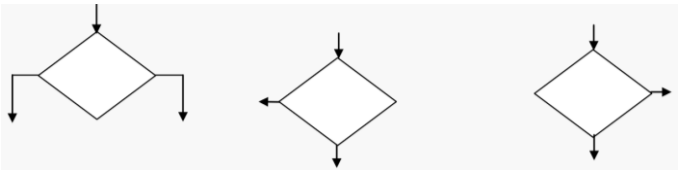
4.2. Algorigramme

Définition : L'algorigramme est la représentation graphique de l'algorithme, il permet de représenter chaque opération élémentaire au moyen d'un symbole graphique normalisé.

Symboles de traitement



Symboles de test logique

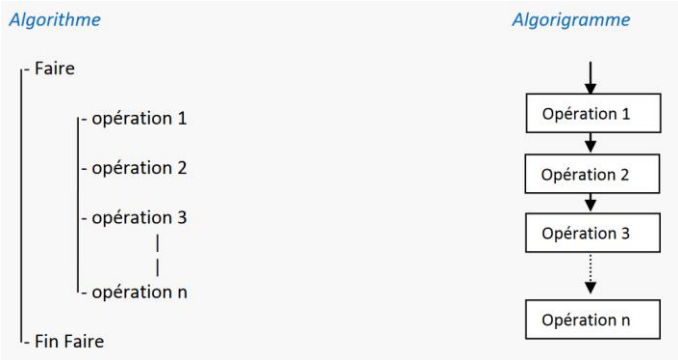


4.3. Règles de construction

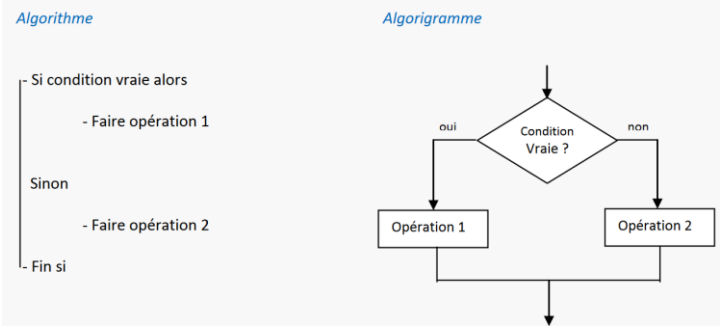
- Centrer l'algorigramme sur une feuille.
- Construire l'algorigramme afin que sa lecture s'effectue verticalement.
- Les lignes de liaison entre symboles ne doivent pas en principe se couper (utiliser un symbole de renvoi O).
- Une ligne de liaison doit toujours arriver sur le haut et au centre d'un symbole.
- Les commentaires sont à placer de préférence à droite et les renvois de branchement à gauche.

4.4. Structures classiques

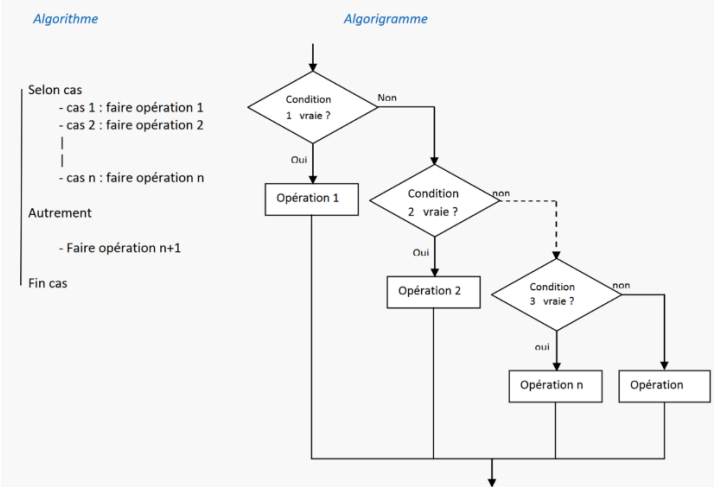
Structure séquentielle ou linéaire



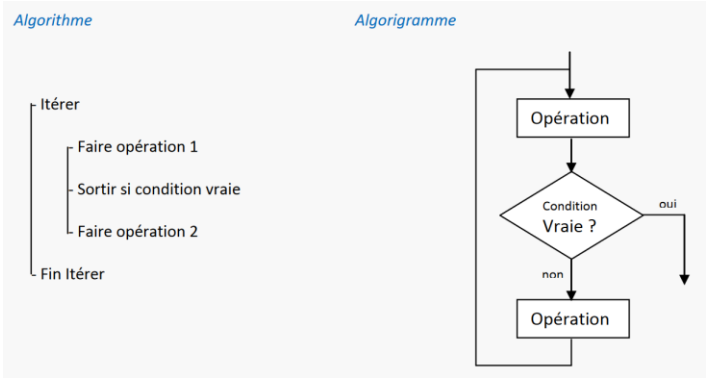
Structure conditionnelle (sélection simple)



Structure conditionnelle (sélection multiple)



La structure itérative ou de répétition



Cas d'une répétition contrôlée