UML

Chapitre 2 : Diagramme d'activités et diagramme de séquences

Leçon 2 : Diagramme de séquences

1) Rôle du diagramme de séquences :

Le diagramme de séquence fait parties des diagrammes comportementaux (dynamique) et plus précisément des diagrammes d'interactions.

- Il permet de représenter des échanges entre les différents objets et acteurs du système en fonction du temps.
- A moins que le système à modéliser soit extrêmement simple, nous ne pouvons pas modéliser la dynamique globale du système dans un seul diagramme. Nous ferons donc appel à un ensemble de diagrammes de séquences chacun correspondant à une sous fonction du système, généralement d'ailleurs pour illustrer un cas d'utilisation.

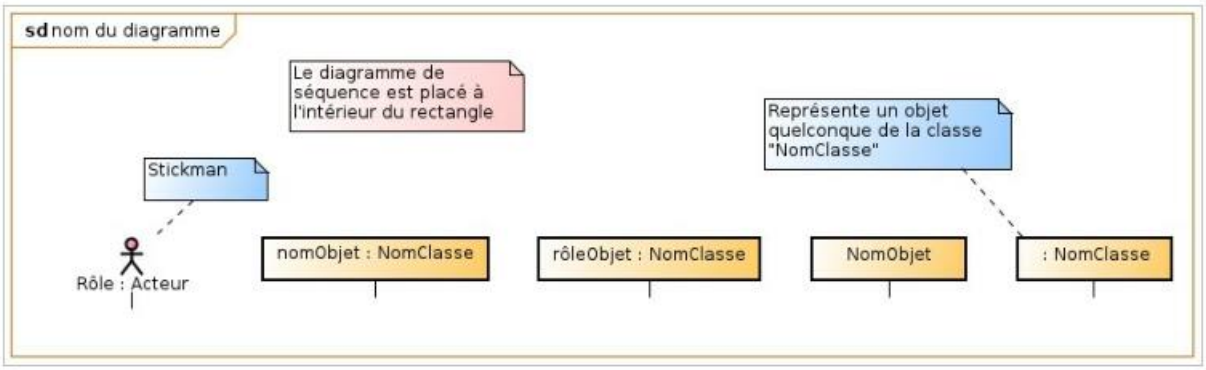
2) Représentation du diagramme de séquence :

2-1) Délimitation du diagramme de séquence :

Le diagramme de séquence est placé dans un rectangle qui dispose d'une étiquette **sd** en haut à gauche (qui signifie **sequence diagramm**) suivi du nom du diagramme.

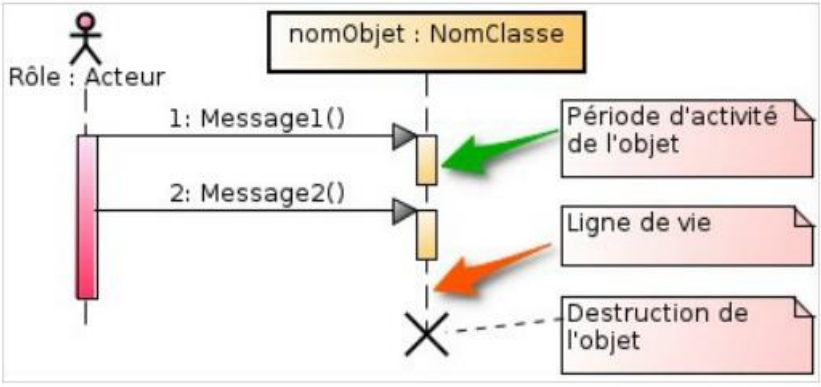
2-2) L'objet :

Dans un diagramme de séquence, l'objet à la même représentation que dans le diagramme des objets. C'est-à-dire un rectangle dans lequel figure le nom de l'objet. Le nom de l'objet est généralement souligné et peut prendre l'une des quatre formes suivantes :



Les diagrammes de séquences représentant les échanges entre les objets mais aussi les échanges avec les acteurs, nous trouverons aussi la représentation du ***stickman*** (qui peut être considéré comme un objet).

2-3) La ligne de vie :



Comme il représente la dynamique du système, le diagramme de séquence fait entrer en action les instances de classes intervenant dans la réalisation d'un cas d'utilisation particulier.

- A chaque objet est associé une ligne de vie (en trait pointillés à la verticale de l'objet) qui peut être considéré comme un axe temporel (le temps s'écoule du haut vers le bas). Dans ce genre de diagramme, la quantification du temps n'a pas d'importance.

- La ligne de vie indique les périodes d'activité de l'objet (généralement, les moments ou l'objet exécute une de ces méthodes).
- Lorsque l'objet est détruit, la ligne de vie s'achève par un croix.

2-4) Les messages :

2-4-1) Définition :

Un message définit une communication particulière entre des lignes de vie. Ainsi, un message est une communication d'un objet vers un autre objet. La réception d'un message est considérée par l'objet récepteur comme un événement qu'il faut traiter (ou pas). Plusieurs types de messages existent, les plus communs sont :

- **L'invocation d'une opération** : ***message synchrone*** (appel d'une méthode de l'objet cible).
- **L'envoi d'un signal** : ***message asynchrone*** (typiquement utilisé pour la gestion événementielle).
- La création ou la destruction d'une instance de classe au cours du cycle principal.

- **Les messages synchrones :**

  La réception d'un message synchrone doit provoquer chez le destinataire le lancement d'une de ses méthodes (qui souvent porte le même nom que le message).
  L'expéditeur du message reste bloqué pendant toute l'exécution de la méthode et attend donc la fin de celle-ci avant de pouvoir lancer un nouveau message.
  C'est le message le plus fréquemment utilisé.

- **Les messages asynchrones :**

  Dans le cas ***d'un message asynchrone, l'expéditeur n'attend pas la fin de l'activation de la méthode invoquée chez le destinataire.*** Un message asynchrone peut être :

  - **Un appel de méthode :**

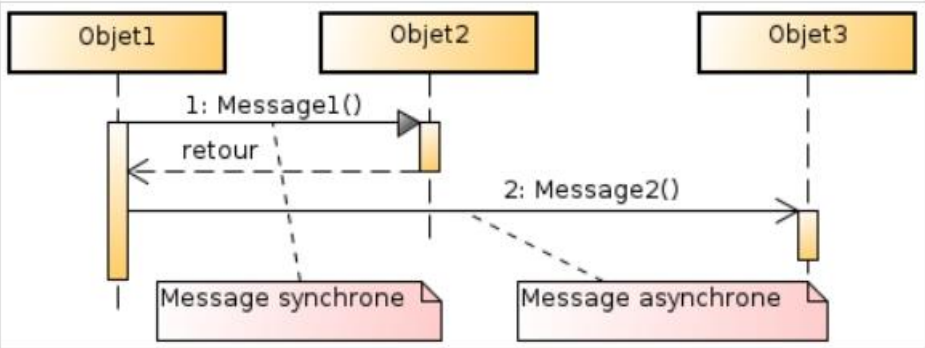
    Fréquent dans un système ***multi-threads*** (multi-tâche). Ainsi, l'objet expéditeur n'étant pas bloqué pendant l'exécution de la méthode, il peut continuer ainsi à envoyer d'autres messages.

  - **Un signal (cas le plus fréquent) :**

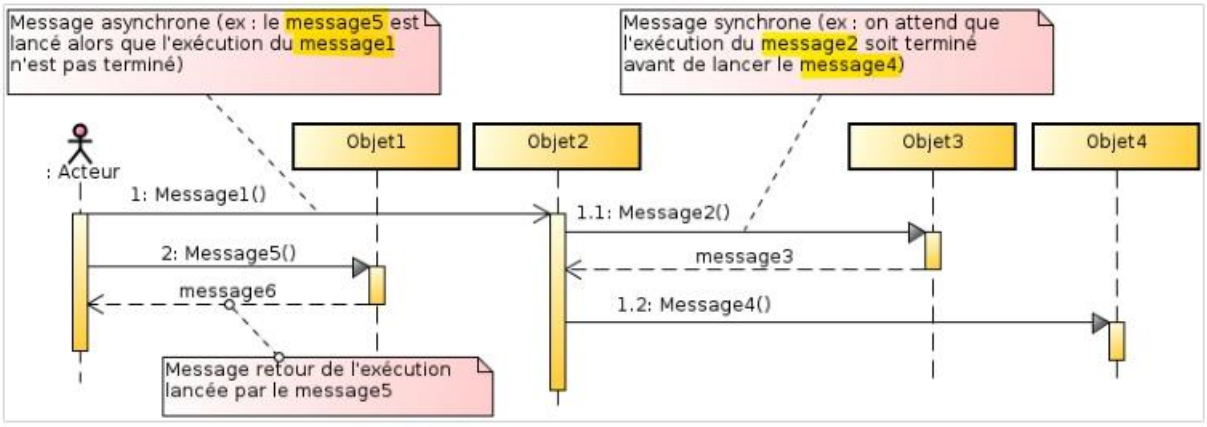
    L'objet expéditeur transmet juste une information à l'objet destinataire. Souvent, ce sont les acteurs ou les périphériques qui envoient des signaux, ***typiquement utilisé dans la gestion événementielle d'une IHM graphique, comme la librairie QT, par exemple.***

2-4-2) Représentation graphique : Dans le diagramme de séquence, les envois de messages sont représentés par des flèches horizontales qui vont de la ligne de vie de l'objet émetteur vers la ligne de vie de l'objet récepteur du message.

- **Les messages synchrones :** *(flèche avec un triangle plein à son extrémité).*
- **Les messages asynchrones :** *(simple flèche)*
- **Les messages de retour (réponses) :** *(simple flèche en pointillés).*

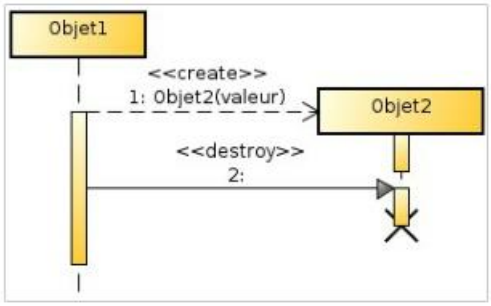


- Si une méthode qui a été activée (par un message) doit retourner des valeurs à la fin de son activation, cela se fait par un message retour.
- Le message de retour n'est donc pas un appel de méthode (il ne provoque donc pas l'activation d'un objet)
- Le message retour porte souvent le nom de l'élément retourné.



2-4-3) <u>Création et destruction d'objets</u> : Une séquence peut aussi contenir la **création** ou la **destruction** d'un objet :

- La création d'un objet est matérialisée par un message spécifique, appel d'un **constructeur**, généralement accompagné du stéréotype ***« create »*** qui pointe sur le début (le sommet) de la ligne de vie de l'objet créé (Le rectangle de l'instance de la classe est alors surbaissée).
- La destruction d'un objet est représentée par une croix à la fin de sa ligne de vie. Souvent l'objet est détruit suite à la réception d'un message mais ce n'est pas obligatoire. Dans ce cas là, il porte le stéréotype ***« destroy ».***



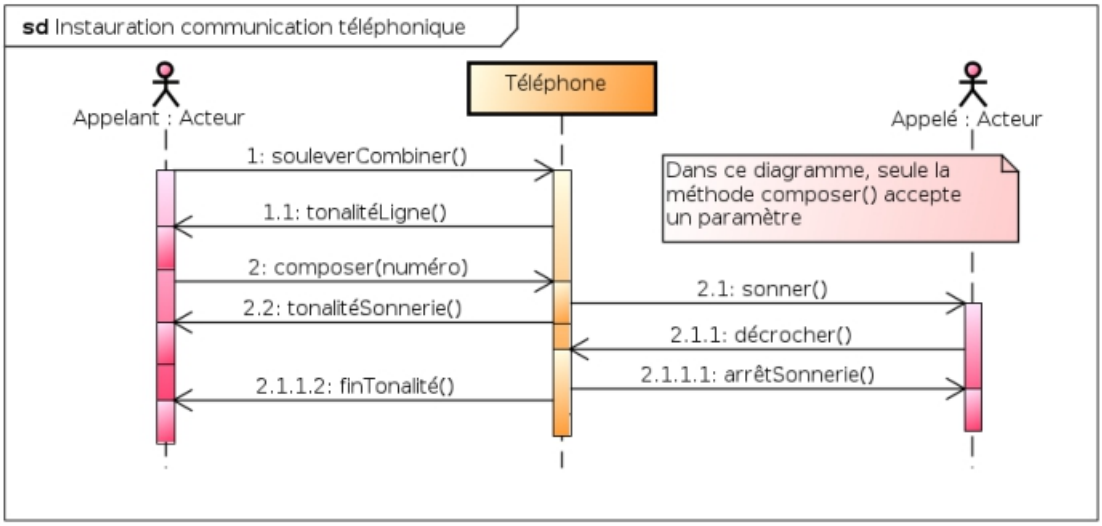
2-4-4) <u>Syntaxe des messages synchrones et asynchrones:</u>

Dans la plupart des cas, la réception d'un message est suivie de l'exécution de la méthode de la classe cible. Cette méthode peut recevoir des arguments et la syntaxe des messages permet de transmettre ces arguments. La plupart du temps, dans un diagramme de séquence, nous pouvons nous contenter de définir un message par :

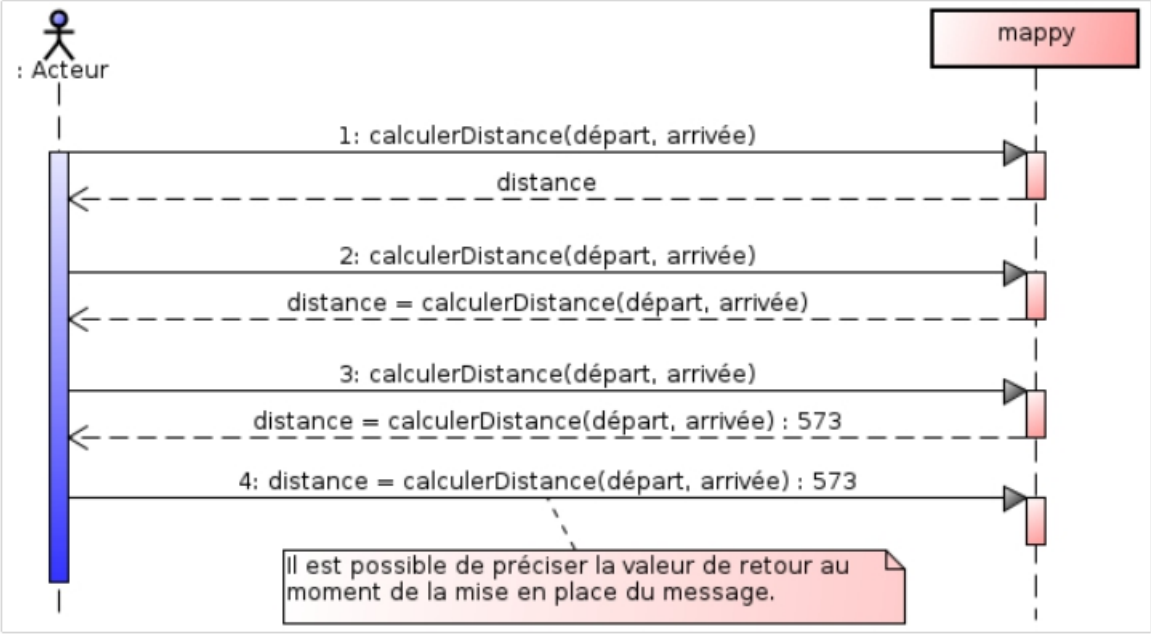
- **Son <u>nom</u>** (qui est le nom de la méthode appelée ou du signal envoyé).

Nous pouvons lui adjoindre facultativement :

- **Une <u>numérotation</u>** devant le nom message (séparé du nom du message par 2 point **" : "**). La numérotation s'effectue séquentiellement à partir de 1.
- **Les <u>paramètres passés à la méthode ou au signal</u>** (entre parenthèses après le nom du message).



2-4-5) <u>Syntaxe des réponses (messages retour)</u> : Comme pour les messages synchrones ou asynchrones, nous pouvons nous contenter de donner au message retour un simple nom, mais nous pouvons aussi les caractériser plus précisément en utilisant la syntaxe suivante : ***numéro : attribut = nomMessage (paramètres) : valeurDeRetour.***

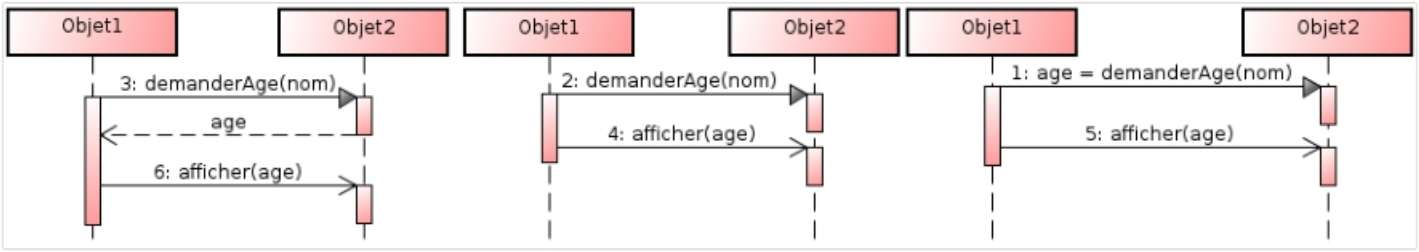


2-4-6) <u>Remarque sur la syntaxe des messages dans les diagrammes de séquence :</u>
La syntaxe que nous venons de voir est simple et très souvent largement suffisante pour décrire les messages des diagrammes de séquence.

3) <u>Quelques séquences expliquées :</u>

3-1) <u>Messages retours implicites et explicites :</u>

Le retour d'un message synchrone peut ne pas être représenté, le retour est alors ***implicite***.
Par contre, dans le cas d'un message asynchrone, il est impératif de faire apparaître le message de retour. Le retour est ***explicite.*** Bien entendu, si l'exécution de la méthode lancée par le message asynchrone ne doit rien retourner, il est évident que nous ne devons pas représenter le message de retour (c'est généralement le cas le plus classique, notamment avec la gestion événementielle).
Les 3 diagrammes suivants sont équivalents :

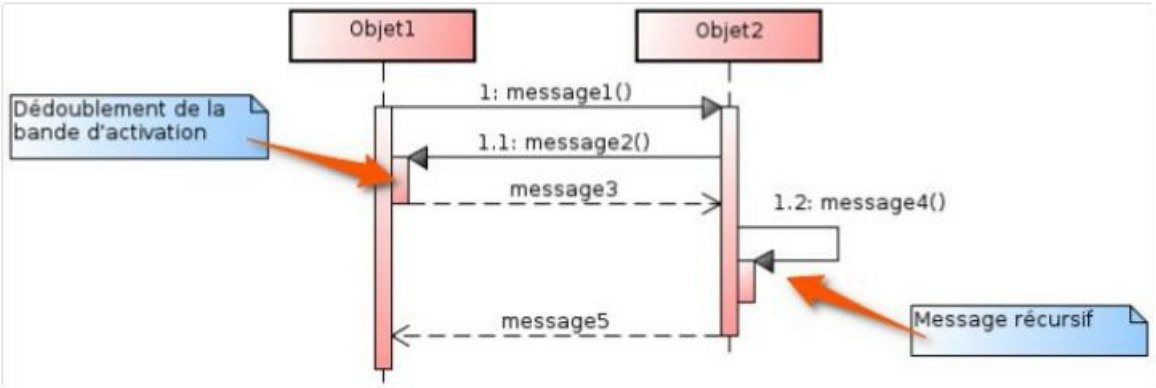


3-2) <u>Recouvrement des bandes d'activations :</u>

Lorsqu'un objet est déjà activé il peut quand même recevoir d'autres messages (***appel d'une autre de ses méthodes***), cela se représente par un dédoublement de la bande d'activation.

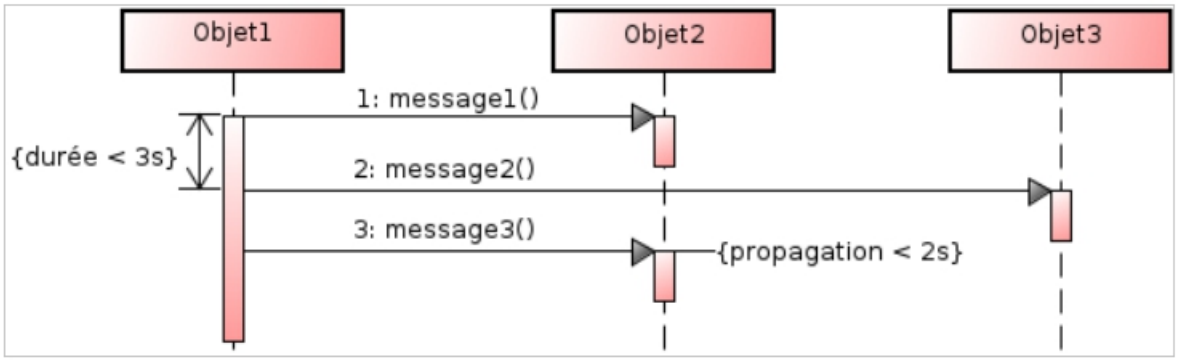
3-3) <u>Messages récursifs :</u>

Un objet peut s'envoyer un message à lui-même (***utilisation d'une autre méthode du même objet***). Cela se représente là aussi par un dédoublement de la bande d'activation.



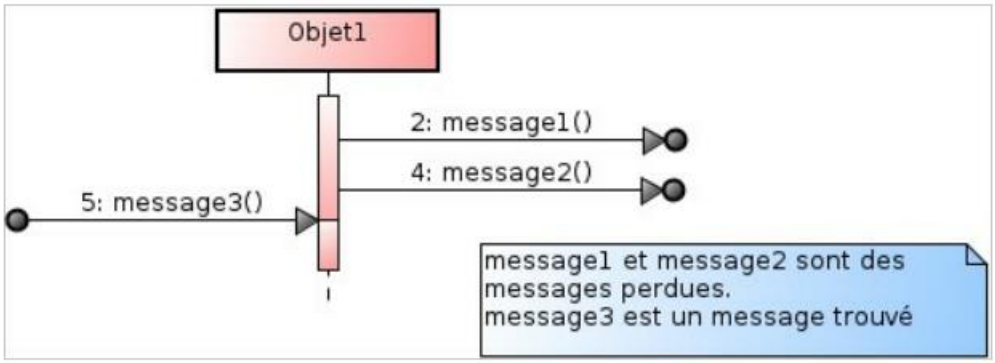
3-4) Contraintes temporelles :

Des repères temporels avec des contraintes peuvent être placés le long de la ligne de vie. Un message avec un temps de propagation non négligeable peut être représenté par une flèche oblique ou en l'écrivant explicitement.



3-6) Messages perdus et trouvés :

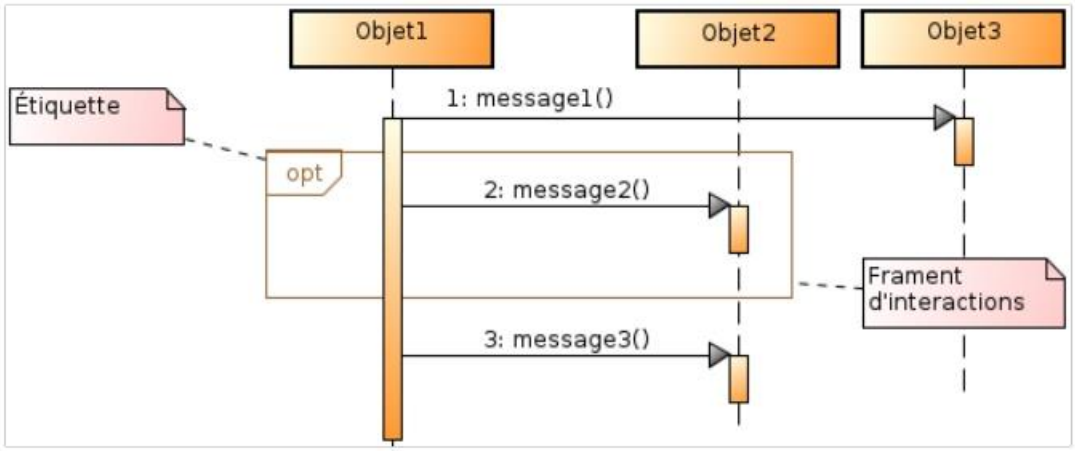
- **Le message perdu** est un message dont nous connaissons l'émetteur mais pas le récepteur. Il est représenté par une flèche partant de la ligne de vie d'un élément vers un disque noir. Le message perdu pouvant être, à l'origine, synchrone ou asynchrone, nous avons donc les types de pointes de flèches. ***Cette sorte de message permet de modéliser, par exemple, les scenarii de pertes de message sur un réseau.***



- **Le message trouvé** est un message dont nous connaissons le destinataire mais pas l'émetteur. Il est représenté par une flèche partant d'un disque noir vers la ligne de vie d'un élément. ***Ce message peut être utilisé pour modéliser le comportement d'un élément suite à la réception d'un message d'exception.***

4) Fragments d'interactions combinés :

Un fragment d'interactions est une partie du diagramme de séquence (délimitée par un rectangle) associée à une étiquette (dans le coin supérieur gauche). L'étiquette contient un **opérateur d'interaction** qui permet de décrire des modalités d'exécution des messages à l'intérieur du cadre.

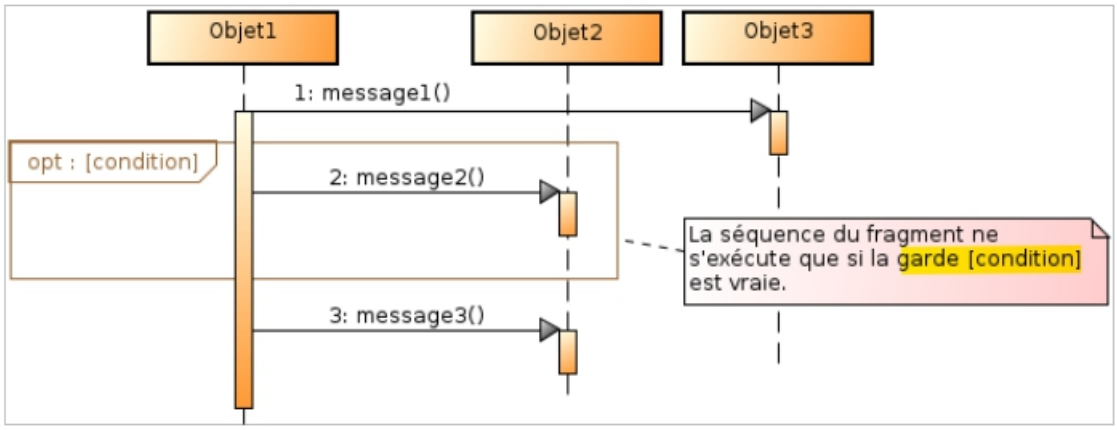


Les ***opérandes*** d'un ***opérateur d'interaction*** sont séparés par une ligne pointillée. Les conditions de choix des ***opérandes*** (***éventuels***) sont données par des **expressions booléennes** entre crochets (***[ ]***).

Les principales modalités sont les boucles, les branchements conditionnels, les alternatives, les envois simultanés, etc.

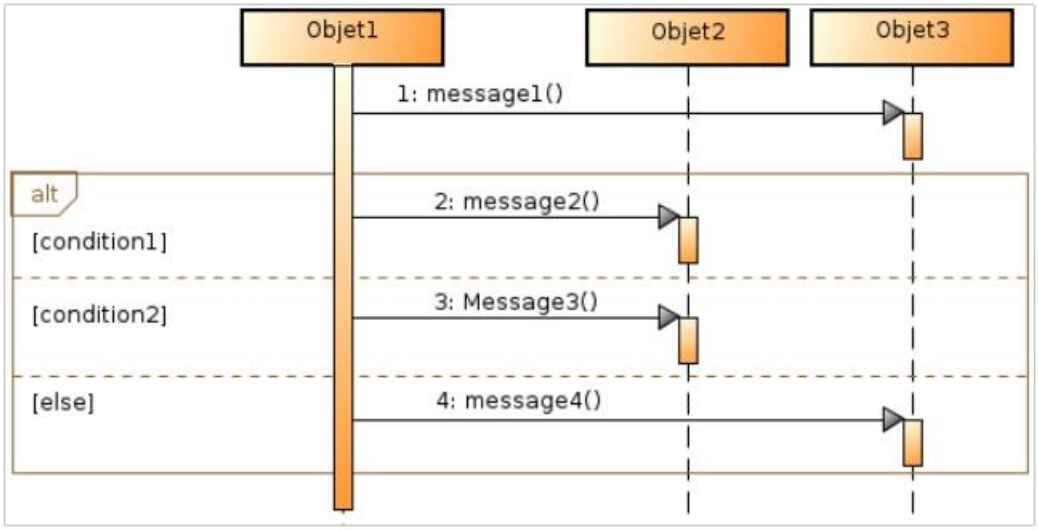
4-1) Fragment d'interaction avec opérateur « opt » :

L'opérateur ***option*** (**opt**) comporte un opérande et une condition de garde associée. Le sous-fragment s'exécute si la condition de garde est vraie et ne s'exécute pas dans le cas contraire.



4-2) Fragment d'interaction avec opérateur « alt » :

L'opérateur ***alternatives*** (**alt**) est un opérateur conditionnel possédant plusieurs opérandes séparés par des pointillés. ***C'est l'équivalent d'une exécution à choix multiples. Chaque opérande détient une condition de garde.*** Seul le sous-fragment dont la condition est vraie est exécuté. La condition ***else*** est exécutée que si aucune autre condition n'est valide.



4-3) Fragment d'interaction avec opérateur « loop » :

L'opérateur de ***boucle*** (**loop**) exécute une itérative dont la séquence qu'elle contient est exécutée tant que la garde qui lui est associée est vraie.

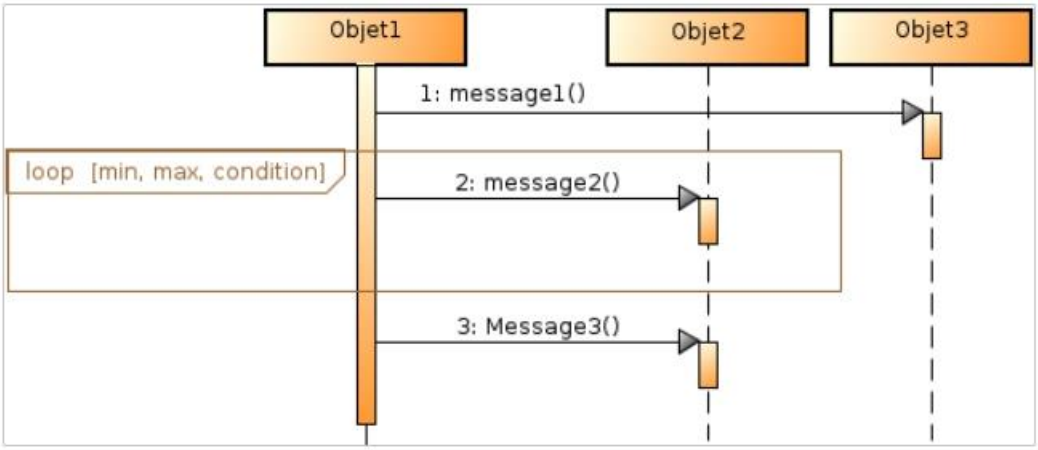
- La garde s'écrit de la façon suivante :

loop [min, max, condition] : Chaque paramètre (min, max et condition) est optionnel.

- Le contenu du cadre est exécuté ***min*** fois, puis continue à s'exécuter tant que la condition et que le nombre d'exécution de la boucle ne dépasse pas ***max*** fois.

- Exemple de gardes :

loop[3]→La séquence s'exécute **3** fois.
Loop[1, 3, code=faux] La séquence s'exécute **1** fois puis un maximum de **2** autres fois si **code=faux**.



4-4) <u>Fragment d'interaction avec opérateur « par » :</u>

Un fragment d'interaction avec l'opérateur ***de traitements parallèles*** (**par**) contient au moins deux sous fragments (***opérandes***) séparés par des pointillés qui s'exécutent simultanément (***traitements concurrents***).