Diagramme de séquence *(Sequence diagram)*

Les principales informations contenues dans un diagramme de séquence sont les messages échangés entre les lignes de vie, présentés dans un ordre chronologique. Ainsi, contrairement au diagramme de communication, le temps y est représenté explicitement par une dimension (la dimension verticale) et s'écoule de haut en bas.

1 Représentation des lignes de vie

Une ligne de vie se représente par un rectangle, auquel est accroché une ligne verticale pointillée, contenant une étiquette dont la syntaxe est :

```
[<nom_du_rôle>] : [<Nom_du_type>]
```

Au moins un des deux noms doit être spécifié dans l'étiquette, les deux points (:) sont, quand à eux, obligatoire.

2 Représentation des messages

Un message définit une communication particulière entre des lignes de vie. Plusieurs types de messages existent, les plus commun sont :

- l'envoi d'un signal ;
- l'invocation d'une opération ;
- la création ou la destruction d'une instance.

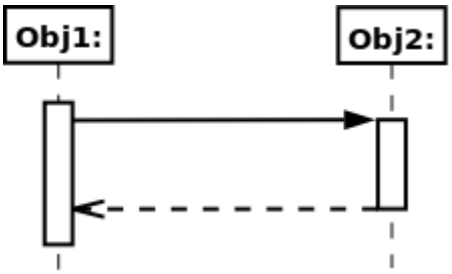
Messages asynchrones

Représentation d'un message asynchrone.

Une interruption ou un évènement sont de bons exemples de signaux. Ils n'attendent pas de réponse et ne bloquent pas l'émetteur qui ne sait pas si le message arrivera à destination, le cas échéant quand il arrivera et s'il serra traité par le destinataire. Un signal est, par définition, un message asynchrone.

Graphiquement, un message asynchrone se représente par une flèche en traits pleins et à l'extrémité ouverte partant de la ligne de vie d'un objet expéditeur et allant vers celle de l'objet cible

Messages synchrones

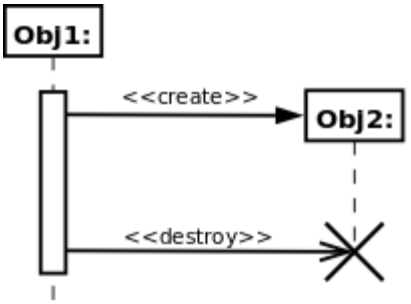


Représentation d'un message synchrone.

L'invocation d'une opération est le type de message le plus utilisé en programmation objet. L'invocation peut être asynchrone ou synchrone. Dans la pratique, la pluspart des invocations sont synchrones, l'émetteur reste alors bloqué le temps que dure l'invocation de l'opération.

Graphiquement, un message synchrone se représente par une flèche en traits pleins et à l'extrémité pleine partant de la ligne de vie d'un objet expéditeur et allant vers celle de l'objet cible. Ce message peut être suivi d'une réponse qui se représente par une flèche en pointillé.

Messages de création et destruction d'instance

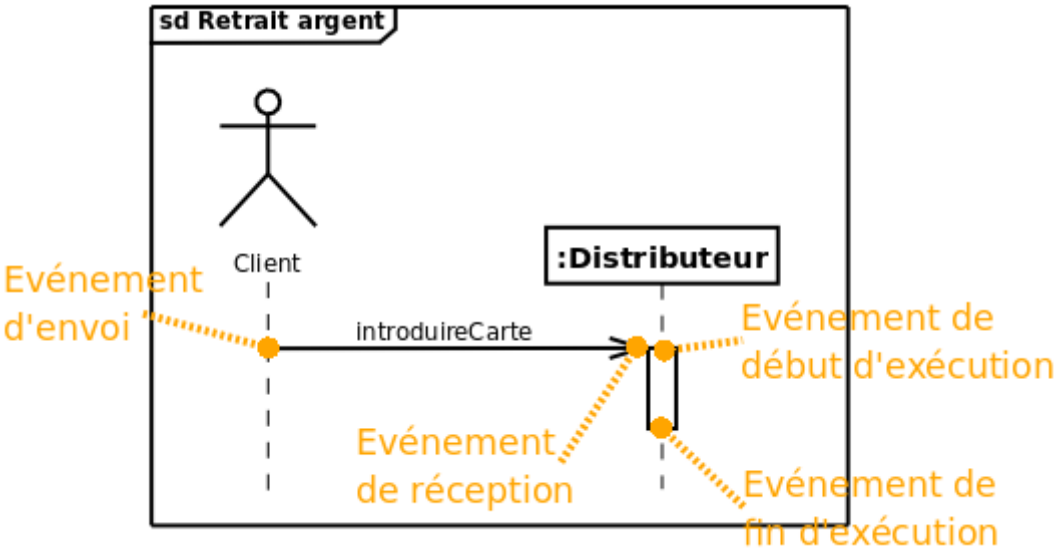


Représentation d'un message de création et destruction d'instance.

La création d'un objet est matérialisée par une flèche qui pointe sur le sommet d'une ligne de vie.

La destruction d'un objet est matérialisée par une croix qui marque la fin de la ligne de vie de l'objet. La destruction d'un objet n'est pas nécessairement consécutive à la réception d'un message.

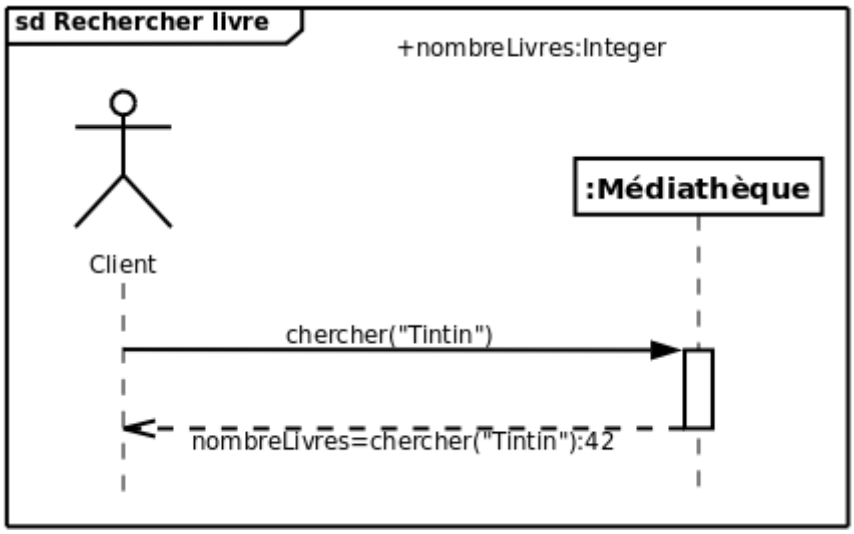
Événements et messages



Les différents évènements correspondant à un message asynchrone.

UML permet de séparer clairement l'envoi du message, sa réception, ainsi que le début de l'exécution de la réaction et sa fin.

Syntaxe des messages et des réponses



Syntaxe des messages et des réponses.

Dans la plupart des cas, la réception d'un message est suivie de l'exécution d'une méthode d'une classe. Cette méthode peut recevoir des arguments et la syntaxe des messages permet de transmettre ces arguments. La syntaxe de ces messages est la même que pour un diagramme de communication excepté deux points :

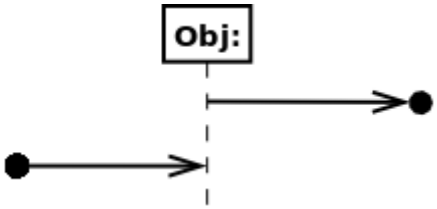
- la direction du message est directement spécifiée par la direction de la flèche qui matérialise le message, et non par une flèche supplémentaire au dessus du connecteur reliant les objets comme c'est le cas dans un diagramme de communication ;
- les numéros de séquence sont généralement omis puisque l'ordre relatif des messages est déjà matérialisé par l'axe vertical qui représente l'écoulement du temps.

La syntaxe de réponse à un message est la suivante :

```
[<attribut> = ] message [ : <valeur_de_retour>]
```

où `message` représente le message d'envoi.

Message perdu et trouvé



Représentation d'un message perdu et d'un message trouvé.

Un message complet est tel que les événements d'envoi et de réception sont connus. Comme nous l'avons déjà vu, un message complet se représente par une simple flèche dirigée de l'émetteur vers le récepteur.

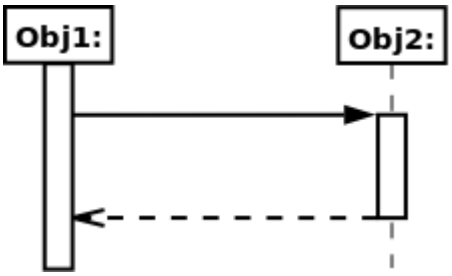
Un message perdu est tel que l'événement d'envoi est connu, mais pas l'événement de réception. Il se représente par une flèche qui pointe sur une petite boule noire.

Un message trouvé est tel que l'événement de réception est connu, mais pas l'événement d'émission. Une flèche partant d'une petite boule noire représente un message trouvé

Porte

Une porte est un point de connexion qui permet de représenter un même message dans plusieurs fragments d'interaction. Ces messages entrants et sortants vont d'un bord d'un diagramme à une ligne de vie (ou l'inverse).

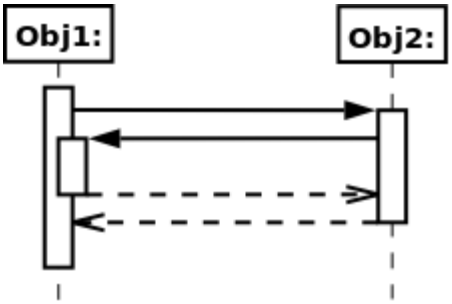
Exécution de méthode et objet actif



Représentation d'un objet actif (à gauche) et d'une exécution sur un objet passif (à droite).

Un objet actif initie et contrôle le flux d'activités. Graphiquement, la ligne pointillée verticale d'un objet actif est remplacée par un double trait vertical.

Un objet passif, au contraire, a besoin qu'on lui donne le flux d'activité pour pouvoir exécuter une méthode. La spécification de l'exécution d'une réaction sur un objet passif se représente par un rectangle blanc ou gris placé sur la ligne de vie en pointillée. Le rectangle peut éventuellement porter un label.



Représentation d'une exécution simultanée (à gauche).

Les exécutions simultanées sur une même ligne de vie sont représentées par un rectangle chevauchant comme le montre la figure

3 Fragments d'interaction combinés

Introduction

Un fragment combiné représente des articulations d'interactions. Il est défini par un opérateur et des opérandes. L'opérateur conditionne la signification du fragment combiné. Il existe 12 d'opérateurs définis dans la notation UML 2.0. Les fragments combinés permettent de décrire des diagrammes de séquence de manière compacte. Les fragments combinés peuvent faire intervenir l'ensemble des entités participant au scénario ou juste un sous-ensemble.

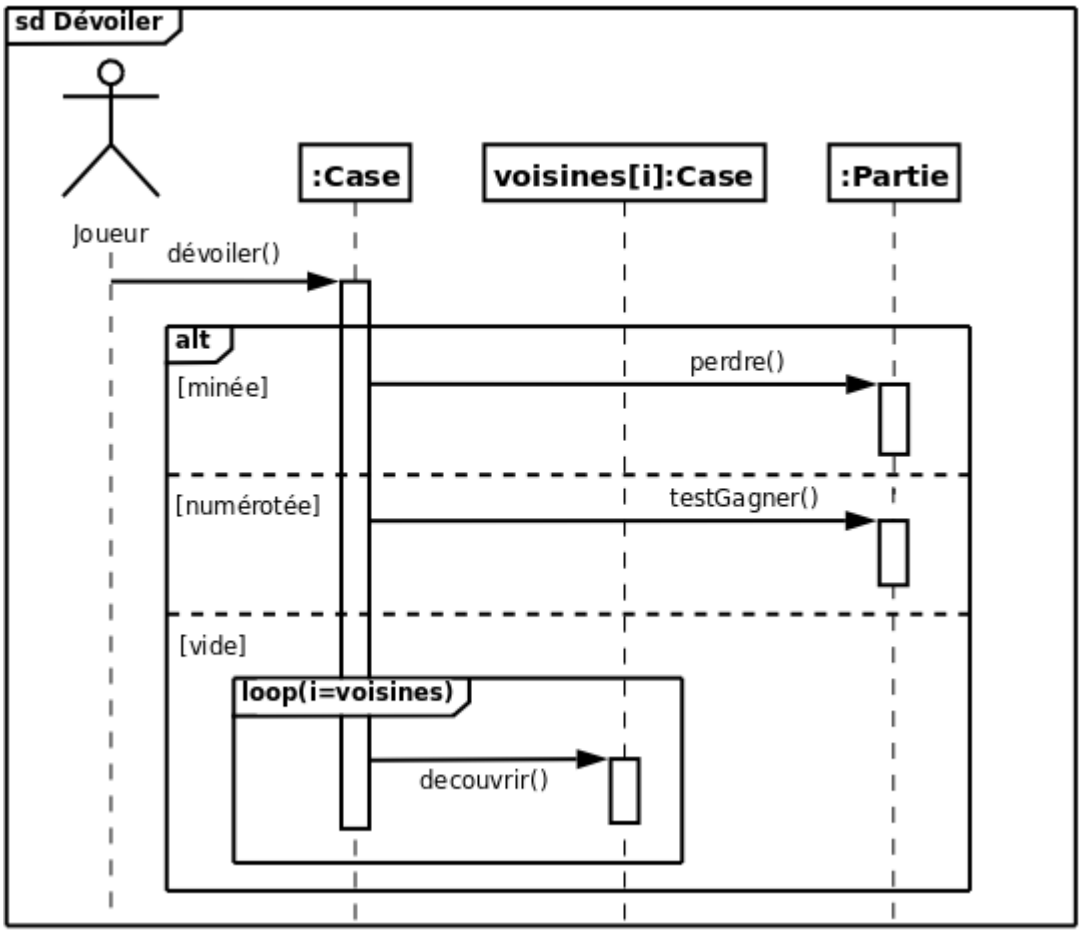
Un fragment combiné se représente de la même façon qu'une interaction. Il est représenté dans un rectangle dont le coin supérieur gauche contient un pentagone. Dans le pentagone figure le type de la combinaison, appelé *opérateur d'interaction*. Les opérandes d'un opérateur d'interaction sont séparés par une ligne pointillée. Les conditions de choix des opérandes sont données par des expressions booléennes entre crochets ([ ]).

La liste suivante regroupe les opérateurs d'interaction par fonctions :

- les opérateurs de choix et de boucle : **alternative**, **option**, **break** et **loop** ;
- les opérateurs contrôlant l'envoi en parallèle de messages : **parallel** et **critical region** ;
- les opérateurs contrôlant l'envoi de messages : **ignore**, **consider**, **assertion** et **negative** ;
- les opérateurs fixant l'ordre d'envoi des messages : **weak sequencing** , **strict sequencing**.

Nous n'aborderons que quelques-unes de ces interactions dans la suite de cette section.

Opérateur *alt*



Représentation d'un choix dans un diagramme de séquence illustrant le découvrement d'une case au jeu du démineur.

L'opérateur *alternative*, ou *alt*, est un opérateur conditionnel possédant plusieurs opérandes. C'est un peu l'équivalent d'une exécution à choix multiple (condition *switch* en C++). Chaque opérande détient une condition de garde. L'absence de condition de garde implique une condition vraie (*true*). La condition *else* est vraie si aucune autre condition n'est vraie. Exactement un opérande dont la condition est vraie est exécuté. Si plusieur opérandes prennent la valeur vraie, le choix est non déterministe.

Opérateurs *opt*

L'opérateur *option*, ou *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.

Opérateur *loop*

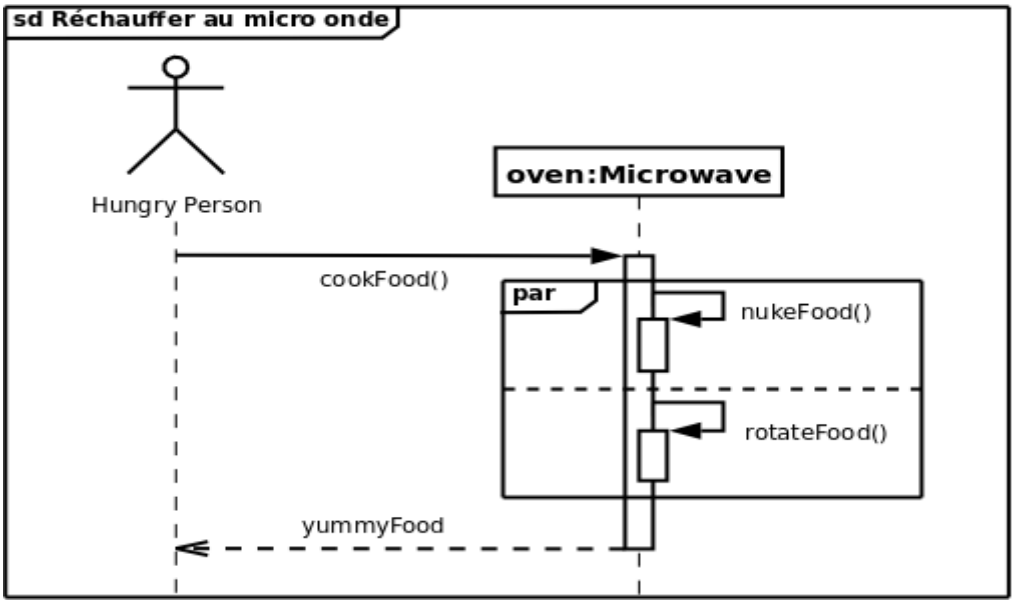
Un fragment combiné de type *loop* possède un sous-fragment et spécifie un compte minimum et maximum (boucle) ainsi qu'une condition de garde.

La syntaxe de la boucle est la suivante :

```
loop[ '('<minInt> [ ',' <maxInt> ] ')' ]
```

La condition de garde est placée entre crochets sur la ligne de vie. La boucle est répétée au moins `minInt` fois avant qu'une éventuelle condition de garde booléenne ne soit testée. Tant que la condition est vraie, la boucle continue, au plus `maxInt` fois. Cette syntaxe peut être remplacée par une indication intelligible comme sur la figure.

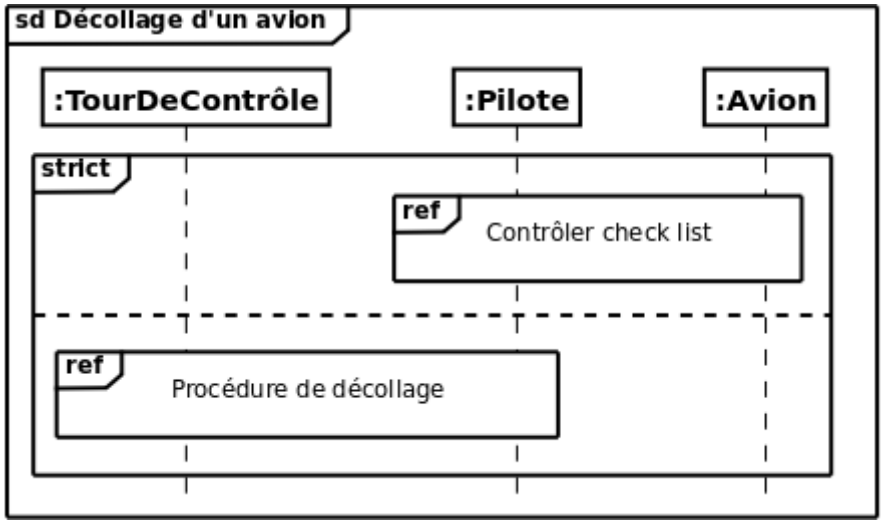
Opérateur *par*



Microwave est un exemple d'objet effectuant deux tâches en parallèle.

Un fragments combiné de type *parallel*, ou *par*, possède au moins deux sous-fragments exécutés simultanément. La concurrence est logique et n'est pas nécessairement physique : les exécutions concurrentes peuvent s'entrelacer sur un même chemin d'exécution dans la pratique.

Opérateur *strict*



Procédures de décollage d'un avion dans l'ordre.

Un fragments combiné de type *strict sequencing*, ou *strict*, possède au moins deux sous-fragments. Ceux-ci s'exécutent selon leur ordre d'apparition au sein du fragment combiné. Ce fragment combiné est utile surtout lorsque deux parties d'un diagramme n'ont pas de ligne de vie en commun.

4 Utilisation d'interaction (*interaction use*)

Il est possible de faire référence à une interaction (on appelle cela une *utilisation d'interaction*) dans la définition d'une autre interaction. Comme pour toute référence modulaire, cela permet la réutilisation d'une définition dans de nombreux contextes différents.

Lorsqu'une utilisation d'interaction s'exécute, elle produit le même effet que l'exécution d'une interaction référencée avec la substitution des arguments fournie dans le cadre de l'utilisation de l'interaction. L'utilisation de l'interaction doit couvrir toutes les lignes de vie qui apparaissent dans l'interaction référencée. L'interaction référencée ne peut ajouter des lignes de vie que si elles ont lieu en son sein.

EXERCICE

Gestion d'accès aux salles d'un bâtiment, accès par badge

Le but est de protéger un bâtiment en restreignant l'accès à certaines salles.
L'ouverture de chacune des portes de ces salles est commandée par un lecteur de badges placé à proximité.

Les badges qui permettent l'ouverture des portes ne sont délivrés qu'aux personnes qui doivent accéder aux locaux protégés dans l'exercice de leurs fonctions. Les droits d'accès sont alloués entre les groupes de personnes et les groupes de portes, de sorte qu'une personne ou une porte doit toujours être au moins dans un groupe (le sien).

Un groupe de portes peut contenir des portes dispersées dans tout le bâtiment. Une porte donnée ne peut appartenir qu'à un seul groupe de portes.

La même personne peut appartenir à plusieurs groupes, de sorte que ses droits d'accès correspondent à l'union des droits d'accès de chacun des groupes qui la contiennent.

La définition des droits d'accès est effectuée en décrivant pour chaque groupe de personnes les différents groupes de portes qui sont accessibles et sous quelle contrainte horaire. Les droits d'accès sont décrits dans un calendrier annuel qui décrit la situation semaine par semaine. Vu la faible variation des droits dans le temps, un calendrier peut être initialisé au moyen de semaines types qui décrivent une configuration de droits donnée. Le superviseur peut créer autant de semaines type qu'il le désire. Les changements apportés à une semaine sont automatiquement propagés dans tous les calendriers qui utilisent cette semaine type.

Le système de contrôle d'accès doit fonctionner de la manière la plus autonome possible. Un superviseur est responsable de la configuration initiale et de la mise à jour des différentes informations de définition des groupes de personnes et de portes. Un gardien dispose d'un écran de contrôle et est informé des tentatives de passage infructueuses. Les alarmes sont transmises en temps légèrement différé : la mise à jour de l'information sur l'écran de contrôle est effectuée toutes les minutes.

TRAVAIL A FAIRE :

1. Donnez les diagrammes de séquence :
    - Porteur Badge et le système
    - Gardien et le système
    - Superviseur et le système