CI 05 - Décrire le comportement des systèmes à événements discrets

Sommaire

1. Systèmes à événements discrets (SED)

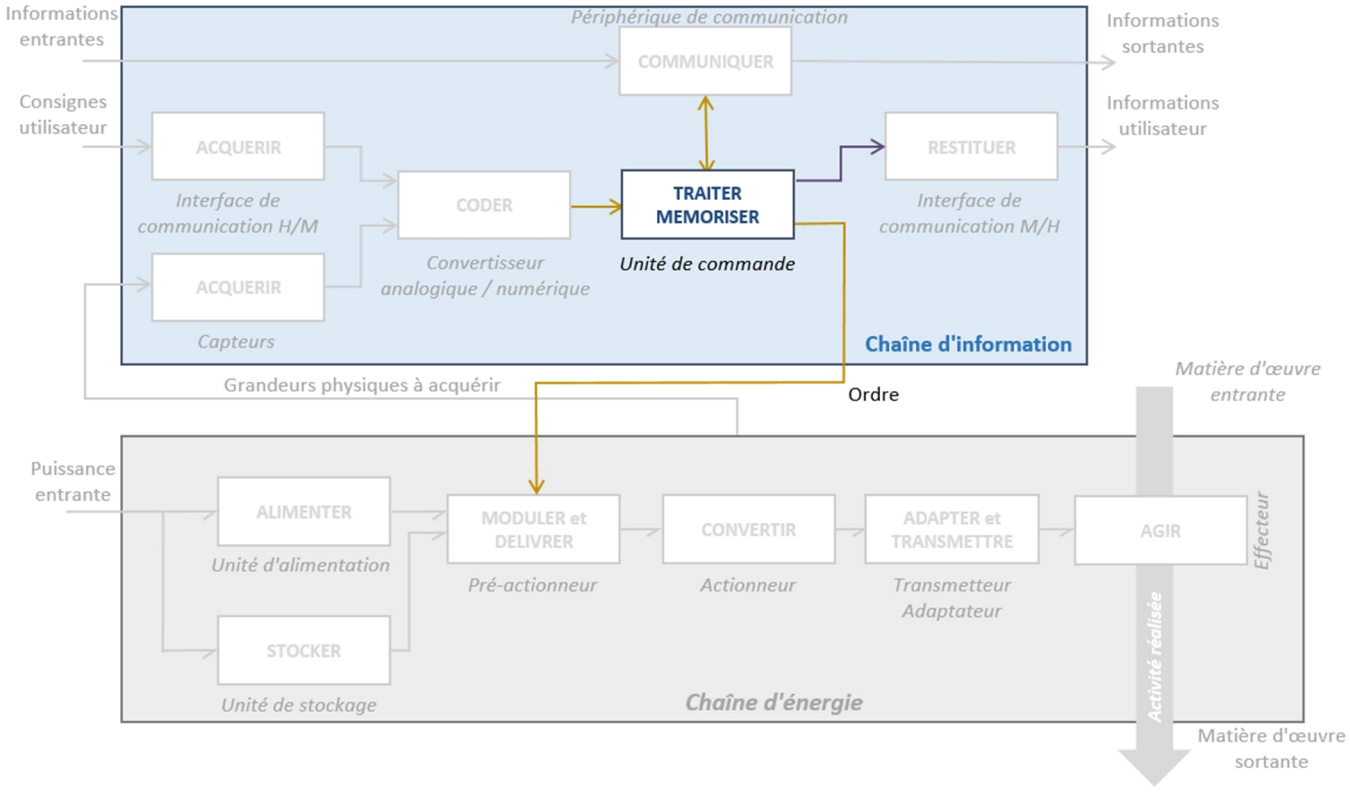
Les systèmes à événements discrets (SED) se définissent par opposition aux systèmes continus (SC) dont l'évolution est continue dans le temps et peut être décrite par des équations différentielles.
Dans un SED, le passage d'un *état* à un autre est déclenché par des *événements* ponctuels.
Contrairement aux systèmes continus où les informations traitées sont de nature analogique, les systèmes à événements discrets manipulent des informations logiques ou numériques.

Les SED sont le plus souvent séquentiels, c'est-à-dire que la ou les sorties dépendent de la combinaison des entrées et de l'état précédent des sorties.

Ainsi :
- une même cause (même combinaison des entrées) peut produire des effets différents ;
- le temps peut être une cause déclenchante ;
- l'effet peut persister si la cause disparaît.

Télécommande Smart Touch de Samsung

2. Modéliser la structure d'un programme



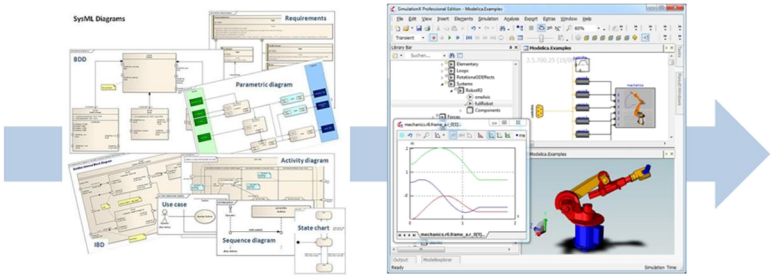
Unité de commande : fonction, entrée et sorties

L'unité de commande de ces systèmes à événements discrets est très souvent réalisée à partir de circuits logiques et de calculateurs. Elle est programmée à partir de l'analyse d'un cahier des charges avec comme objectif, d'aboutir à un fonctionnement du système conforme aux attentes de ses utilisateurs.
Le risque d'une analyse erronée ou incomplète des exigences de l'utilisateur est alors d'aboutir à un programme qui ne répond pas complétement à ce qui est attendu, ou bien à la nécessité de l'ajuster par ajouts successifs de fragments de code correctifs non cohérents qui conduisent à l'impossibilité de le maintenir et de le faire évoluer en raison de son manque de cohérence et de lisibilité.

Il se révèle alors plus efficace techniquement et économiquement de modéliser la structure du programme avant tout codage et tout choix de langage.

Cette démarche structurée s'inscrit dans un contexte plus large d'*ingénierie système*.

Elle peut être mise en œuvre via le langage SysML et plus particulièrement grâce à l'utilisation des diagrammes d'état (*state machine diagramme* ou *stm*) et du diagramme de séquence (*sequence diagram*, ou *seq*).



Ingénierie système : diagramme du langage SysML et logiciels de simulation associés

Notons qu'il Il existe d'autres outils pour modéliser le programme d'un système à évènement discret ou décrire son comportement séquentiel, et que pour un même outil il existe souvent de nombreuses solutions.

3. Diagramme d'état : *state machine diagram* ou stm

3.1. Présentation

En langage SysML, un diagramme d'état (stm) est nécessairement associé à un bloc du diagramme de définitions de blocs (bdd) ou du diagramme de blocs internes (ibd). Ce bloc peut être le système, un sous-système ou un composant.

Le diagramme d'état décrit les différents états pris par le bloc ainsi que les transitions possibles entre ces différents états.

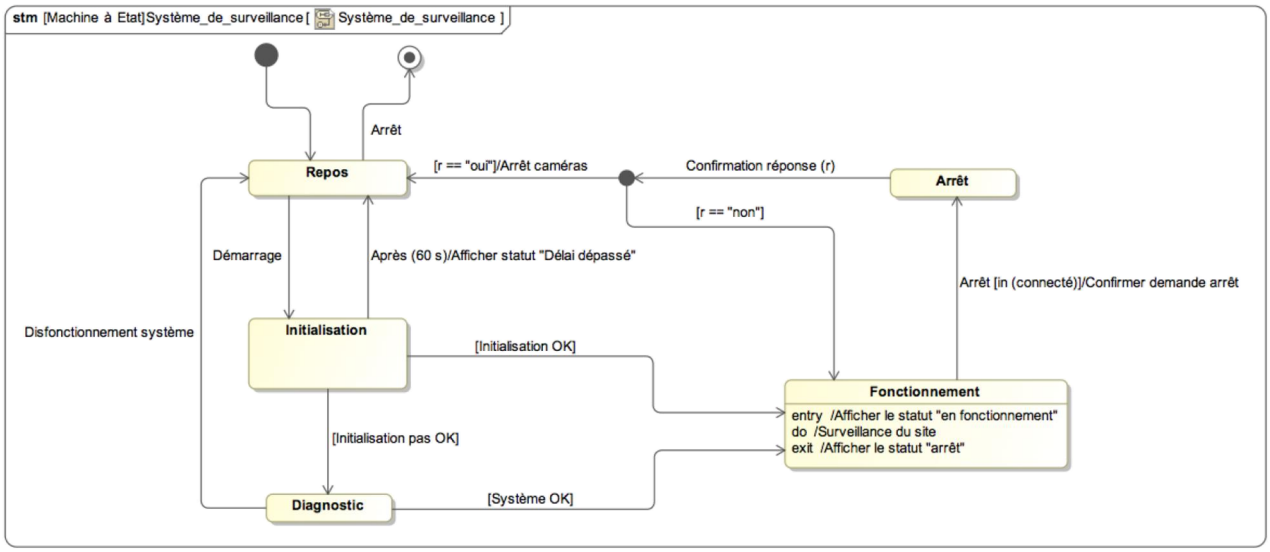
Chaque bloc d'un bdd ou d'un ibd ne conduit pas nécessairement à un diagramme d'état. En effet, certains blocs dont le comportement est statique ne requièrent pas nécessairement un diagramme d'état. Seuls ceux qui ont un comportement dynamique complexe nécessiteront une description poussée par diagramme d'état.

Le diagramme ci-dessous décrit le fonctionnement d'un système de vidéo-surveillance.
On y trouve :

- 5 états : *Repos* , *Initialisation*, *Diagnostic*, *Arrêt* et *Fonctionnement* ;
- 1 pseudo-état initial : ●→
- 1 pseudo-état final : →◉
- des transitions entre les états, représentées par des flèches, et qui précisent sous quelle condition le système passe d'un état à un autre.

Système de vidéo-surveillance

Dans cet exemple, la représentation du comportement est fonctionnelle. Aucune information technique sur la manière dont sont transmises les informations, ni sur la façon dont sont réalisées les activités, sont précisées.



3.2. Démarche de modélisation du comportement séquentiel d'un système par un graphe d'état

Une bonne modélisation du comportement séquentiel d'un système, en vue de programmer son unité de commande, nécessite de respecter la démarche suivante :

Étape 1
- Définir la frontière du système et recenser les variables d'entrée (consignes de l'utilisateur et grandeurs physiques acquises par les capteurs) et de sortie (messages à destinations de l'utilisateur ou d'autres systèmes et ordres vers les préctionnneurs).

Étape 2
- Recenser, nommer et tracer les différents états dans lesquels peut se retrouver le système lors de son utilisation. Identifier les activités associées à chaque état.

Étape 3
- Identifier et tracer les transitions possibles entre les états en fonction du comportement séquentiel souhaité ou observé.

Étape 4
- Définir les conditions et événements déclencheurs associés à chaque transition et qui autorisent son franchissement.

3.3. État et activités associées

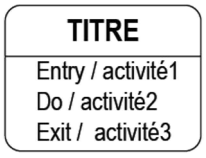
Un état modélise une phase du fonctionnement du système.
Pendant cette période, l'état est dit ***actif*** et le système accomplit ***une simple activité, une séquence d'activités*** ou ***est en attente*.**
En dehors de cette période, l'état est dit ***inactif***.
Par définition, il n'y a qu'un seul état actif à chaque instant.

3.3.1. États intermédiaires

Les états intermédiaires représentent les étapes de la vie du système de son démarrage jusqu'à son arrêt. Représentés graphiquement par des rectangles aux coins arrondis, ils possèdent un titre, ***unique dans le diagramme***, informant de l'état du système. Un état sans titre est appelé état anonyme.

Le lancement des activités à l'intérieur de l'état actif est organisé selon des mots réservés :

entry	L'entrée dans l'état déclenche l'exécution de l'activité associée. C'est impérativement une ***activité instantanée***, c'est-à-dire qu'elle ***ne peut pas être interrompue*.**
do	Activité(s) à exécuter dans l'ordre de leur écriture, à partir de l'instant où l'activité associée à *entry* est terminée. Les activités associées au *do* sont ***interruptibles***.
exit	La sortie de l'état qui va entrainer sa désactivation, entraine l'interruption des activités associées à *do* et l'exécution de celle associée à *exit.* C'est impérativement une ***activité instantanée***, c'est-à-dire qu'elle ***ne peut pas être interrompue*.**



Remarques :

- si aucun mot réservé n'est utilisé, cela correspond à un *do* ;
- les trois comportements *entry*, *do* et *exit* ne peuvent être ***utilisés qu'une seule fois par état***, mais il est également possible de n'en utiliser qu'une partie (seulement *entry* par exemple) ;
- un ***état « vide »*** (sans activité) indique un ***état d'attente*** ;
- il est possible de placer dans un état des activités associées à un événement ou à une garde (voir 3.4.1 et 3.4.2)

3.3.2. Pseudos-états

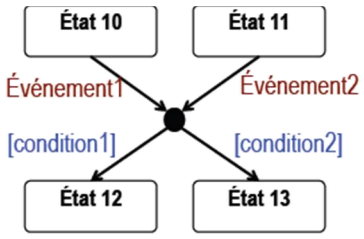
Le terme *pseudo-état* est utilisé pour indiquer qu'il n'y ne peut pas y avoir d'activités associées, ce sont essentiellement des éléments de liaisons.

Le ***pseudo-état initial*** ● est unique et obligatoire. Il est activé au lancement de la machine à états et marque le début de l'exécution du diagramme d'état. Il n'a aucune transition entrante.

Le ***pseudo-état final*** ◉ est optionnel. Il signe la fin de l'exécution du diagramme d'état. Lorsque cet état est activé, la machine à états s'arrête. Il n'a donc aucune transition sortante. Il peut y en avoir plusieurs car différents scénarios peuvent être possibles pour mettre fin à un comportement.

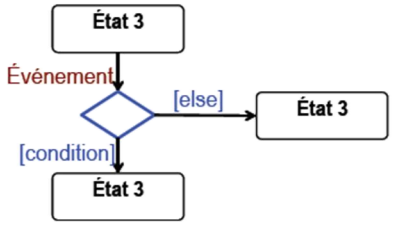
Le ***pseudo-état de jonction*** ● est utilisé pour regrouper (« factoriser ») des conditions de franchissement de transitions, en particulier des gardes communes à un événement. Cela permet de partager des segments de transition et d'aboutir à une notation plus lisible des chemins alternatifs. L'évaluation des conditions de garde en aval du pseudo-état est réalisée avant qu'il ne soit atteint.

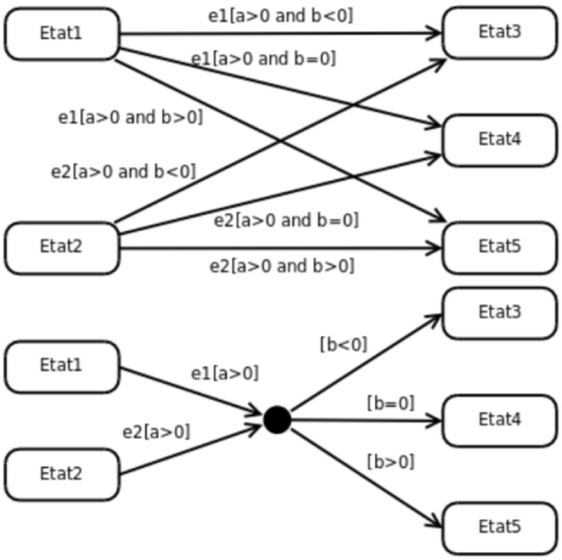
Par exemple, ici : on passe de l'état 10 à l'état 13 si l'événement 1 apparait et que la condition 2 est vrai.

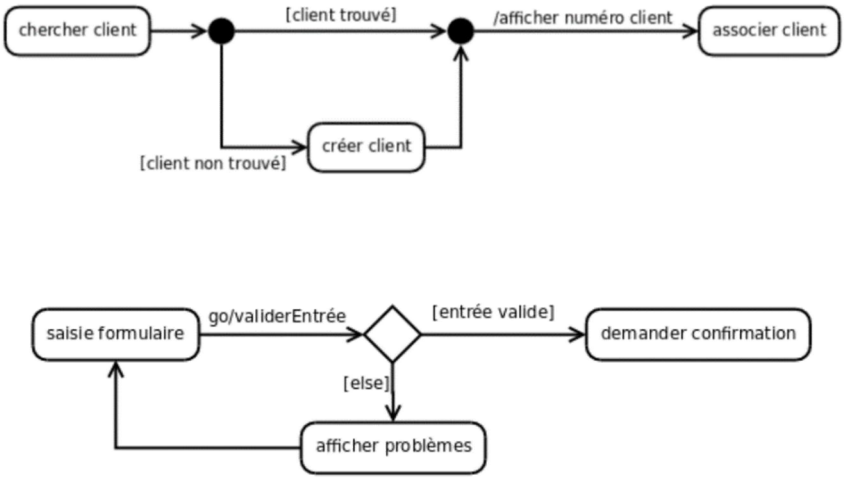


Le **pseudo-état de *choix*** ◇ est utilisé dans deux cas : sélection et convergence de séquences exclusives.
Contrairement à un point de jonction, les gardes situées après le point de décision sont évaluées au moment où il est atteint. Cela permet de baser le choix sur des résultats obtenus en franchissant le segment avant le point de choix, en particulier lorsqu'un effet (activité) est placée dans celui-ci.
Les conditions de gardes doivent être exclusives. L'utilisation d'une clause [*else*] est recommandée après un point de décision, car elle garantit un modèle correct en englobant tout ce qui n'est pas décrit dans les autres expressions booléennes et en assurant ainsi qu'un moins un segment en aval est franchissable.

Par exemple, ici : dès que l'évènement apparaît, le pseudo-état de choix est atteint. Si la condition est vraie, c'est l'état 3 qui devient actif, sinon, c'est l'état 2.





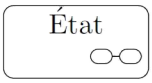


Quelques exemples d'utilisation des pseudo-état de jonction et de choix

3.3.3. État composite et hiérarchisation

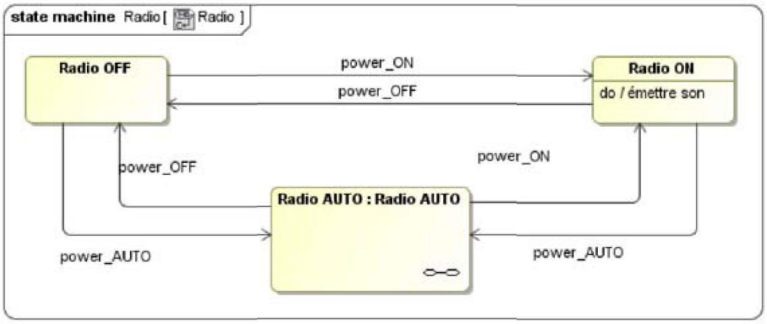
Un état composite, appelé parfois *super-état*, décrit les évolutions internes d'un état à l'aide d'un autre diagramme d'état. Cette structure qui permet d'englober plusieurs sous-états exclusifs qui sont considérés comme *hiérarchiquement inférieur* au diagramme principal, permet de rendre ce dernier plus lisible en entrant séparément dans le détail des évolutions internes du système.

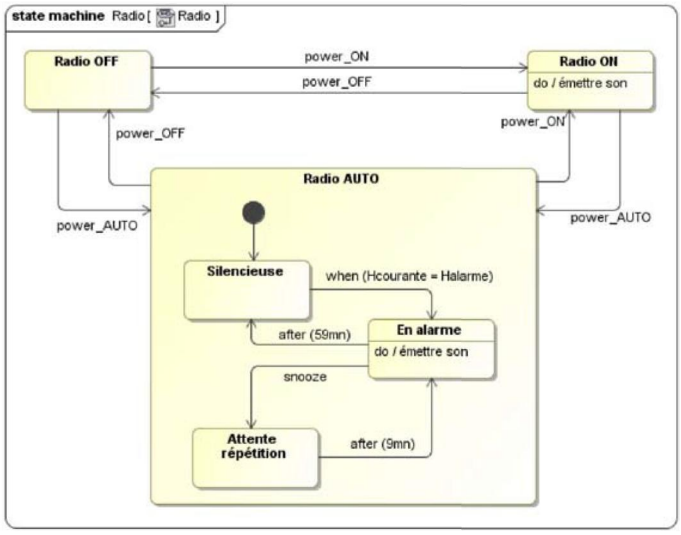
Pour repérer un état composite, un signe symbolisant des lunettes est apposé sur l'état.



L'utilisation d'un état composite s'avère indispensable lorsque le comportement à décrire comprend des diagrammes s'exécutant en parallèle (voir 3.3.4)

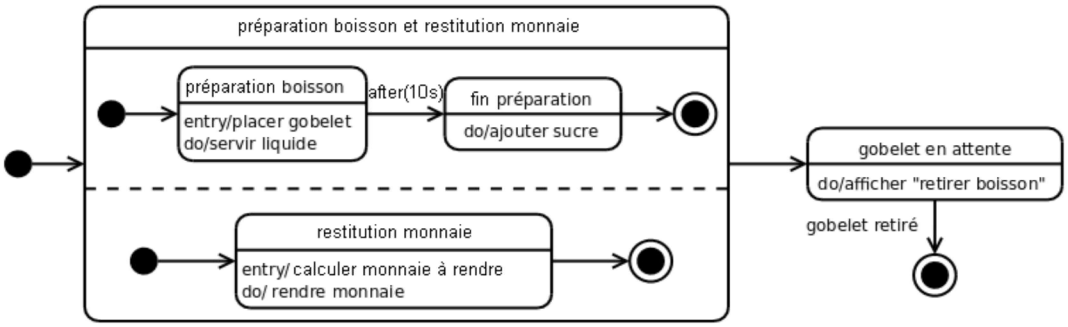
Radio réveil





3.3.4. État composite orthogonal

Dans un état composite orthogonal, plusieurs graphes d'états peuvent évoluer simultanément dans des *régions* séparées par des pointillés. Les différentes régions de l'état orthogonal fonctionnent en parallèle sans aucune influence les unes sur les autres. Dans ce cas de figure ***plusieurs états sont actifs en même temps*.**



Distributeur de boisson

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.

Toutes les régions 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 d'utiliser des **transitions constituées de barres de synchronisation *fork* et *join***.

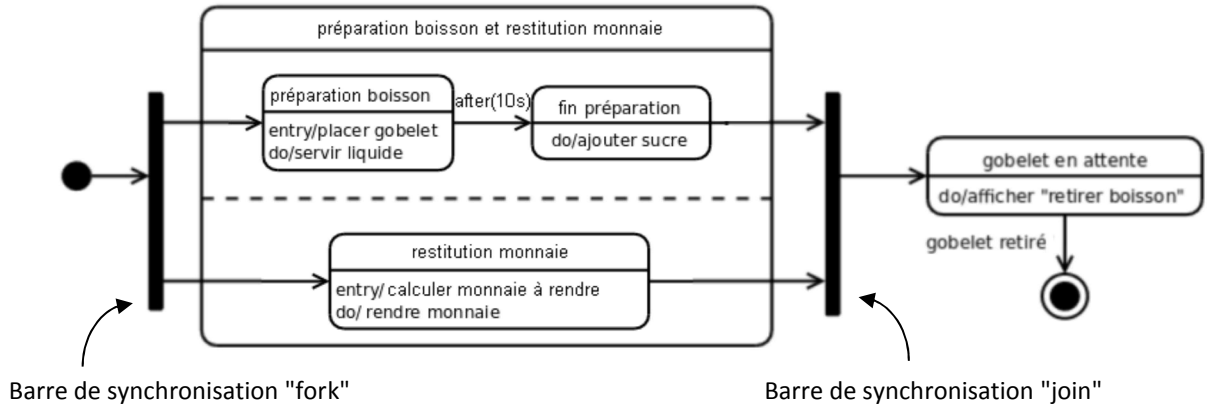


Diagramme d'état de la machine à café équivalente à celui au dessus

Les transitions automatiques (sans évènement ou condition de garde) qui partent d'une barre de synchronisation "*fork*" se ***déclenchent en même temps***.

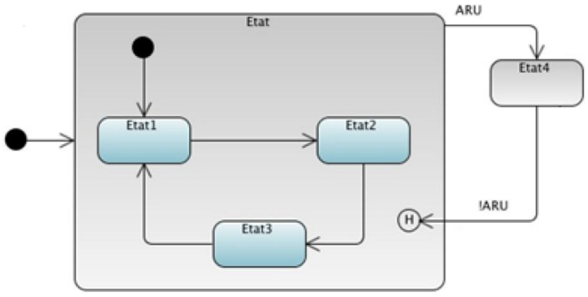
Une barre de synchronisation *join* n'est ***franchie*** qu'***après franchissement de toutes les transitions qui s'y rattachent***.

3.3.5. Historique d'un état composite

L'état actif au moment de la sortie d'un état composite peut être mémorisé par l'indication *historique*. 

Lors de la désactivation de l'état composite, l'état actif situé au même niveau hiérarchique est mémorisé. Lors de la réactivation de l'état composite, la machine d'état se réactive au niveau de l'état composite.

L'*historique* est particulièrement utilisé pour permettre à un système de recommencer en cours de cycle lors du redémarrage après un appui sur stop ou l'arrêt d'urgence.



Historique et arrêt d'urgence (ARU)

Une région ne peut avoir qu'un seul pseudo-état initial et historique à la fois.

3.4. Transition : événement déclencheur, condition de garde et effet associé

Une fois recensés tous les états d'un système, il faut ensuite en modéliser ses évolutions en identifiant les conditions de changement d'état.

On appelle ***état source*** l'état de départ d'une transition

Une transition modélise la possibilité d'un passage instantané d'un état vers un autre.
Elle est :

- orientée ;
- n'a pas de durée ;
- n'est évaluée que si l'état source est actif.

On appelle ***état cible*** l'état d'arrivée d'une transition

Lors du franchissement d'une transition d'un état source à un état cible :

- *l'éventuelle activité do de l'état source est* ***interrompue*** ;
- ↓
- l'éventuelle activité *exit* de l'état source est exécutée ;
- ↓
- l'état source devient inactif ;
- ↓
- l'éventuelle activité *effet* est exécutée (voir 3.4.3) ;
- ↓
- l'état cible devient actif.

Le franchissement de la transition est conditionné par des évènements déclencheurs et des conditions de garde. Ces événements et conditions de franchissement ainsi que l'éventuel effet associé à la transition sont indiqués le long de la flèche qui la symbolise suivant la notation :

événement [garde] / effet

3.4.1. Evénement

Un **événement est daté** dans le temps et il est ***traité instantanément lors de son occurrence*** (apparition) : *appui sur un bouton, arrivée en fin de course d'un mécanisme, dépassement d'une valeur seuil…*
Sauf indication contraire, c'est le front montant de l'évènement qui est considéré.
Il n'est pas caractérisé par une durée, c'est à dire qu'il a lieu ou non. ***Il n'est jamais mémorisé*** et est ***donc perdu s'il ne mène à aucune évolution du diagramme d'état***.
L'événement est le déclencheur du franchissement de la transition. On distingue trois types :

- l'événement de ***signal*** prend en compte l'envoi ou la réception d'un signal : *appui sur un bouton, arrivée en fin de course d'un mécanisme* ;
- l'événement de ***changement*** prend en compte le changement d'une valeur interne du modèle. Il se modélise avec le mot clé *when* suivi d'une expression booléenne concernant une variable interne : *compteur when(N=3)* ;
- l'événement ***temporel*** :
  - relatif : *after (T)* se déclenche après une durée T passé dans l'état d'amont. Il s'agit d'une temporisation ;
  - absolu : *at(D)* se déclenche à la date D dans un référentiel de temps dont l'origine correspond généralement au démarrage du fonctionnement du système.

Si plusieurs événement au choix sont écrits, ils sont séparés par une virgule.

3.4.2. Garde

La garde est une ***condition de franchissement de la transition*****.** C'est une condition supplémentaire à l'événement déclencheur, mais optionnelle.

C'est une condition booléenne. Contrairement à l'événement qui lui est localisé dans le temps, **elle traduit une condition qui dure dans le temps** et ***qui doit persister*** : *vitesse non nulle, température >20°….*
Par exemple, lors de l'appui d'un bouton :

- L'événement modélise le fait de savoir s'il vient d'être enfoncé ;
- La garde modélise le fait de savoir s'il est enfoncé.

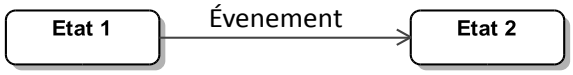
La syntaxe d'une condition de garde vérifiant l'activité de l'état TOTO est : [in TOTO].

Si la garde n'existe pas, elle est considérée toujours vraie.
Plusieurs transitions avec le même événement doivent avoir des conditions de garde différentes.

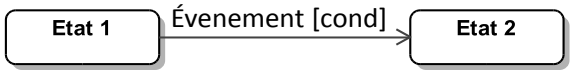
3.4.3. Effet

Un effet est un comportement (activité) accompli lorsque la transition est franchie.
Les activités associées aux effets sont considérées instantanées, c'est-à-dire qu'elles ne peuvent être interrompues.
Une transition peut ne pas avoir d'effet.

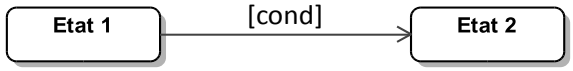
Quelques cas à maitriser :



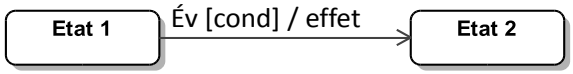
À l'occurrence (apparition) de l'événement, la transition est franchie, sans condition. l'éventuelle activité *do* de l'état 1 est interrompue et l'état 2 s'active.



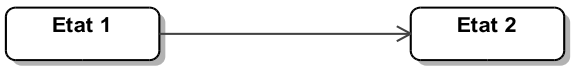
À l'occurrence de l'événement, si la garde « cond » est vraie, la transition est franchie : l'éventuelle activité *do* de l'état 1 est interrompue et l'état 2 s'active.
Sinon, ***l'événement est « perdu »*** et ***il faut attendre une seconde occurrence*** pour éventuellement franchir la transition.



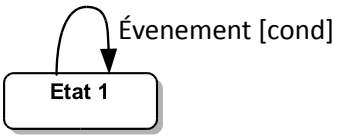
Si la garde « cond » est vraie, la transition est :
- immédiatement franchie s'il n'y a pas d'activité associée à l'état 1 ;
- franchie dès la fin de l'éventuelle activité associée à l'état 1 afin de provoquer l'événement déclencheur implicite. ***Il faut donc que ce soit une activité ayant une fin***.



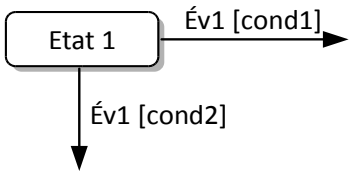
À l'occurrence de l'événement, si la garde « cond » est vraie, la transition est franchie : les activités associées à l'effet sont effectuées puis l'état 2 s'active.



Transition dite automatique. Dans ce cas, la condition prise en compte pour la transition est la fin de l'activité associée au comportement *do* de l'état 1.

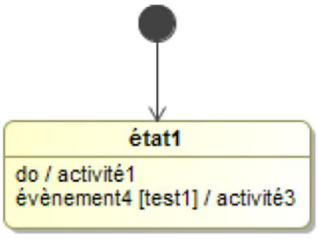


Une transition réflexive entraîne une sortie de l'état puis un retour dans ce même état, avec appel des éventuelles activités *exit* et *entry*.



Plusieurs transitions peuvent quitter un même état.
Une seule d'entre elles doit être déclenchée ; ***les événements et/ou les conditions de garde doivent donc être exclusives***.

Remarque : il est possible de placer, dans un état, des activités associées à un événement ou/et à une garde :



Notons cependant qu'il est déconseillé d'utiliser ces transitions que l'on peut qualifier d'*internes*. Il est plutôt recommandé de changer d'état lors de l'apparition des événements.

4. Diagramme de séquence (*Sequence Diagram* ou *sd*)

4.1. Rappel : diagramme des cas d'utilisation

L'objectif d'un diagramme des cas d'utilisation est de spécifier les acteurs qui utilisent le système et la manière dont elles l'utilisent.

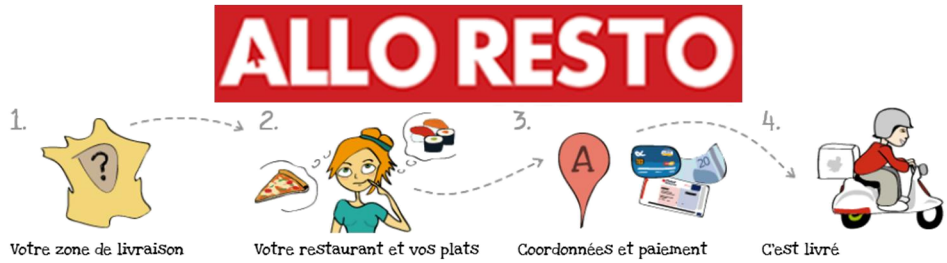
L'énoncé d'un cas d'utilisation est purement fonctionnel. Il est défini en termes de résultats attendus et est donc totalement indépendant des solutions technologiques choisies pour pouvoir le réaliser.

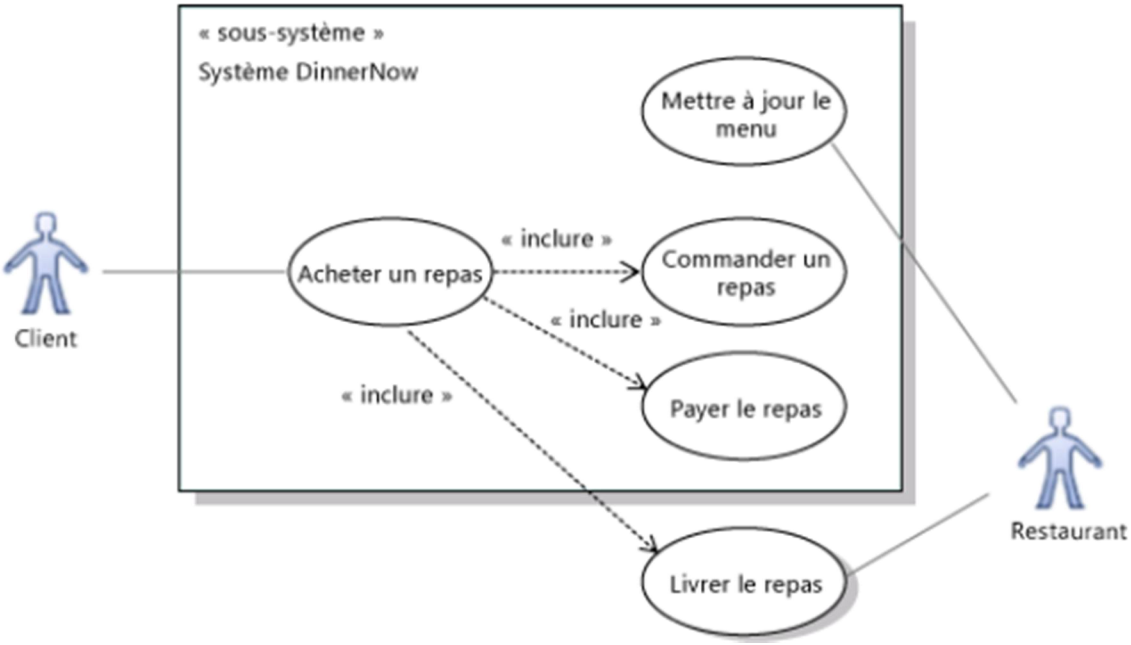
Les acteurs sont placés à l'extérieur de la frontière d'étude du système :

- à gauche s'ils sont considérés comme « principaux » ;
- et à droite s'ils sont considérés comme « secondaires ».

Les cas d'utilisation pour lesquelles le système n'a pas été spécifiquement conçus (lavage, recyclage, réparation,…) ne doivent pas apparaître sur ce diagramme.

Dans le cas d'un système de vente de repas en ligne (« Dinner Now ») qui doit permettre aux clients de sélectionner des éléments dans un menu, et aux restaurants proposant ce service de mettre le menu à jour, le diagramme des cas d'utilisation peut être le suivant :

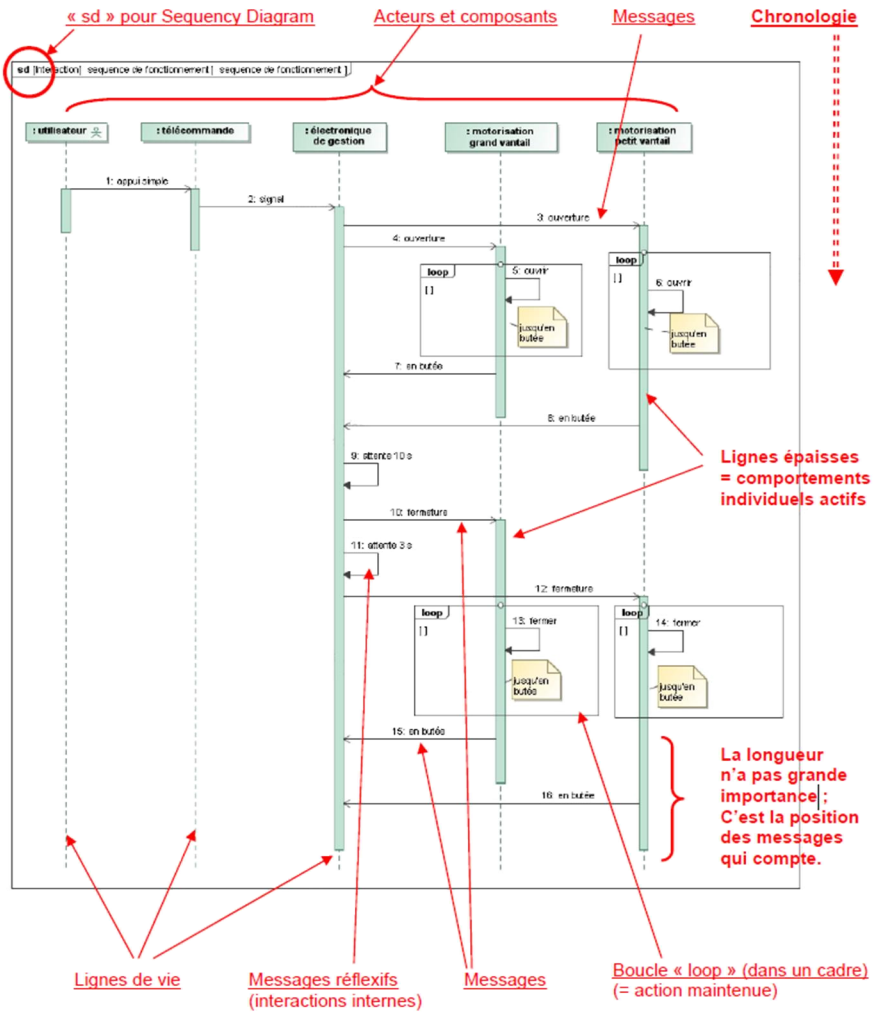




4.2. Diagramme de séquence

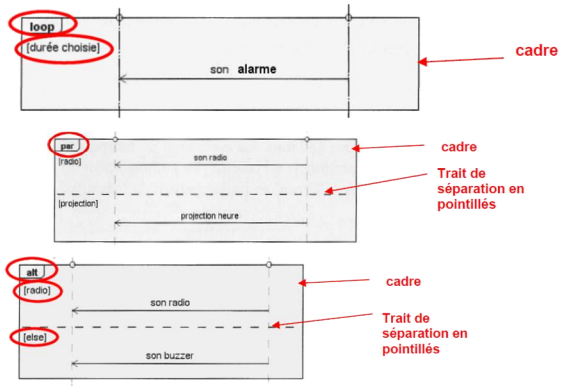
Le diagramme de séquence, décrit, ***dans l'ordre chronologique***, l'enchaînement des interactions (appelées *messages*) entre les acteurs du système ou entre des composants du système eux-mêmes.

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.



Les principaux éléments sont :

Ligne de vie	Ligne verticale en pointillée. Une pour chaque élément dialoguant (acteur, système, sous-système ou composant).
Message	Flèche horizontale unidirectionnelle entre deux lignes de vie représentant un élément de communication. Elle déclenche une période d'activité (un comportement) chez le receveur du message, pour qui, l'arrivée d'un message est un événement. Ce message peut être : • synchrone : l'émetteur attend une réponse, son comportement est bloqué pendant l'attente ; • asynchrone : l'émetteur n'attend pas de réponse, son comportement continu ; • une réponse. Il est possible d'avoir un message réflexif correspondant à une interaction interne au composant.
Période d'activité	Bande verticale sur une ligne de vie. Elles sont optionnelles mais peuvent faciliter la lecture du diagramme.
fragment	Cadre englobant une partie de la séquence ***Loop*** Boucle, le fragment est maintenu pendant une durée qui dépend de la condition de garde. ***par*** Séquence en parallèle. Les régions peuvent être exécutées simultanément. ***alt*** Séquences alternatives conditionnées. Seule la région dont la condition est vraie s'exécute.



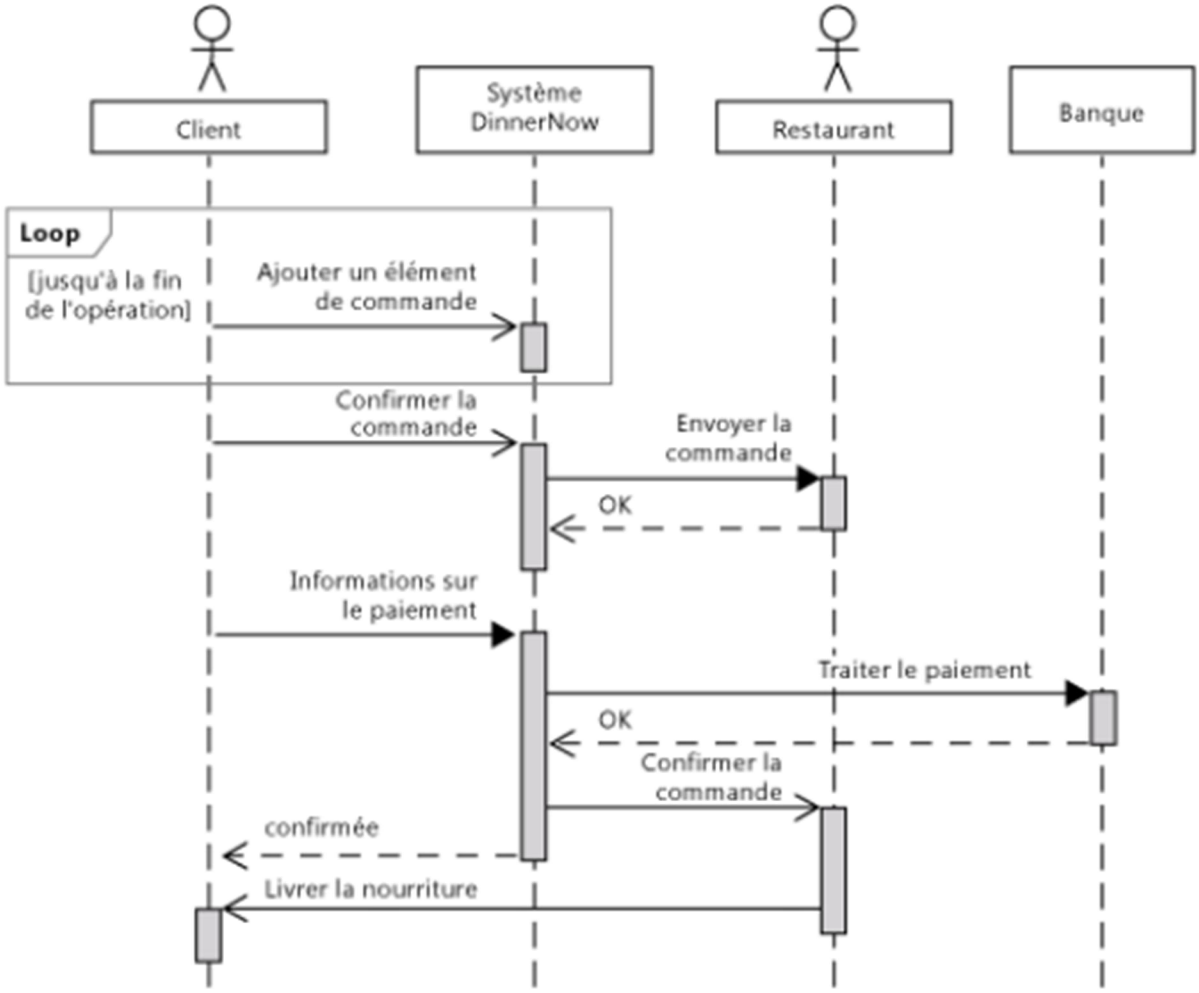


Diagramme de séquence du système « Dinner Now »