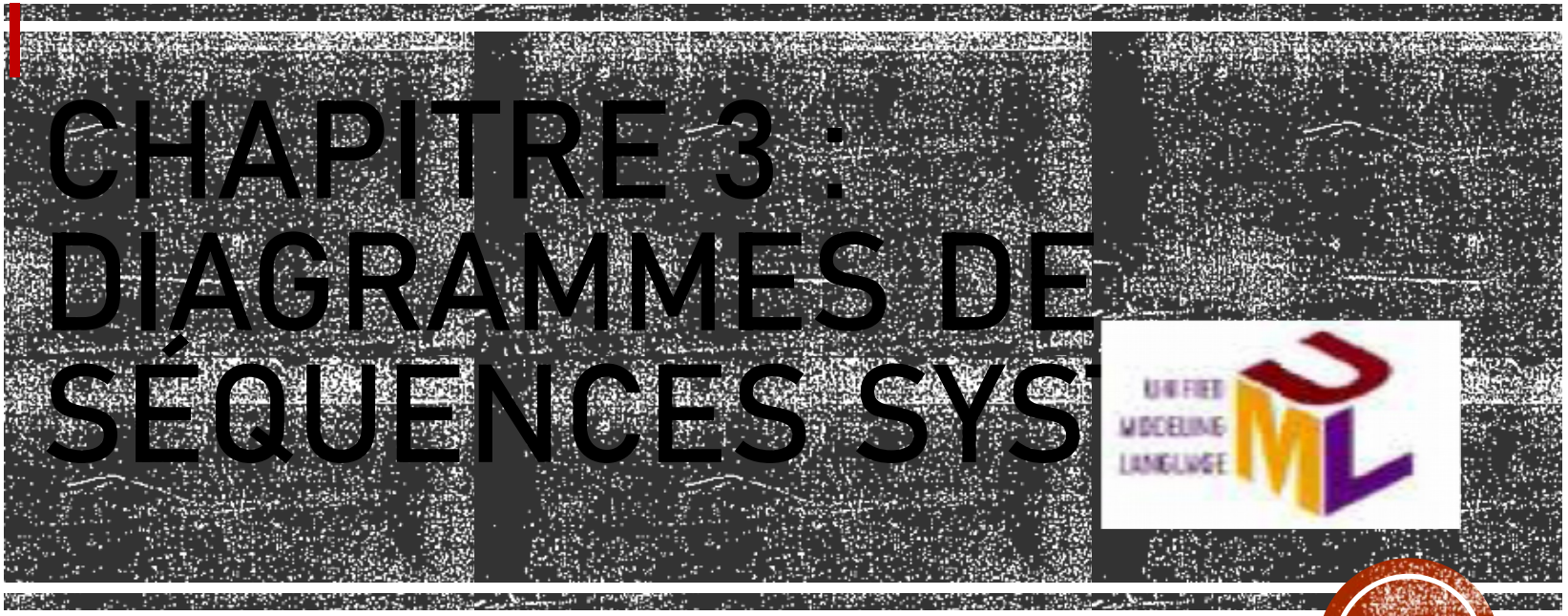
Cours : Analyse Conception des SI



Dr. Ilhem ABDELHEDI ABDELMOULA

Email : ilhemabdelmoula13@gmail.com

Université de Carthage – **Enicarthage** École Nationale des Ingénieurs à Carthage

Département: Informatique **Niveau : 1ère année / 2ème année**

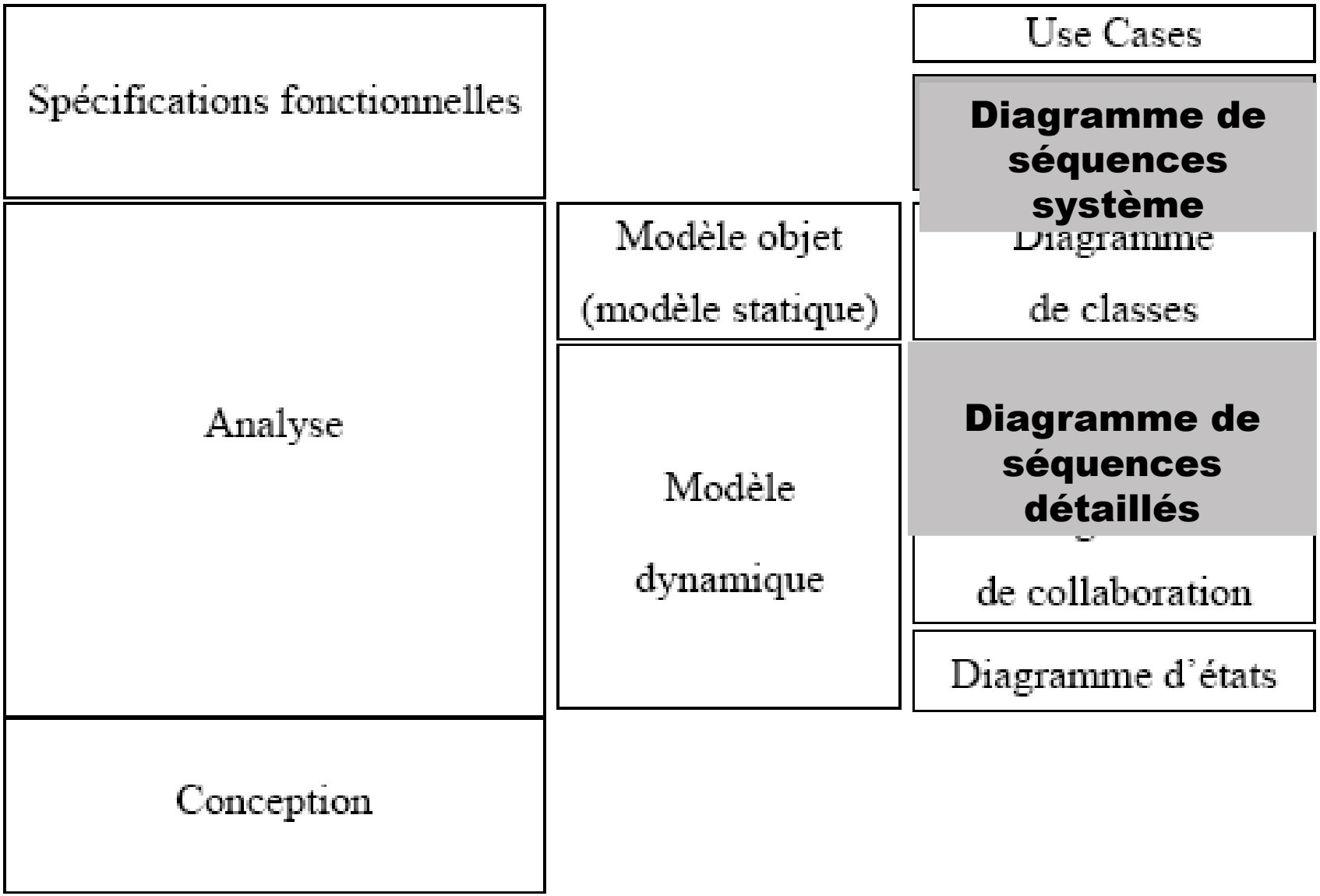
INFO ING **Semestre : 2** **Année universitaire: 2019 - 2020**

UTILITÉ DES DIAGRAMMES D'INTERACTION

- Présentent la vue dynamique d'un système.
- Représentent les interactions entre un ensemble d'objets et leurs relations, y compris les messages qu'ils peuvent échanger.

1. Diagramme de *séquence* mettant l'accent sur le **classement chronologique des messages**
2. Diagramme de *collaboration* mettant l'accent sur l'**organisation** structurelle des éléments

- isomorphes = l'un peut être transformé en l'autre.


Spécifications fonctionnelles
Analyse
Conception
Modèle objet
(modèle statique)
Modèle
dynamique
Use Cases
Diagramme de séquences système
Diagramme
de classes
Diagramme de séquences détaillés
de collaboration
Diagramme d'états

LES DIAGRAMMES DE SÉQUENCES SYSTÈME

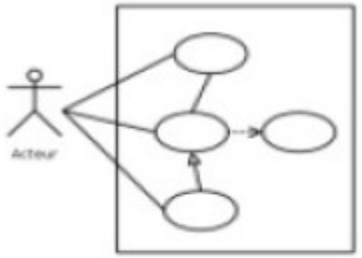
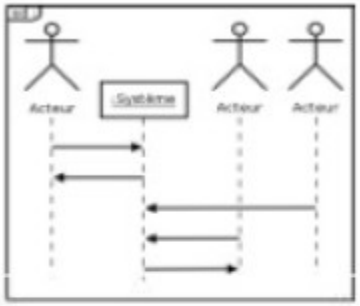


Diagramme de cas d'utilisation



Diagrammes de séquence système

Besoins

Illustrer les scénarii d'utilisation nominal, alternatif ou erreur

une transcription sous une autre ***forme graphique*** *les* ***interactions entre le système et les acteurs ou objets*** *cités dans la spécification textuelle d'un cas d'utilisation.*

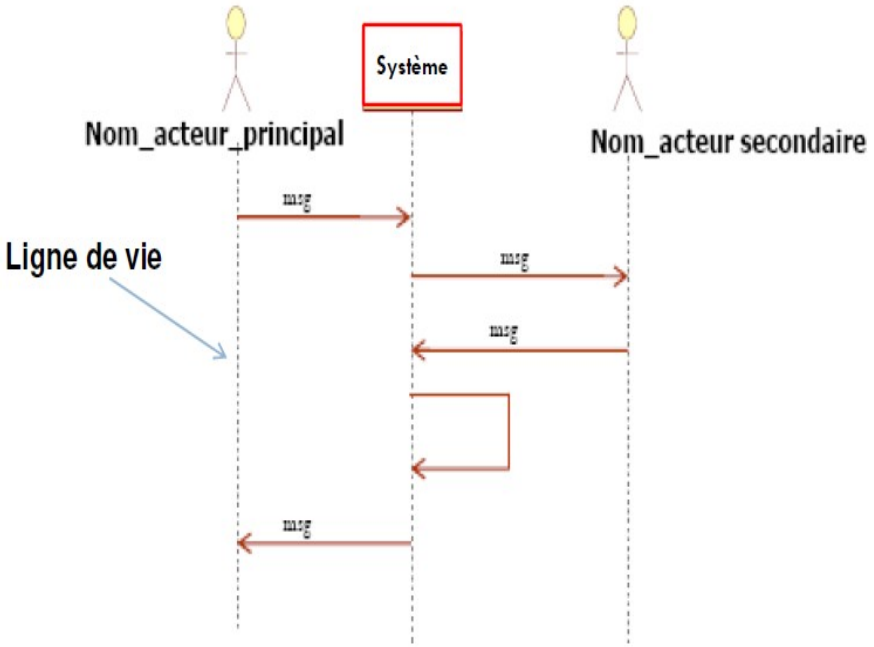
RÔLE DU DIAGRAMME DE SÉQUENCES SYSTÈME

- Système informatique = B**oîte noire** ∀ le scénario d'utilisation (nominal/alternatif/erreur).
- Le comportement du système est décrit **vu de l'extérieur**.
- la boîte noire s'ouvrira en **conception**.
- **On n'y décrit ni le contexte ni l'état des objets**
- Les objets ou acteurs communiquent entre eux par **envoi de messages.** Un objet peut recevoir un événement.

CONSTRUCTION D'UN D.SÉQUENCES SYSTÈME

- Acteur ou objet principal **à gauche,**
- Système au centre comme objet unique
- et les éventuels acteurs ou objets secondaires **à droite**

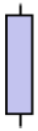
Les interactions entre l'acteur et le système sont marquées par une flèche indiquant la direction du message. Chaque description de l'interaction est écrite au-dessus de la flèche.

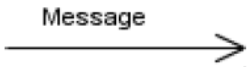


*Un message est une **communication** d'un objet vers un autre objet.*
*La réception d'un message est considérée par l'objet récepteur comme **un événement** qu'il faut traiter (ou pas).*

FORMALISME

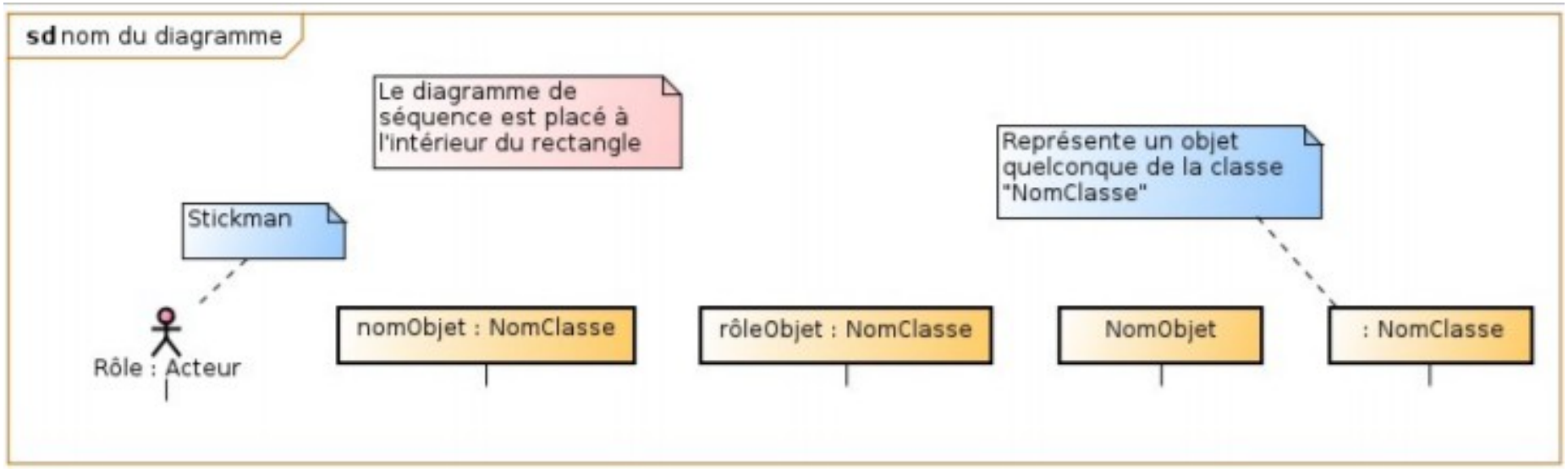
Objet	les objets sont des instances des classes, et sont rangés horizontalement. La représentation graphique pour un objet est similaire à une classe (un rectangle) précédée du nom d'objet (facultatif) et d'un point-virgule (:) .
Acteur	les acteurs peuvent également communiquer avec des objets, ainsi ils peuvent eux aussi être énumérés en colonne. Un acteur est modélisé en utilisant le symbole habituel: *Stickman*.
Ligne de vie	les lignes de vie, *LifeLine*, identifient l'existence de l'objet par rapport au temps. La notation utilisée pour une ligne de vie est une ligne pointillée verticale partant de l'objet.
Activation	les activations, sont modélisées par des boîtes rectangulaires sur la ligne de vie. Elles indiquent quand l'objet effectue une action.
Message	les messages, modélisés par des flèches horizontales entre les activations, indiquent les communications entre les objets.





FORMALISME

- Le D.Seq est placé dans un package qui dispose d'une étiquette **sd en haut à** gauche (**sequence diagram) suivi du nom du diagramme.**
- Il représente les échanges entre les objets mais aussi les échanges avec les acteurs (stickman considéré comme un objet)
- Le nom de l'objet est généralement souligné et peut prendre l'une des quatre formes suivantes :



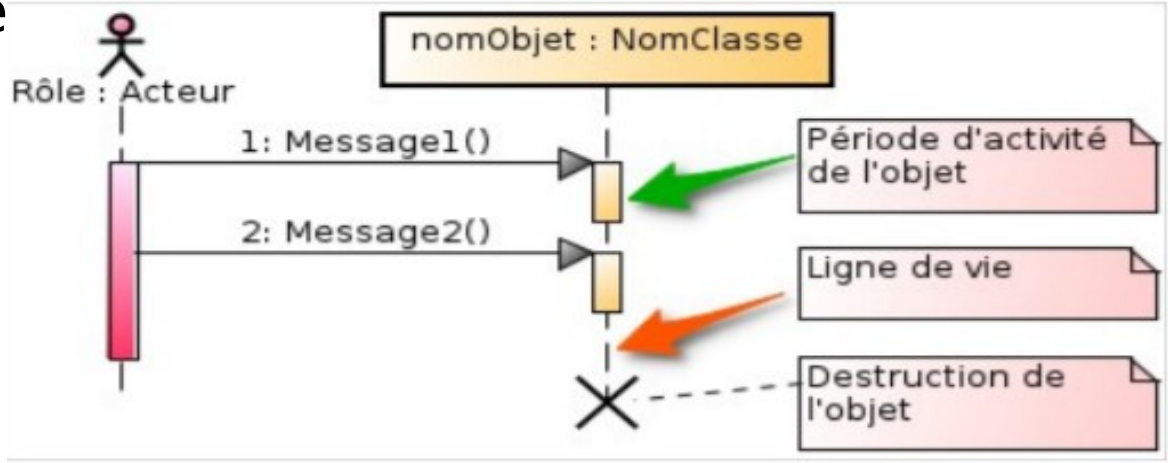
LES OBJETS

1. **Instance d'Acteur**
2. Objet : **instance de classe**
3. **Interface utilisateur** : modélise l'interaction entre le système et les acteurs (ex. abstractions d'écran de saisie, d'affichage, des états, etc. qui peuvent être programmées ou générées par macros commandes).
4. **Contrôle:** concerne la coordination, le séquencement des autres classes (se trouvant entre les entités de classe et interface utilisateur pour éviter un fort couplage entre elles). Ex. à une facture correspond un règlement. La classe contrôle règlement permet de contrôler si le montant de la facture et le montant payé sont égaux ou non.

LIGNE DE VIE D'UN OBJET

- Chaque objet placé dans le DSEQ lui est associé **une ligne de vie** (partant de l'objet en trait pointillés à la verticale): **axe temporel** (le temps s'écoule du haut vers le bas).
  - Elle précise l'existence de l'objet concerné durant une période de temps.
  - Par contre, elle peut débuter et s'interrompre à l'intérieur du diagramme. Lorsque l'objet est détruit, la ligne de vie s'achè

Sens du temps

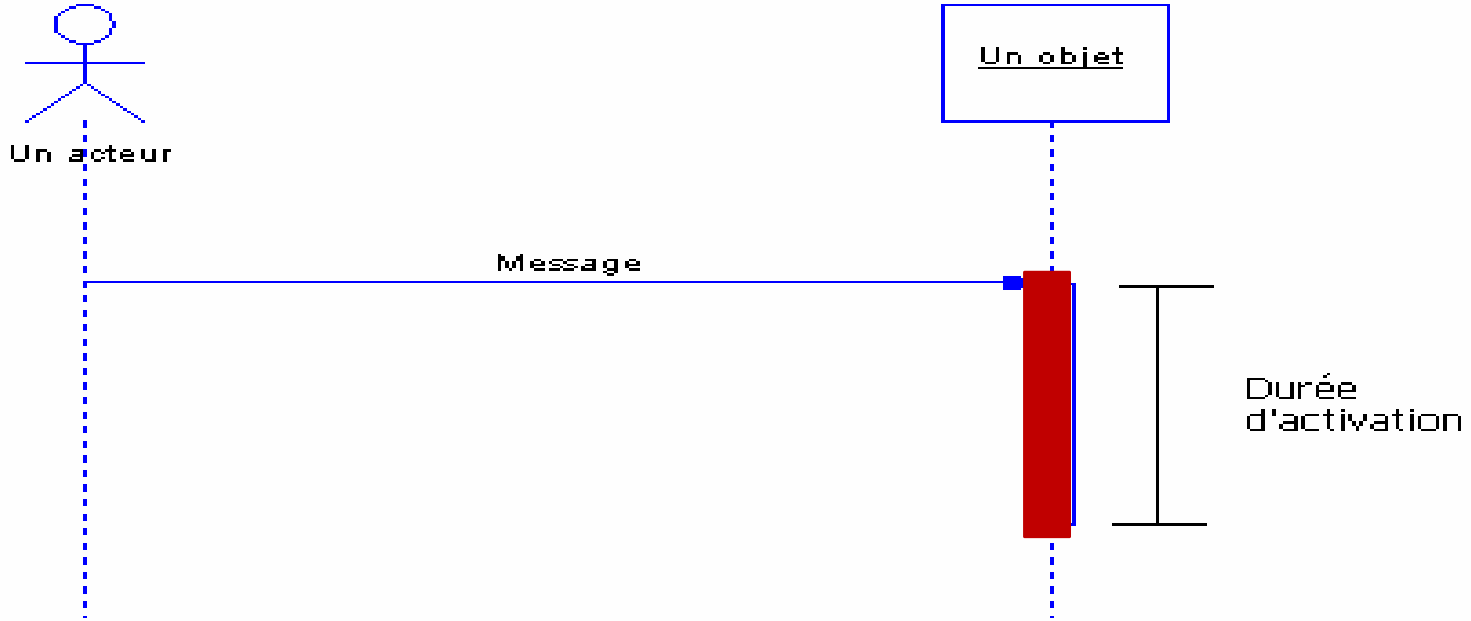


SÉQUENCEMENT DES MESSAGES

- **Le séquencement temporel des messages échangés**
  - *Représenté par les **envois de messages (des flèches horizontales) partant** de la ligne de vie de l'objet **émetteur** vers la ligne de vie de l'objet **récepteur** du message.*
- L'ordre chronologique se déroule du haut **vers le bas** selon un point de **vue temporel**
- **L'ordre des messages** doit suivre la séquence décrite dans le cas d'utilisation.
  - Description textuelle d'un des scénario

LA PÉRIODE D'ACTIVITÉ

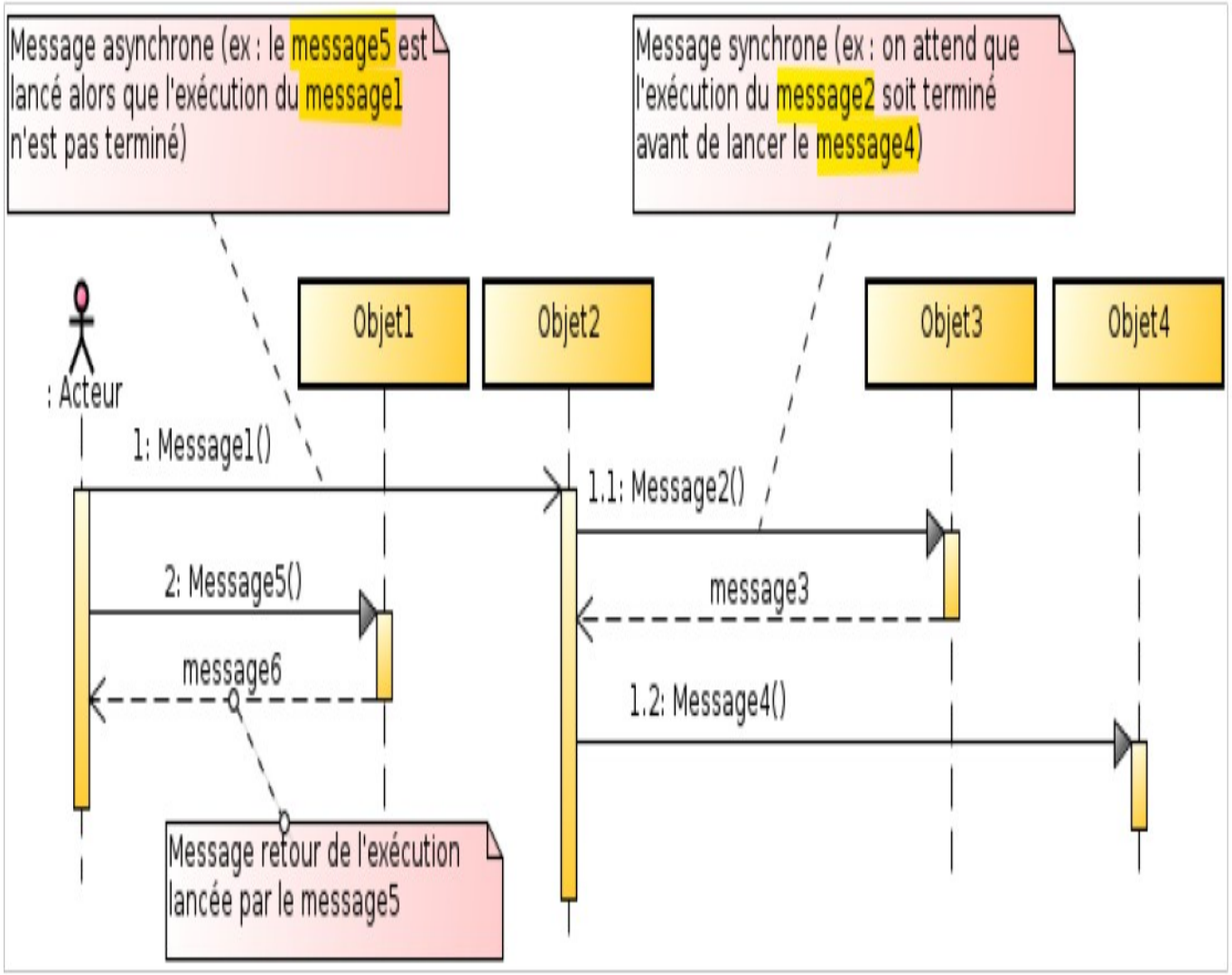
- A la réception d'un message, l'activité de l'objet est déclenchée, **activation ou période d'activité**
  - Durée de temps pendant laquelle un objet effectue une action
  - **Provoque l'exécution d'une méthode chez le récepteur**

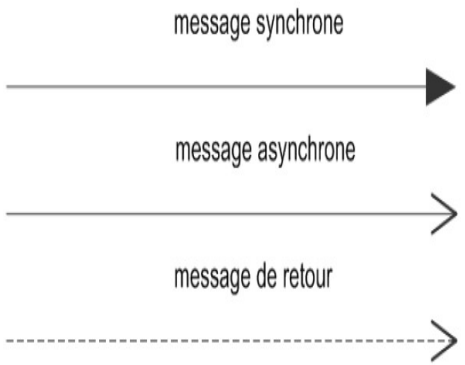


PLUSIEURS TYPES DE MESSAGES

- **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).
- **Le message de retour** (réponse)
  - Si une méthode qui a été activée (par un message) doit retourner des valeurs à la fin de son activation => message retour portant souvent le nom de l'élément retourné.
  - Le message de retour ne provoque pas l'activation de l'objet récepteur
- **La création ou la destruction** d'une instance de classe au cours du cycle principal.

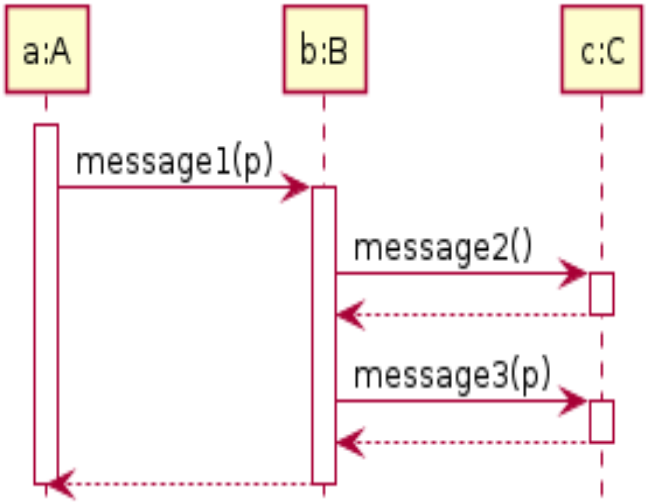
MESSAGE SYNCHRONE, ASYNCHRONE, RETOUR



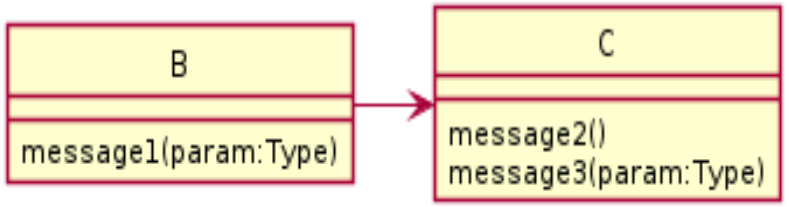


MESSAGE SYNCHRONE

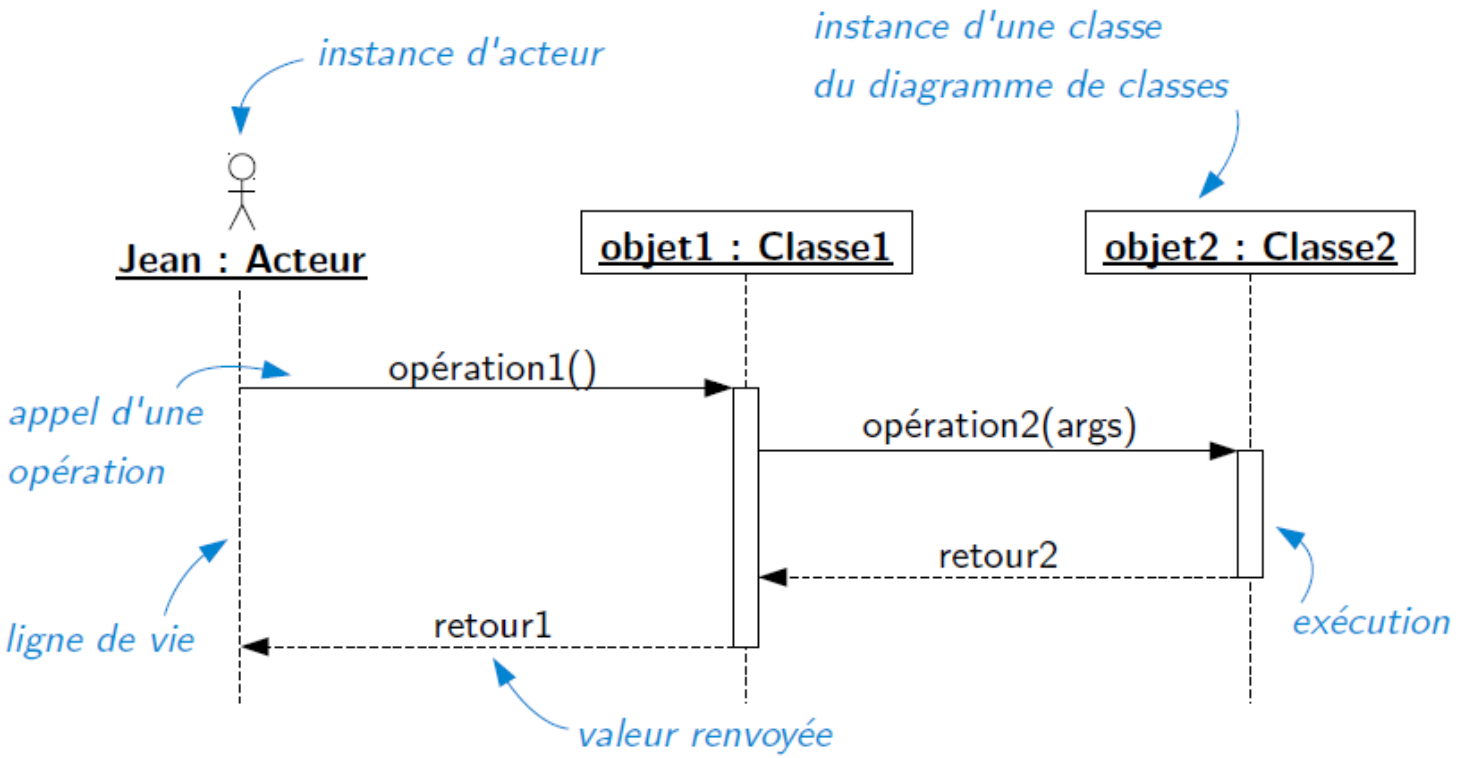
- L'envoi d'une invocation d'**une opération**
  - bloque l'expéditeur jusqu'à la réponse du destinataire.
  - provoque **chez le destinataire** le lancement d'une de ses **méthodes** (portant le même nom que le message), définies dans le **DCL**
  - L'expéditeur attend la réception de la réponse avant de pouvoir lancer un nouveau message.



*On peut associer aux messages **d'appel de méthode** un message de retour (en pointillés) marquant la reprise du contrôle par l'objet émetteur du message synchrone.*

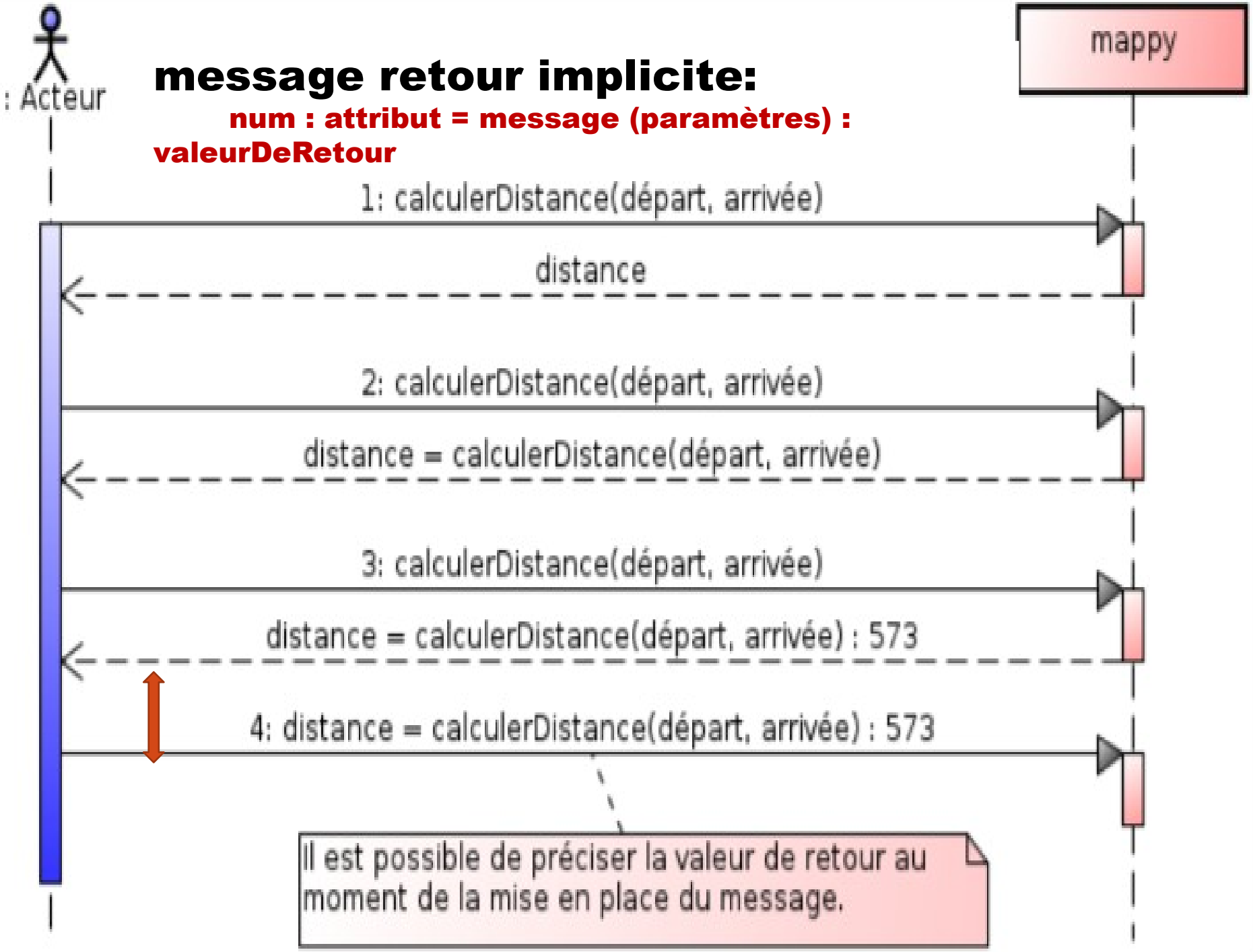


INVOCATION OPÉRATION



message retour implicite:

num : attribut = message (paramètres) : valeurDeRetour

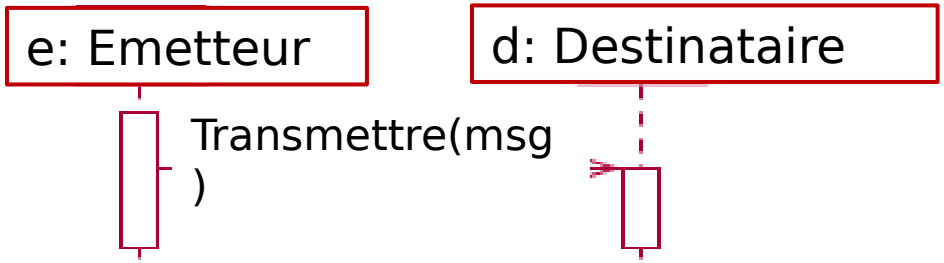


MESSAGE ASYNCHRONE

- L'envoi d'un message asynchrone Courrier (poste, répondeur, messagerie, …)
  - pas bloquant pour l'expéditeur.
  - peut être pris en compte par le récepteur à tout moment ou ignoré.
- Correspond à un **signal** dans le DCL
  - L'objet expéditeur transmet juste une information à l'objet destinataire (acteurs ou périphériques qui envoient des signaux)
  - typiquement utilisé dans la gestion événementielle d'une IHM graphique

MESSAGE ASYNCHRONE

- Exemple : Transmission d'un courrier électronique
- *La réception d'un courrier électronique peut se faire n'importe quand après l'émission. L'émetteur ne reste pas bloqué apres envoi de message*

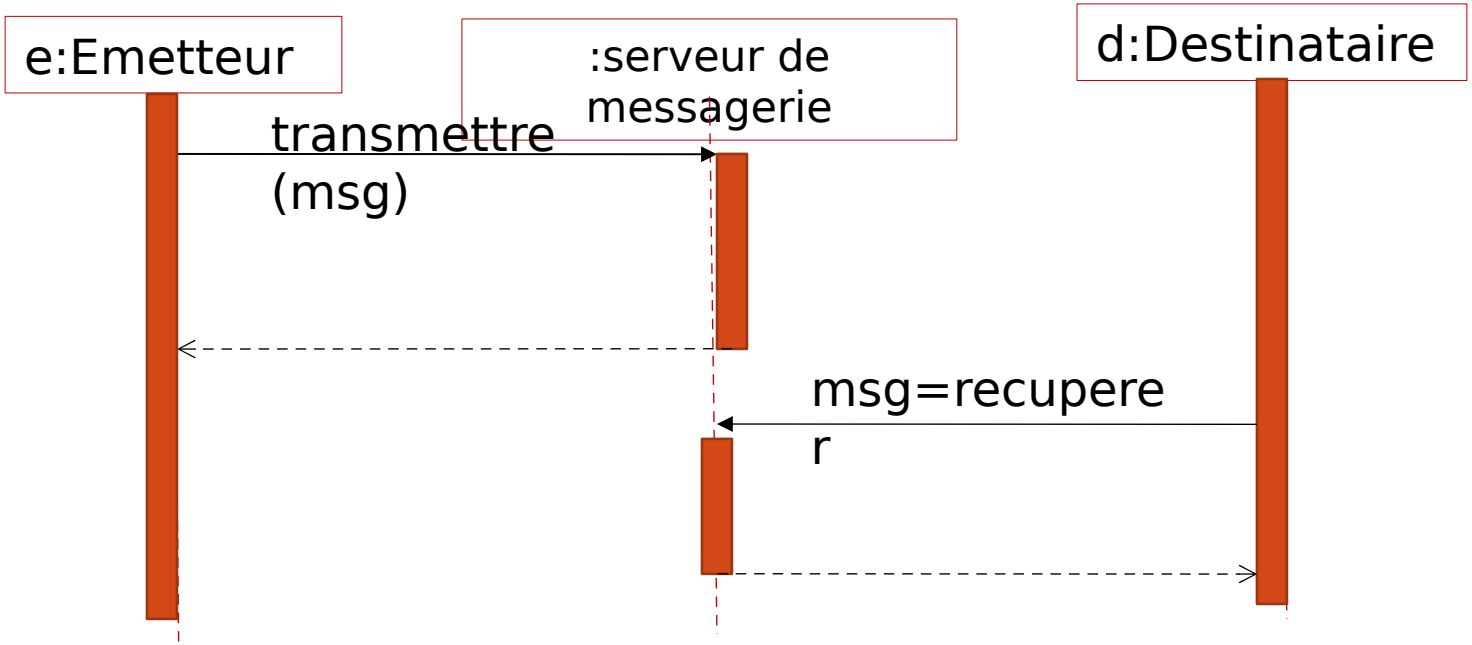


CONTRE EXEMPLE

- Modéliser une transmission d’un courrier électronique via un serveur de messagerie qui réceptionne les messages reçus et les conserve le temps que le destinataire les récupère.

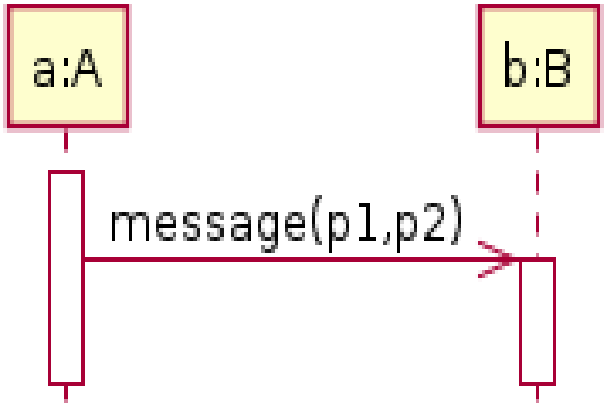
SOLUTION

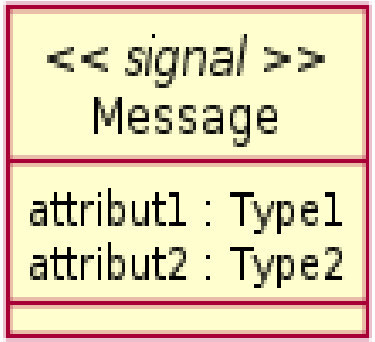
- L'émetteur et le destinataire sont des objets actifs qui sont bloqués le temps d'envoyer ou de récupérer un courrier.



MESSAGE ASYNCHRONE – ENVOI SIGNAUX

- Il peut être parfois un **appel de méthode** : Fréquent dans un **système multi-threads** (multi-tâche) =>Envoi de signaux
- Les signaux sont des objets dont la classe est stéréotypée << signal >> et dont les attributs (porteurs d'information) correspondent aux paramètres du message.
- L'objet expéditeur n'étant pas bloqué pendant l'exécution de la méthode, il peut continuer ainsi à envoyer d'autres messages.



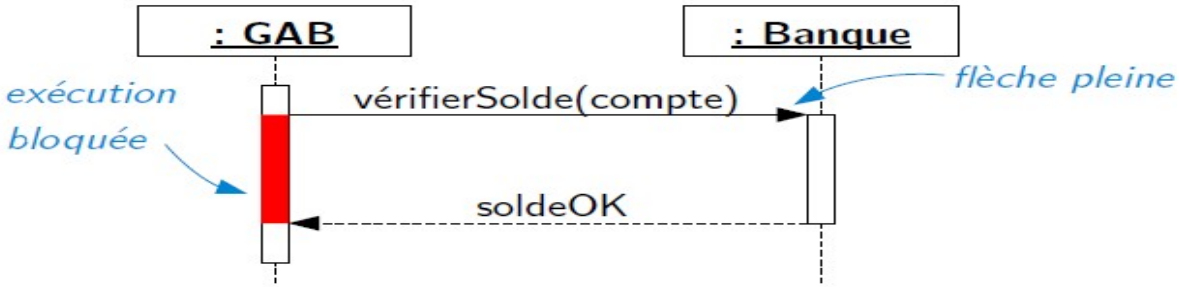


MESSAGE ASYNCHRONE – RETOUR EXPLICITE

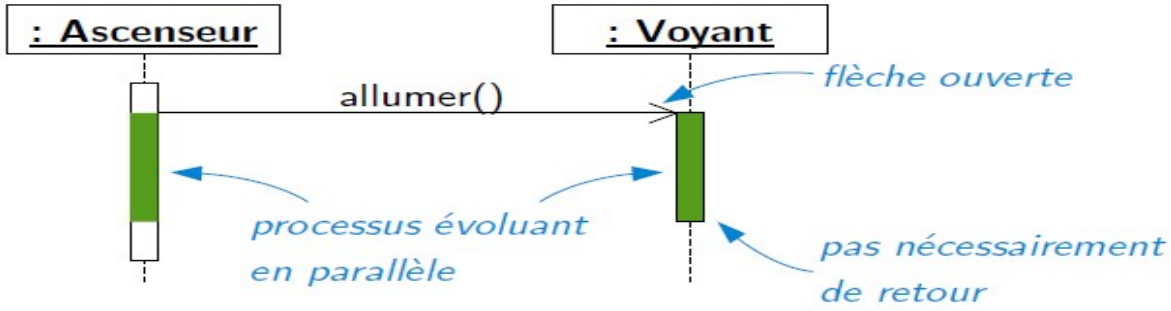
- S’il y a retour, il doit être **explicite**.
  - sinon l’exécution de la méthode lancée par le message asynchrone ne doit rien retourner, il est évident que nous ne devons pas représenter le message de retour.

MESSAGE SYNCHRONE/ASYNCHRONE - EXEMPLES

Message synchrone : Émetteur bloqué en attente du retour

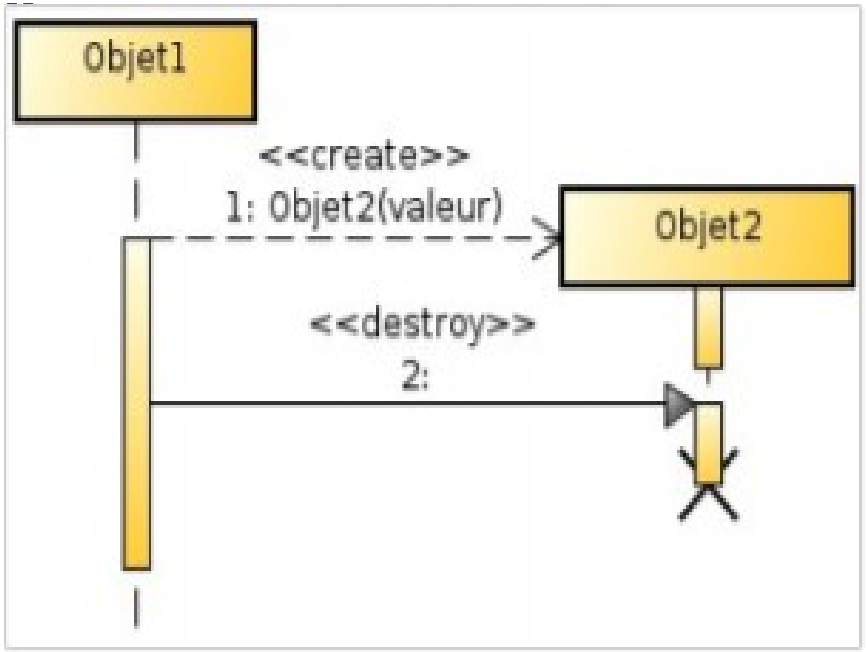


Message asynchrone : Émetteur non bloqué, continue son exécution



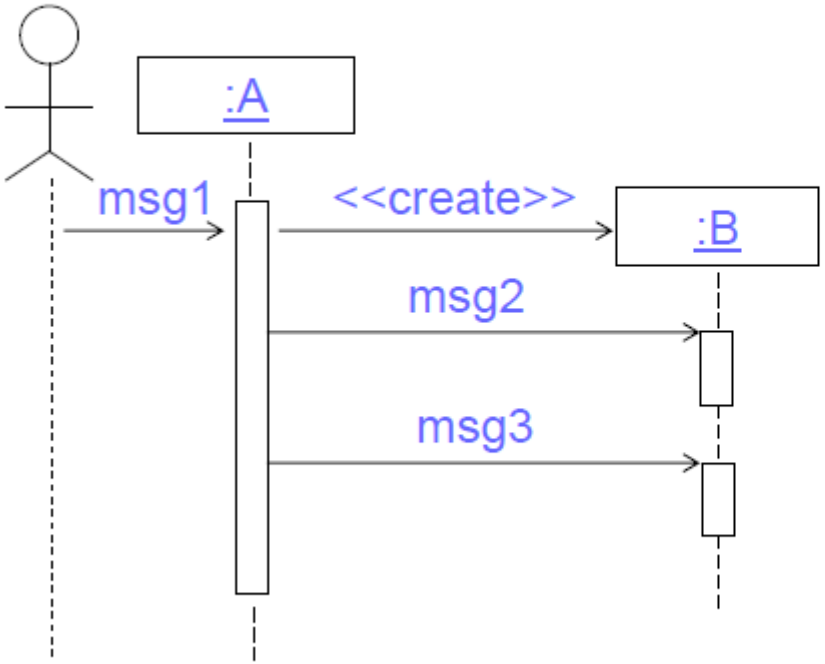
CRÉATION /DESTRUCTION D'OBJETS

- La **création d'un objet** est matérialisée par un message spécifique, **appel d'un constructeur**, accompagné du stéréotype **« Create »** qui pointe sur le début de la ligne de vie de l'objet créé.
- La **destruction d'un objet** est représentée par **une croix** à la fin de sa ligne de vie, à l'aide du stéréotype **« Destroy »**



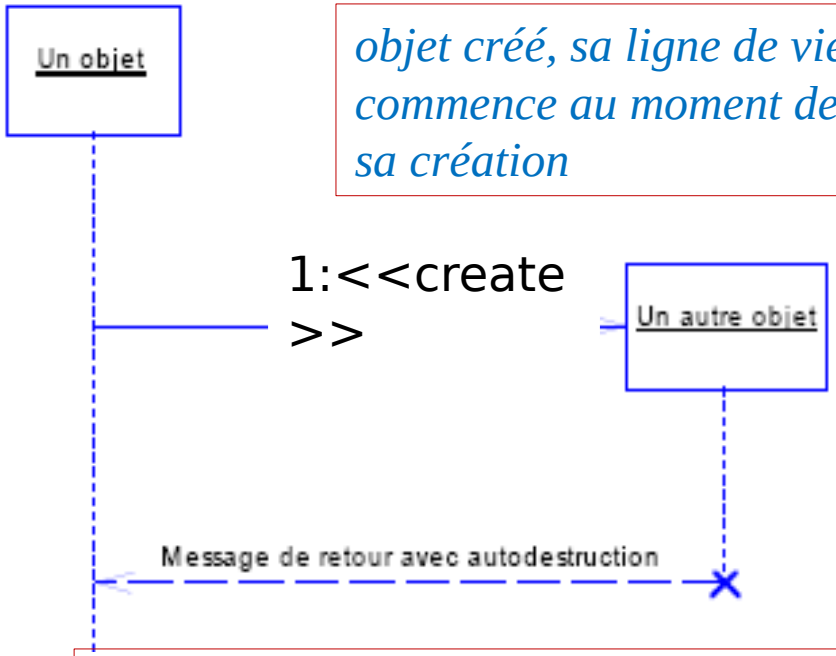
EXEMPLE DE CRÉATION OBJET – CODE

- le diagramme de séquence et le code correspondant



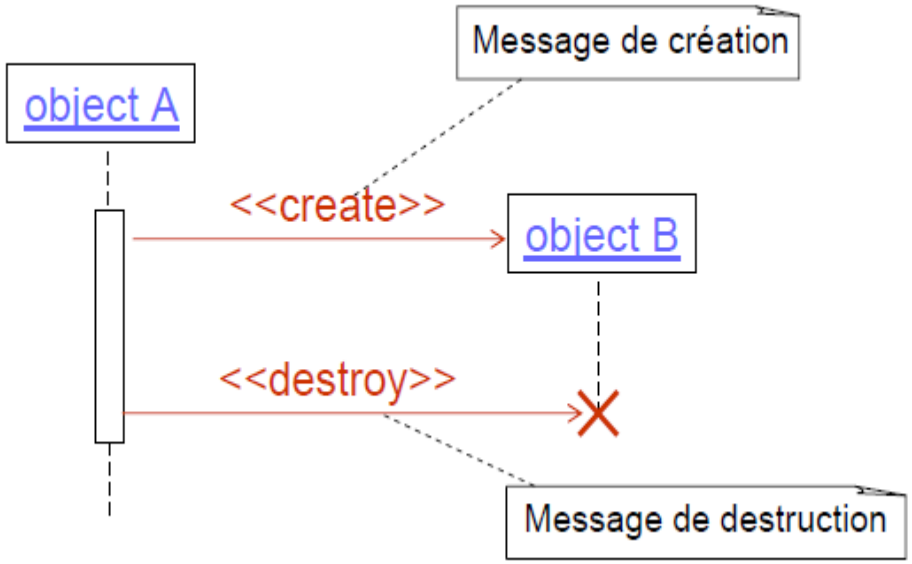

```
public class A
{
  private B objB;
  public void msg1()
  {
     objB = new B();
     objB.msg2();
     objB.msg3();
  }
}

public class B
{
  ...
  public void msg2() { ... }
  public void msg3() { ... }
}
```

EXEMPLE DE DESTRUCTIONS



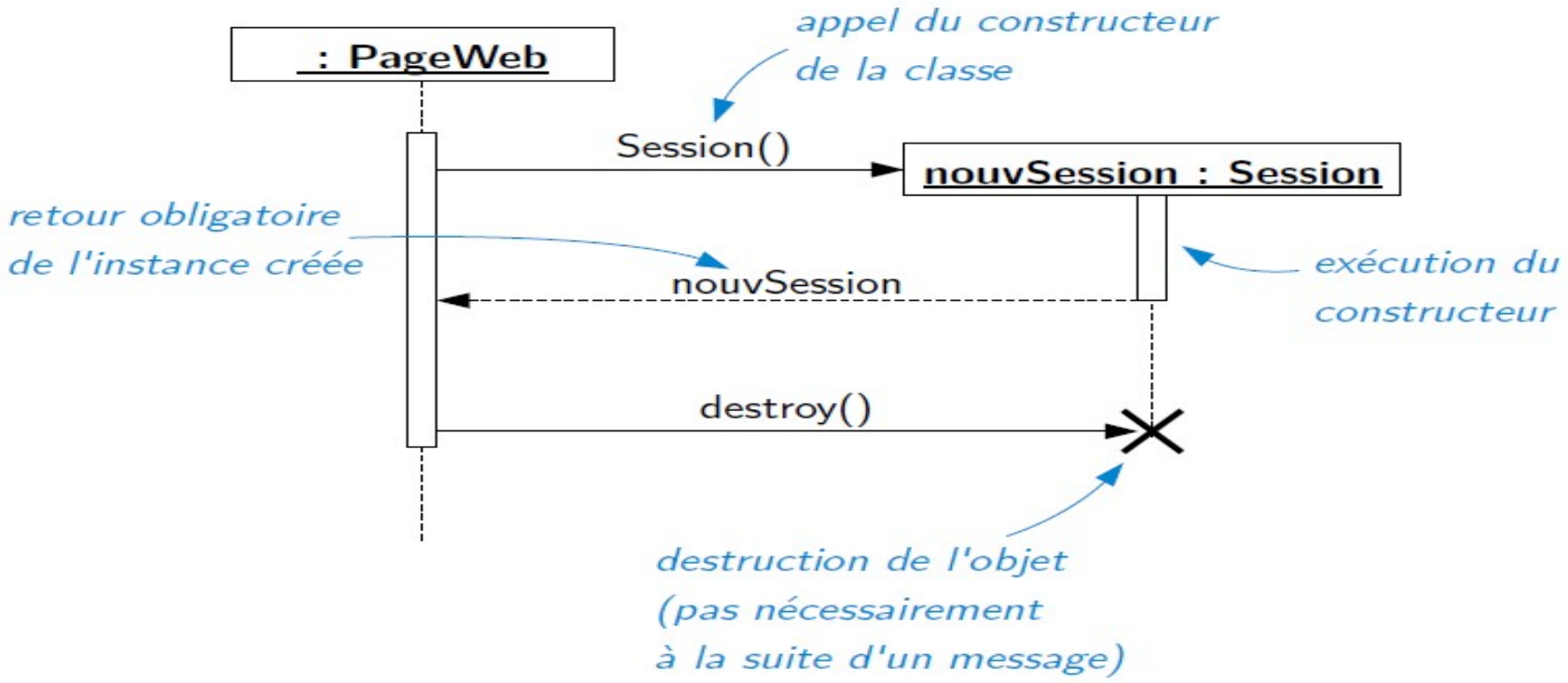
objet créé, sa ligne de vie commence au moment de sa création

La ligne de vie se termine après le dernier message envoyé par un objet qui se suicide.



Objet se détruit et sa ligne de vie se termine à la réception du message de destruction <<destroy>>

CRÉATION/DESTRUCTION OBJET



SYNTAXE DES MESSAGES

- La réception d'un message est suivie généralement de l'exécution de la méthode spécifiée dans le message.
  - Elle peut recevoir des arguments à partir du message reçu.
- Un message est défini par :
  - Son nom (nom de la méthode appelée ou du signal envoyé).
  - Son numéro (séparé du nom du message par 2 point " : ").
    - La numérotation s'effectue séquentiellement à partir de 1.
  - Les paramètres passés à la méthode ou au signal (entre parenthèses après le nom du message).

NUMÉROTATION DES MESSAGES

- Aspect temporel est modélisé par numérotation des messages
  - 1, 2, 3, 4 : **Numérotation simple => séquencement** des messages
  - 1, 1.1, 1.2, 1.2.1, 1.2.2, 1.2.3 : séquencement + un point : ne peut être terminé que si ses sous points le sont aussi
  - 1, 1.1a, 1.1b, 1.2, 1.3 : **Dot notation + concurrence =>** idem dot notation, mais les points 1.1a et 1.1b peuvent être effectués en parallèle

EXEMPLES DE MESSAGES

- l'envoi du message 1.3.5 suit immédiatement celui du message 1.3.4 et ces deux messages font partie du flot 1.3. L'envoi simultané de deux messages 1.3.a et 1.3.b (les indexer par une lettre).

- 3 : bonjour()

Ce message a pour numéro de séquence "3".

- [heure = midi] 1 : manger()

Ce message n'est envoyé que s'il est midi.

- 1.3.6 * : ouvrir()

Ce message est envoyé de manière séquentielle un certain nombre de fois.

EXEMPLES DE MESSAGES

- 3 / *||[i := 1..5] 4 : fermer()

envoi en parallèle de 5 messages, qui ne seront envoyés qu'après l'envoi du message 3.

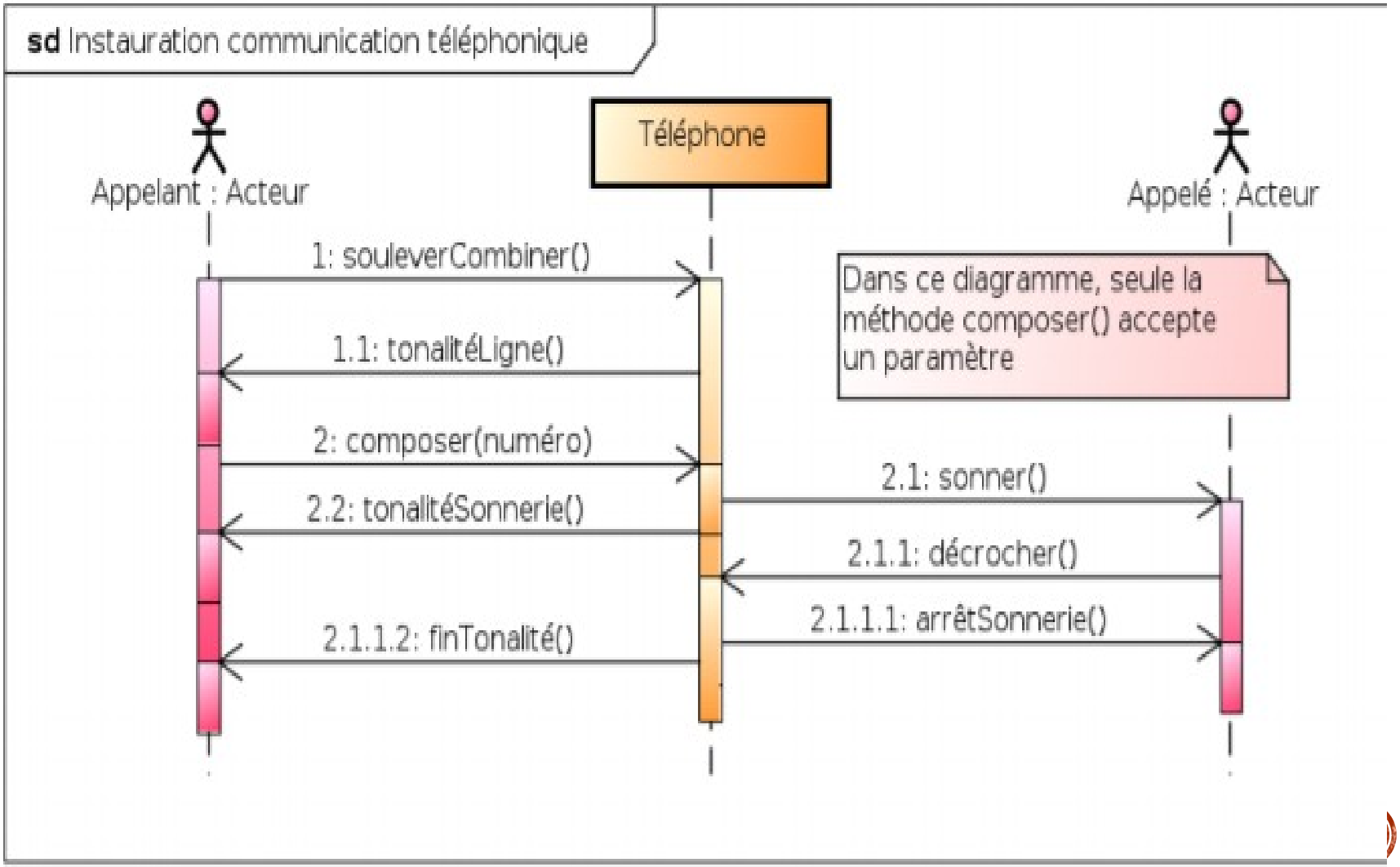
- 1.3,2.1 / [t < 10s] 2.5 : age := demanderAge(nom,prenom)

le message (n° 2.5) ne sera envoyé qu'après les messages 1.3 et 2.1, et que si "t < 10s".

- 1.3 / [disk full] 1.7.a * : deleteTempFiles()
  1.3 / [disk full] 1.7.b : reduceSwapFile(20%)

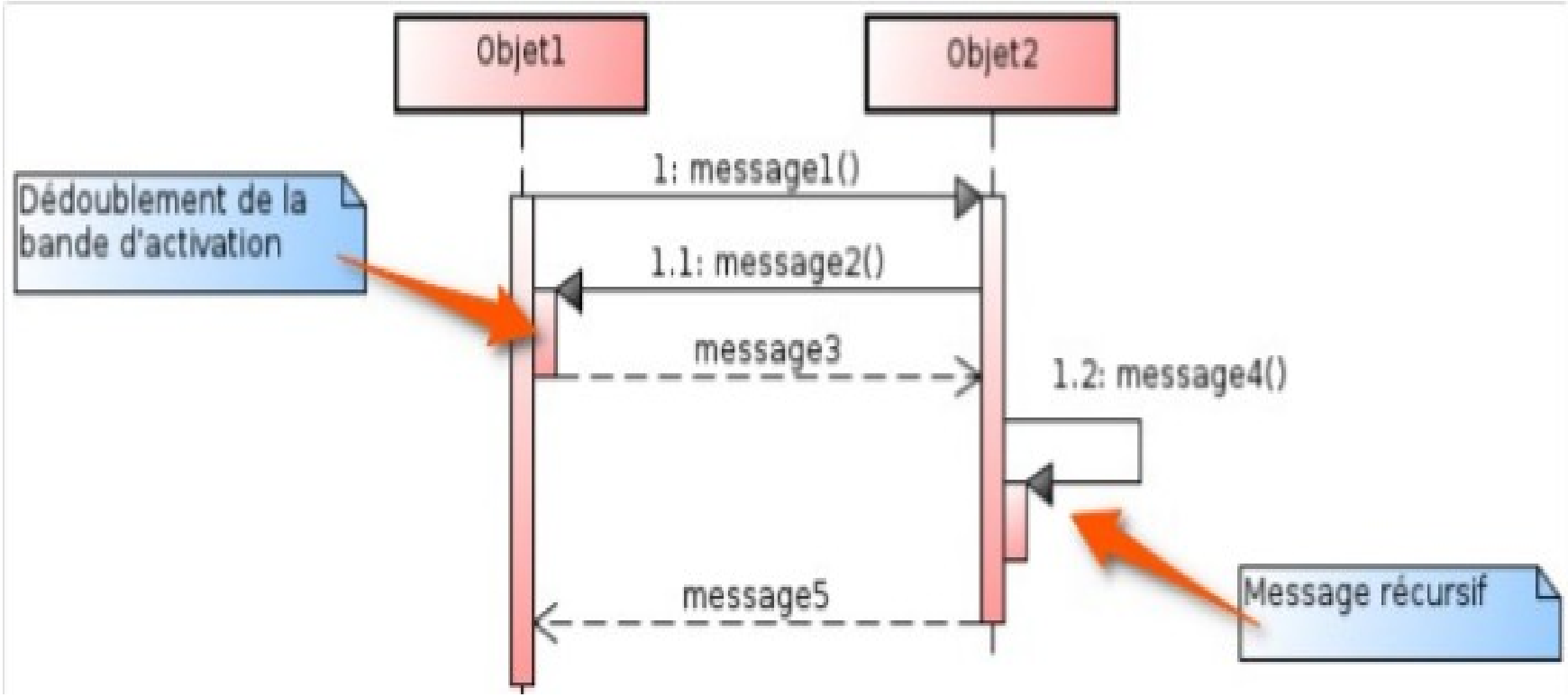
Ces messages ne seront envoyés qu'après l'envoi du message 1.3 et si la condition "disk full" est réalisée. Si tels est le cas, les messages 1.7.a et 1.7.b seront envoyés simultanément. Plusieurs messages 1.7.a peuvent être envoyés.

EXEMPLES

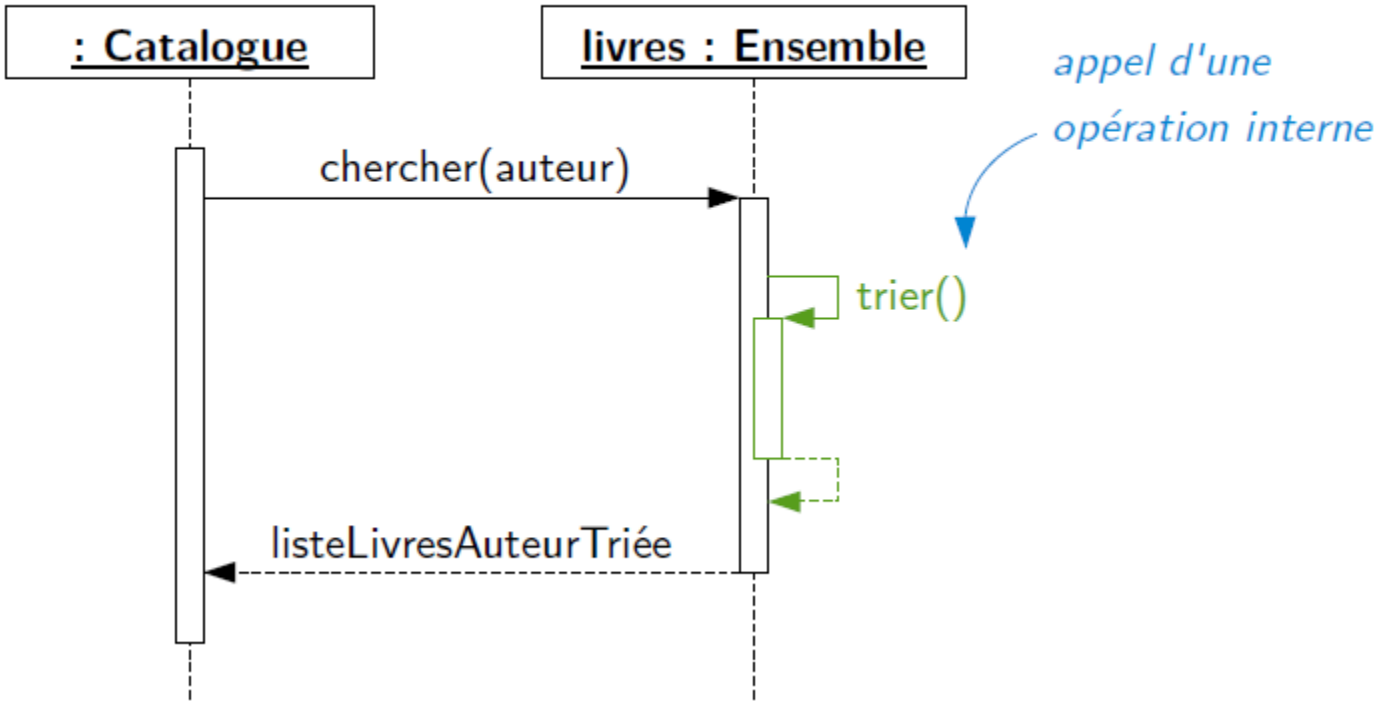

sd Instauration communication téléphonique
Appelant : Acteur
Téléphone
Appelé : Acteur
1: souleverCombiner()
1.1: tonalitéLigne()
2: composer(numéro)
2.2: tonalitéSonnerie()
2.1: sonner()
2.1.1: décrocher()
2.1.1.1: arrêtSonnerie()
2.1.1.2: finTonalité()
Dans ce diagramme, seule la méthode composer() accepte un paramètre

MESSAGES RÉCURSIFS

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

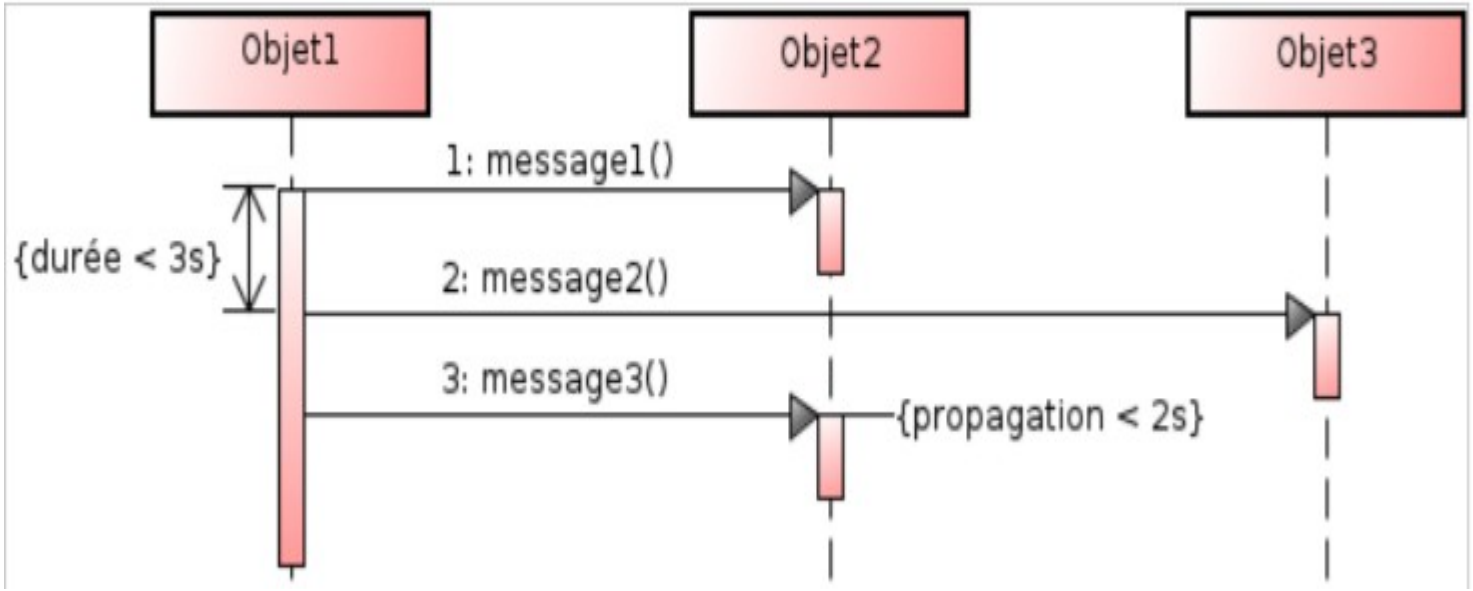


EXEMPLE


: Catalogue
livres : Ensemble
appel d'une
opération interne
chercher(auteur)
trier()
listeLivresAuteurTriée

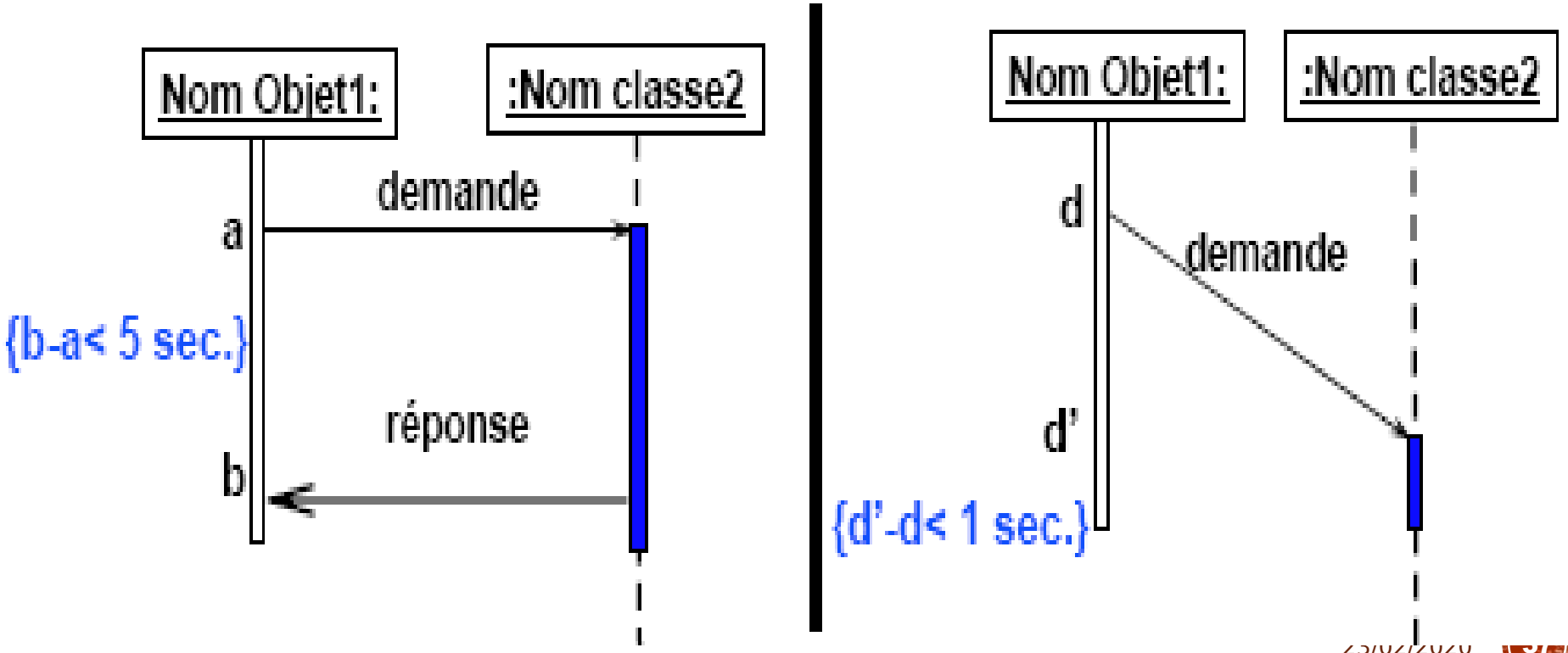
CONTRAINTES TEMPORELLES

- 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.



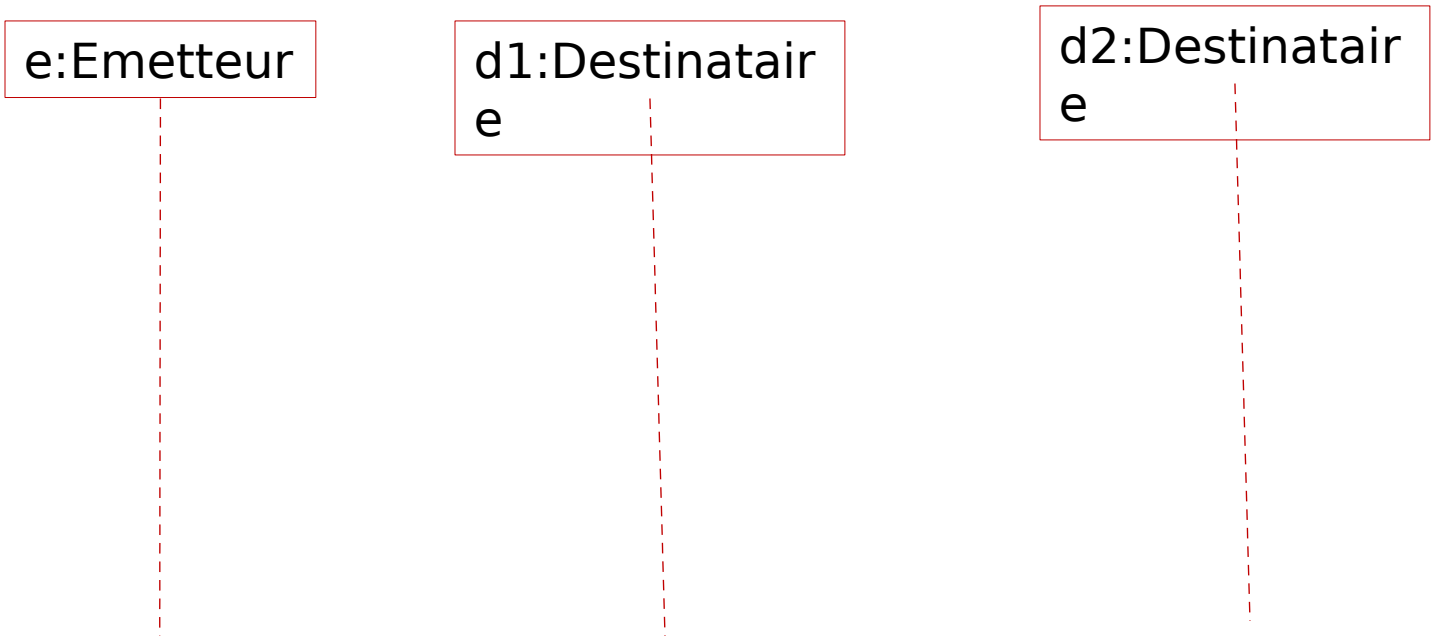
MESSAGE MUNI D'UNE CONTRAINTE TEMPORELLE

- Représenté par une flèche **en oblique** pour matérialiser les délais de transmission non négligeables par rapport à la dynamique générale de l'application.



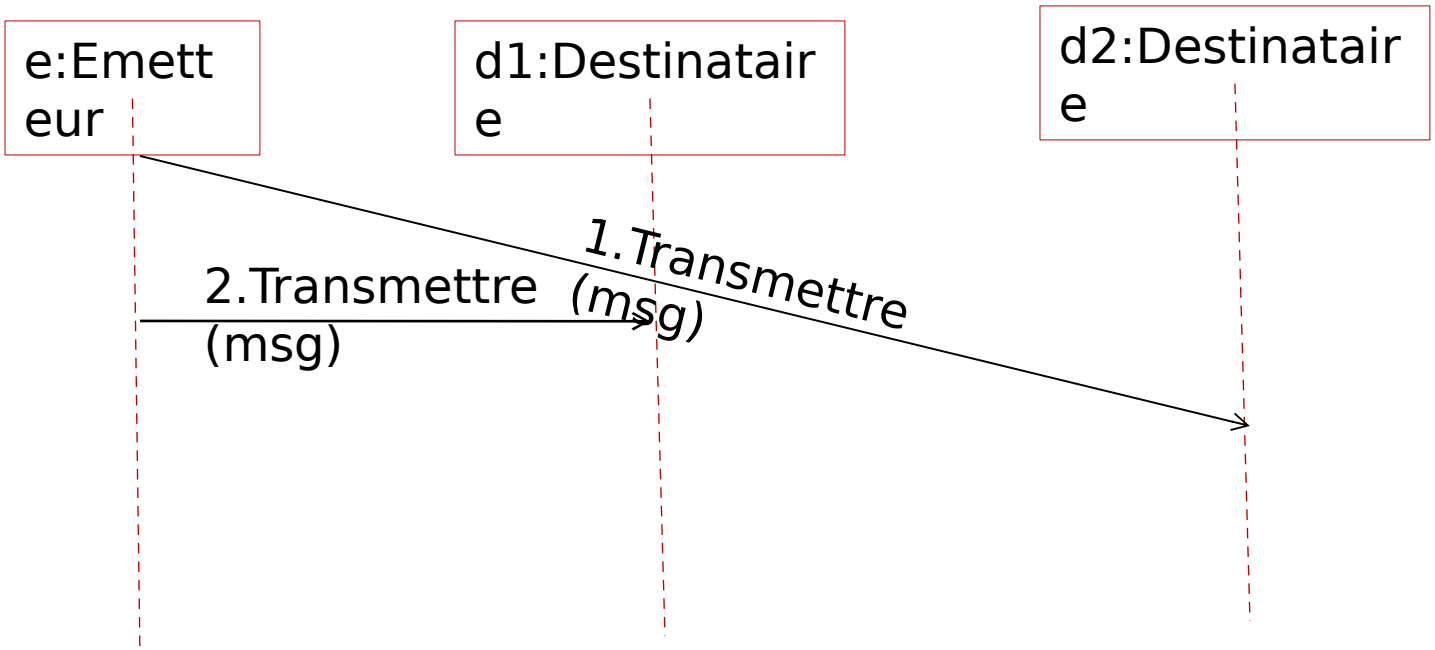
EXERCICE

- Modéliser une transmission d’un courrier électronique à deux destinataires du même émetteur.



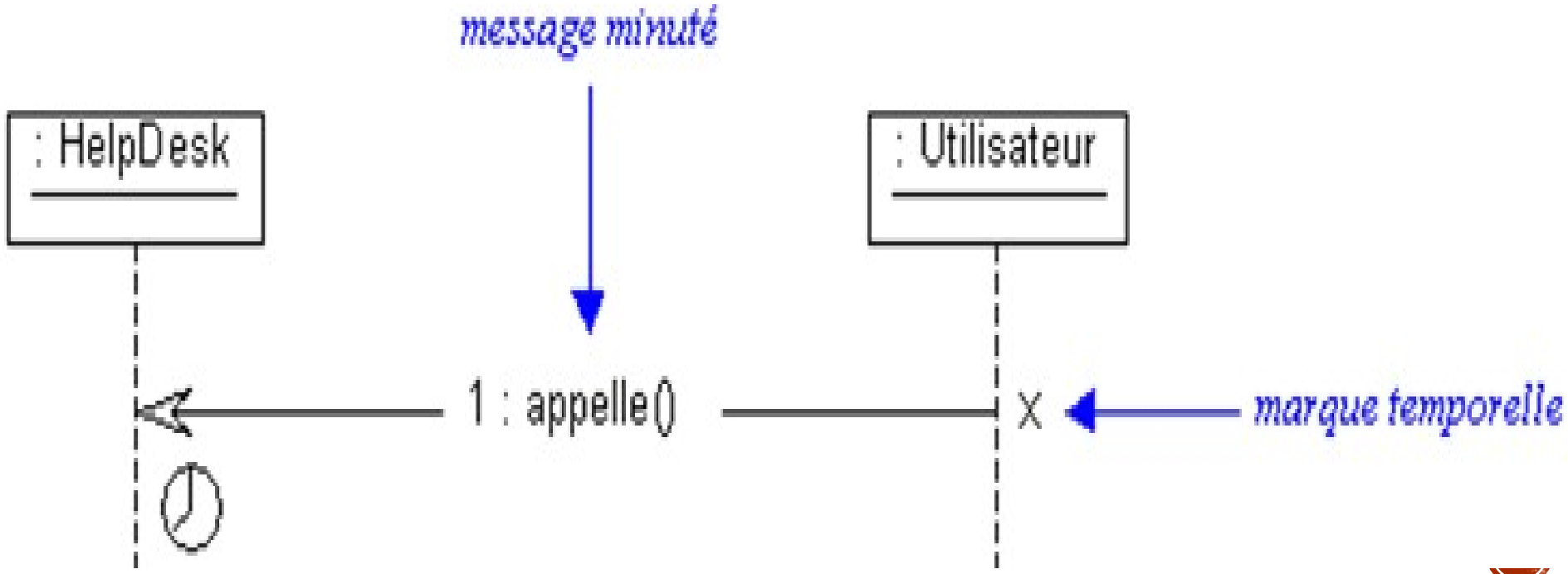
SOLUTION

- Les transmissions sont asynchrones et les messages peuvent être reçus dans un ordre différent de l'ordre d'envoi



MESSAGE MINUTÉ (TIMEOUT)

- L’expéditeur est bloqué pendant le temps alloué au récepteur pour répondre au message
- L'expéditeur est libéré si la prise en compte n'a pas eu lieu pendant le délai spécifié.



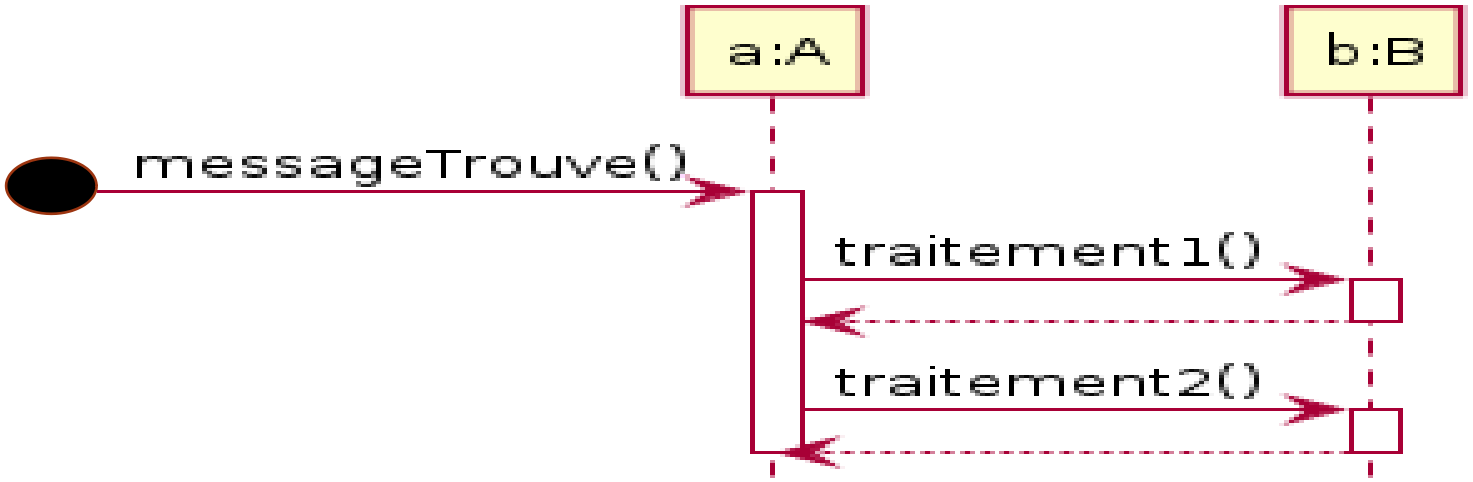
ENVOI DE MESSAGE DÉROBANT

- 2 flots de contrôles = Le destinataire est en attente de messages. L'expéditeur n'attend pas que le destinataire soit disponible
- Ex: Avertissement (alerte, autorisation, …)

Un client fait un prélèvement d'argent à un distributeur de billets. Dès que le distributeur a identifié le client, il distribue l'argent en postulant que le client est bien resté devant l'appareil en attente de son argent.

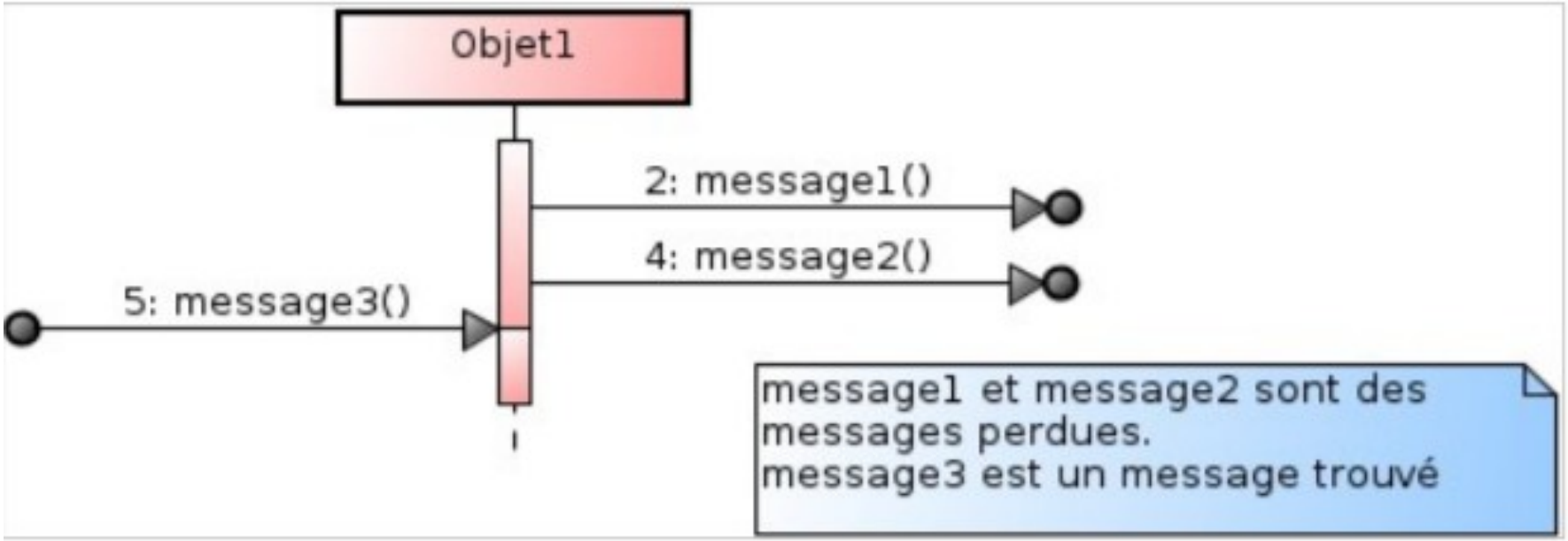
MESSAGES TROUVÉS

- **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.
  - utilisé pour modéliser le comportement d'un élément suite à la réception d'un message d'exception.
  - Les messages trouvés peuvent être synchrones ou asy[illegible]



MESSAGES PERDUS

- à l'inverse des messages trouvés, pour les messages perdus, on connaît l'émetteur mais pas le récepteur.
  - utilisé pour spécifier le résultat d'un message synchrone trouvé.
  - représenté par une flèche partant de la ligne de vie d'un élément vers un disque noir.
- Il peut être synchrone ou asynchrone.
- Utilisé pour modéliser, par exemple, les scenarii de pertes de message sur un réseau.



EXERCICE

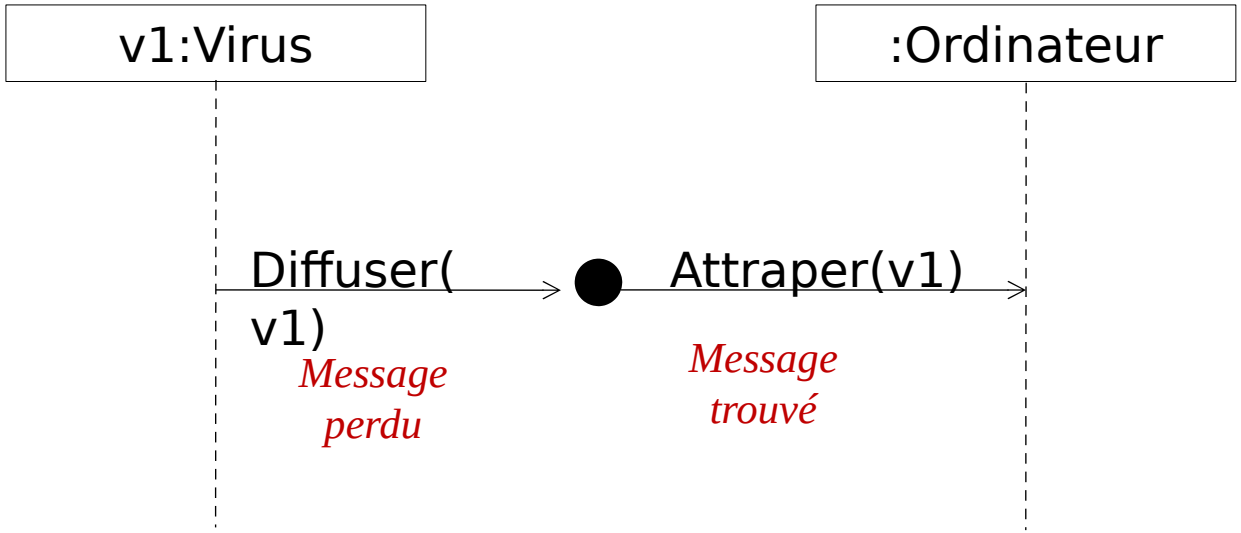
- Modéliser la transmission d'un virus à un ordinateur

La plupart du temps un ordinateur contaminé ne connait pas la source du virus et l'auteur du virus ne souhaite pas être connu

Utilisation de message perdu et trouvé

SOLUTION

- Modéliser la transmission d'un virus à un ordinateur

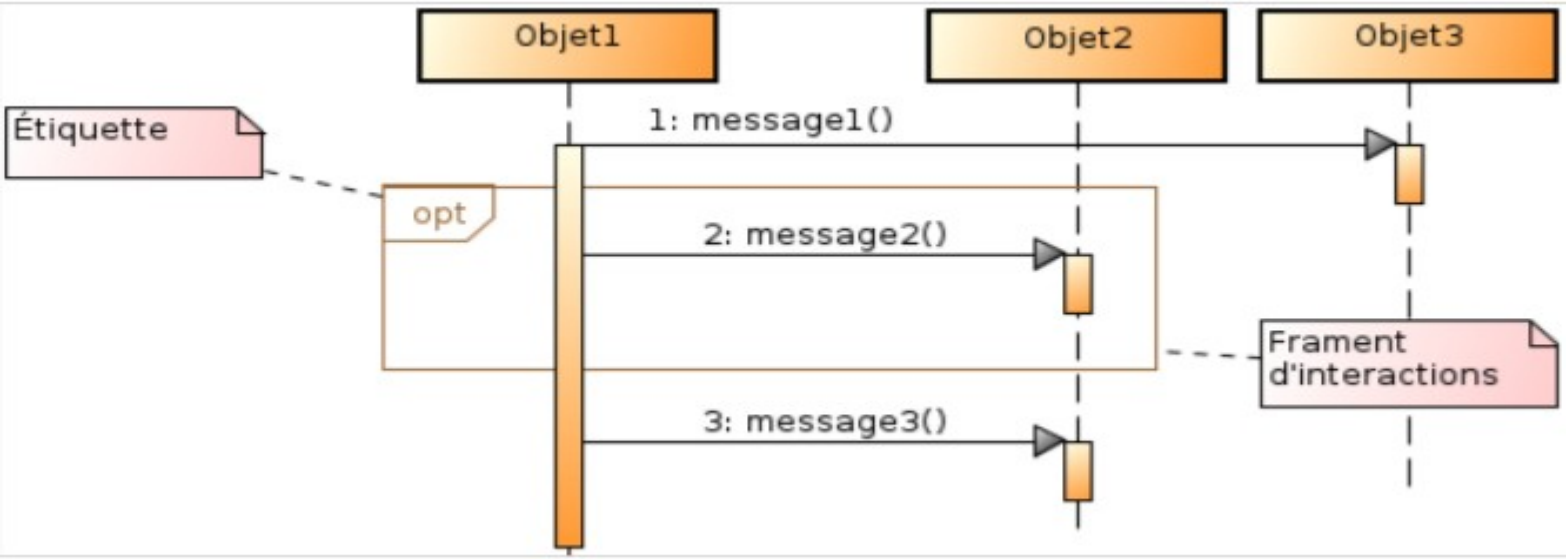


La plupart du temps un ordinateur contaminé ne connait pas la source du virus et l'auteur du virus ne souhaite pas être connu

Utilisation de message perdu et trouvé

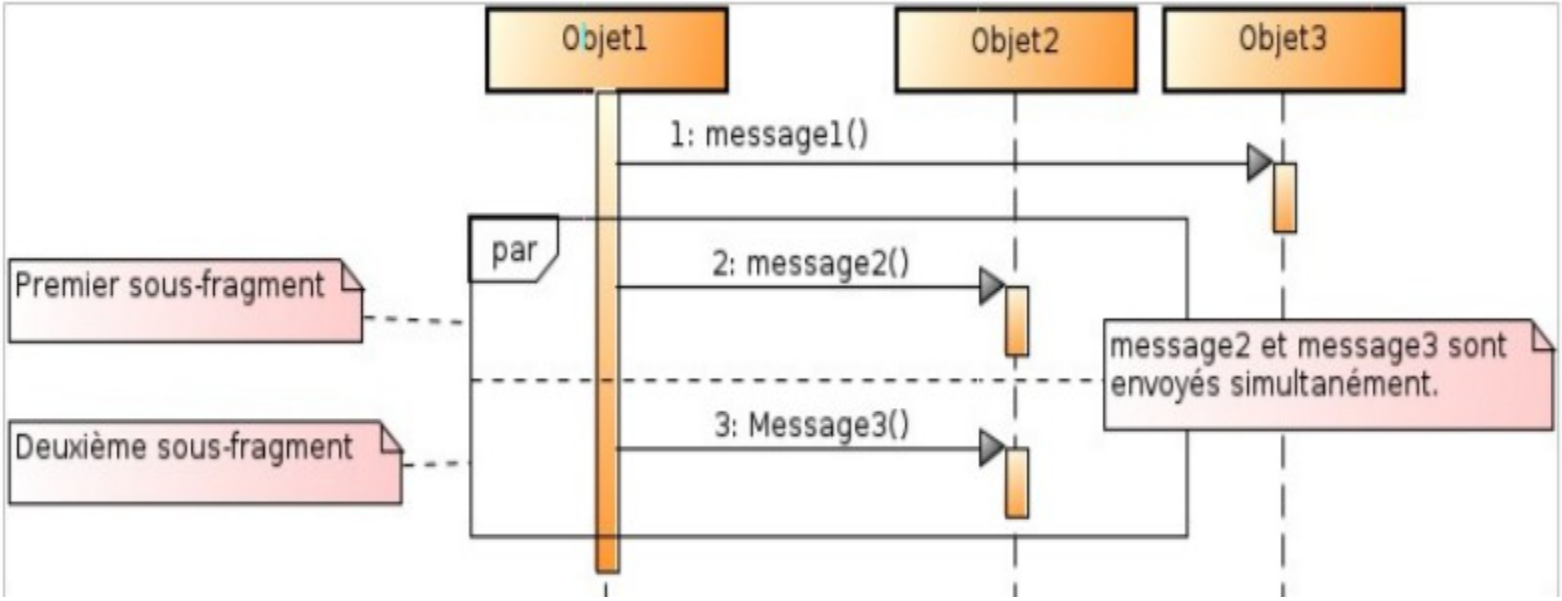
FRAGMENTS D'INTERACTIONS COMBINÉS

- Un fragment d'interactions ou frame 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.



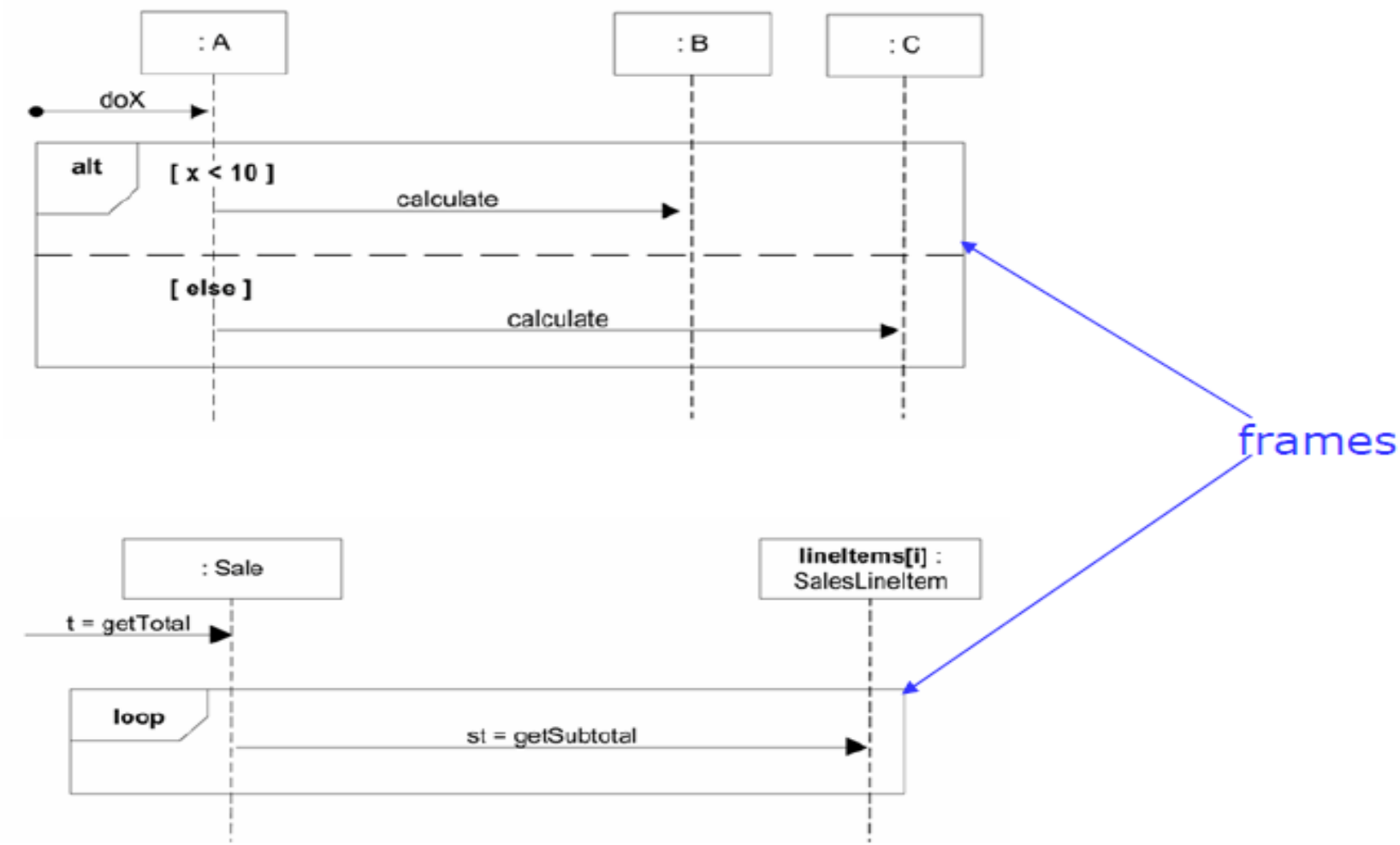
FRAGMENTS D'INTERACTIONS COMBINÉS

- Dans certains cas, 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.**



UTILISATION DES FRAMES

Remarque : les notations UML 2.x permettent d'utiliser les **frames** pour représenter les conditions ou les itérations

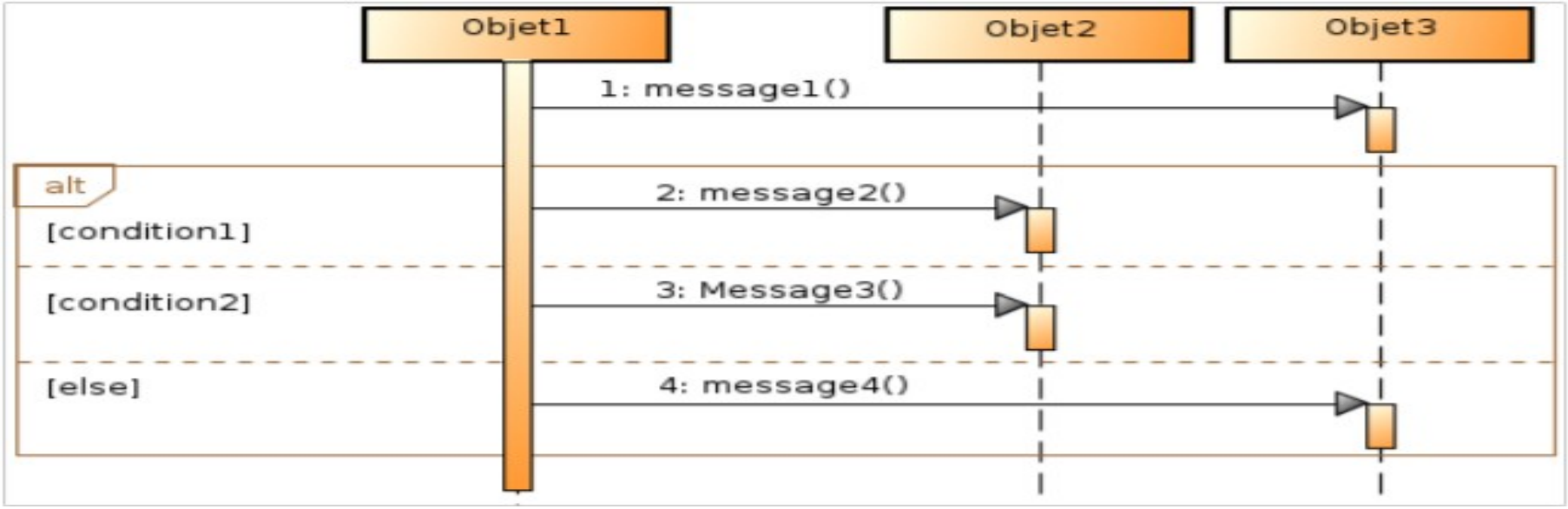


OPÉRATEURS DE FLUX DE CONTRÔLE

- **alt** : Contient une liste des fragments dans lesquels se trouvent d'autres séquences de messages. Une seule séquence peut se produire à la fois.
- **opt** (*facultatif**) : Contient une séquence qui peut ou non se produire. **loop** : Le fragment est répété un certain nombre de fois.
- **break** : scenarii exceptionnel ou de rupture (ex appui sur « Esc ») exécuté si une condition de garde est satisfaite. Si ce fragment est exécuté, le reste de la séquence est abandonné.
- **par** (*parallel*) : Les événements des fragments peuvent être entrelacés.

FRAGMENT D'INTERACTION AVEC OPÉRATEUR « ALT »

- Désigne un choix, une alternative (SI ALORS SINON).
- Contient une liste des fragments dans lesquels se trouvent d'autres séquences de messages. Une seule séquence peut se produire à la fois.
- Plusieurs opérandes séparés par des pointillés : une exécution à choix multiples.
- Les différentes alternatives sont spécifiées dans des zones délimitées par des pointillés et les conditions sont spécifiées entre crochets dans chaque zones.
- On peut utiliser une clause [else].
- 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.

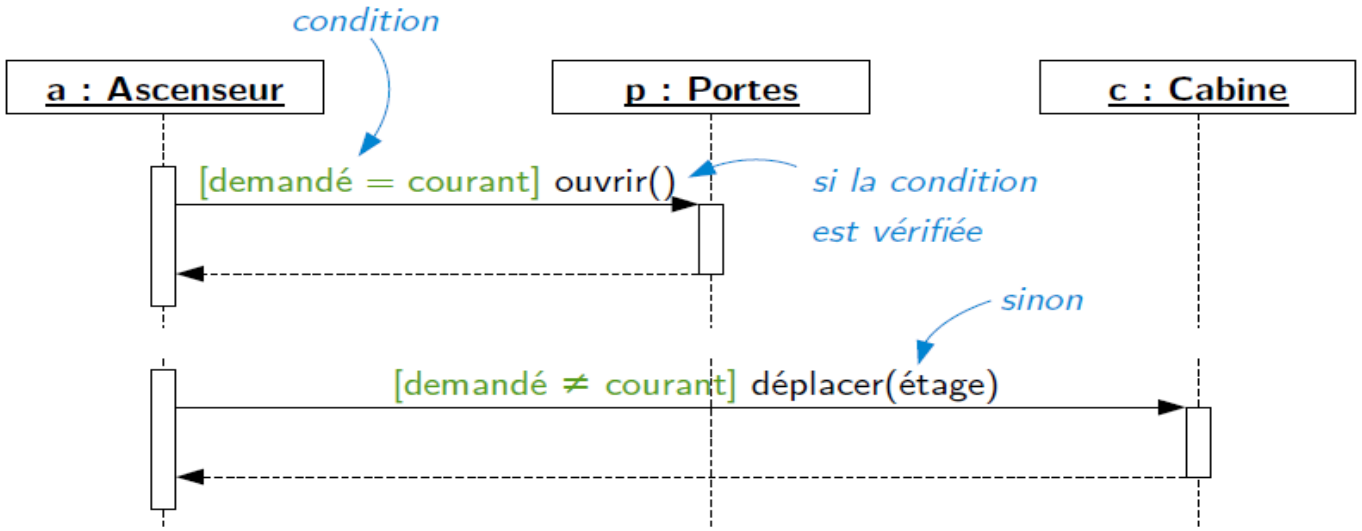


FRAGMENT D'INTERACTION AVEC OPÉRATEUR « ALT »

Principe : Condition à l'envoi d'un message

Notation :

- Deux diagrammes

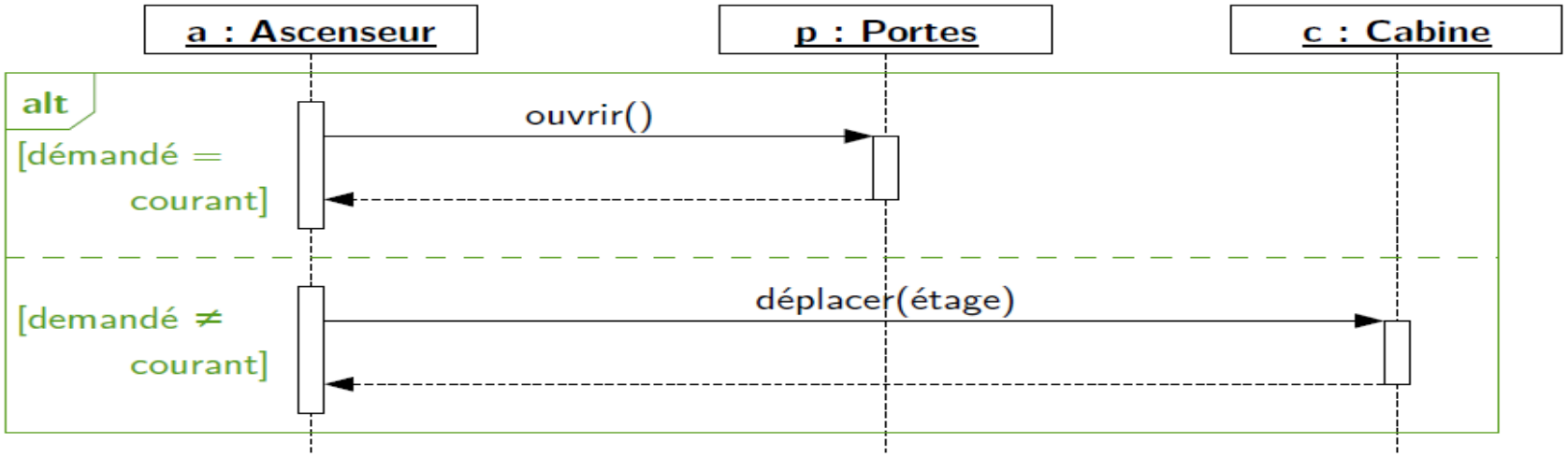


FRAGMENT D'INTERACTION AVEC OPÉRATEUR « ALT »

Principe : Condition à l'envoi d'un message

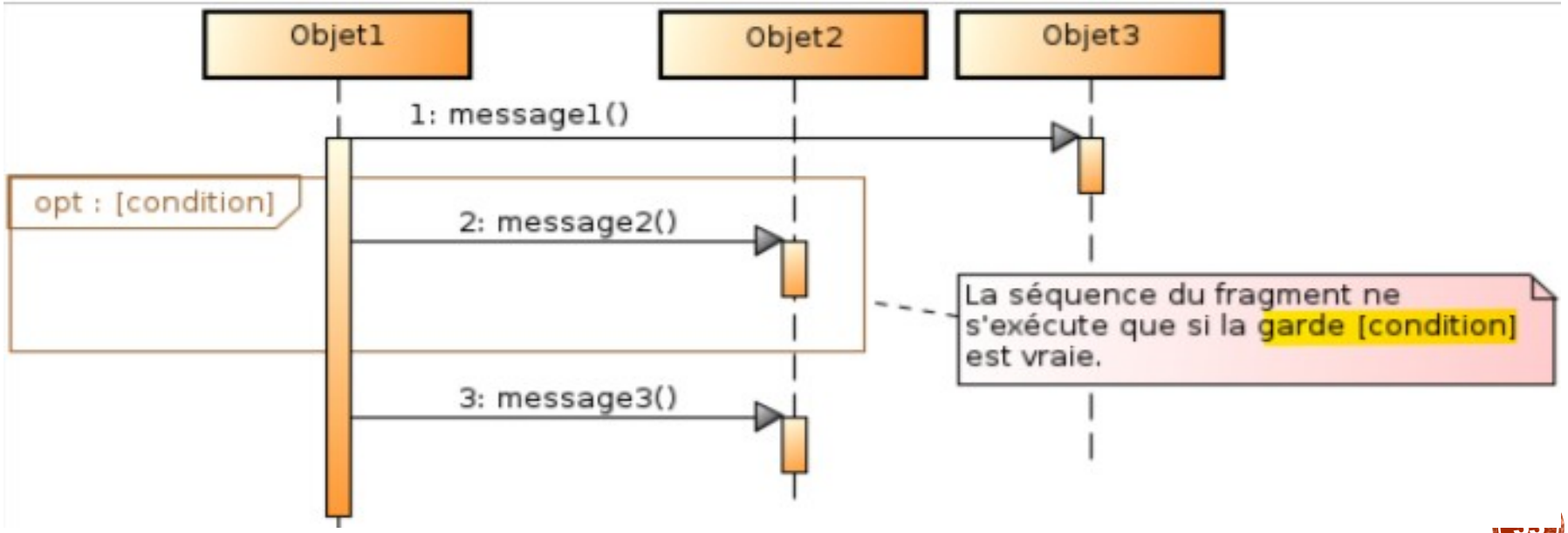
Notation :

- Deux diagrammes
- Bloc d'alternative **alt**



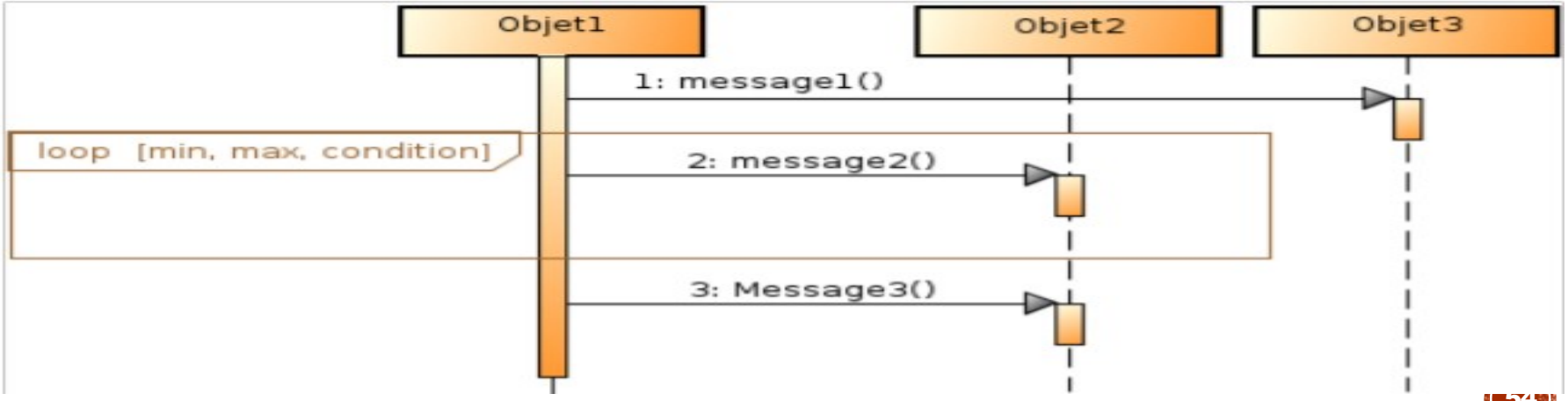
FRAGMENT D'INTERACTION AVEC OPÉRATEUR « OPT »

- Facultatif. Contient une séquence qui peut ou non se produire.
- Un fragment optionnel est équivalent à un fragment "alt" qui ne posséderait pas d'opérande else (qui n'aurait qu'une seule branche).
- Un fragment optionnel est donc une sorte de **SI…ALORS.**
- Le sous-fragment s'exécute si la condition de garde est vraie et ne s'exécute pas dans le cas contraire



FRAGMENT D'INTERACTION AVEC OPÉRATEUR « LOOP »

- utilisé pour décrire un ensemble d'interaction qui s'exécutent **en boucle.**
  - les propriétés **Min et Max** qui indiquent les nombres minimaux et maximaux de fois que le fragment peut être répété
  - ou une condition booléenne **entre crochets** à respecter. L'absence de restriction correspond à la valeur par défaut



FRAGMENT D'INTERACTION AVEC OPÉRATEUR « LOOP »

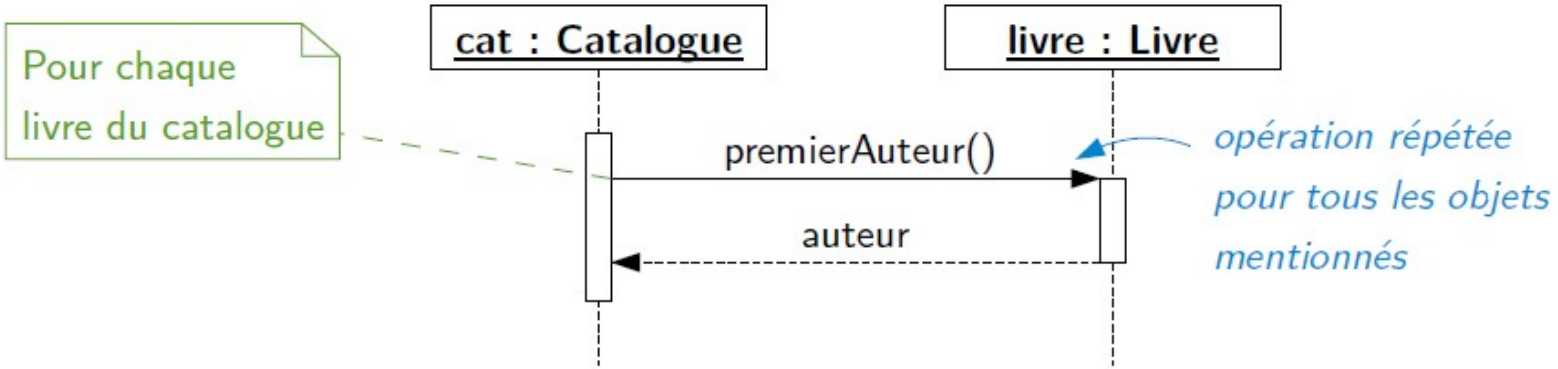
- La garde s'écrit de la façon suivante : **loop [min, max, condition]** Chaque paramètre (min, max et condition) est optionnel.
  - Le contenu du cadre est exécuté min fois, puis continue à s'exécuter tant que la condition et que le nombre d'exécution de la boucle ne dépasse pas max fois.
  - Exemple :
    - loop[3]→La séquence s'exécute 3 fois.
    - Loop[1, 3, code=faux] La séquence s'exécute 1 fois puis un maximum de 3 autres fois si code=faux

EXEMPLE D'ITÉRATION

Principe : Répéter un enchaînement de messages

Notation :

- Note

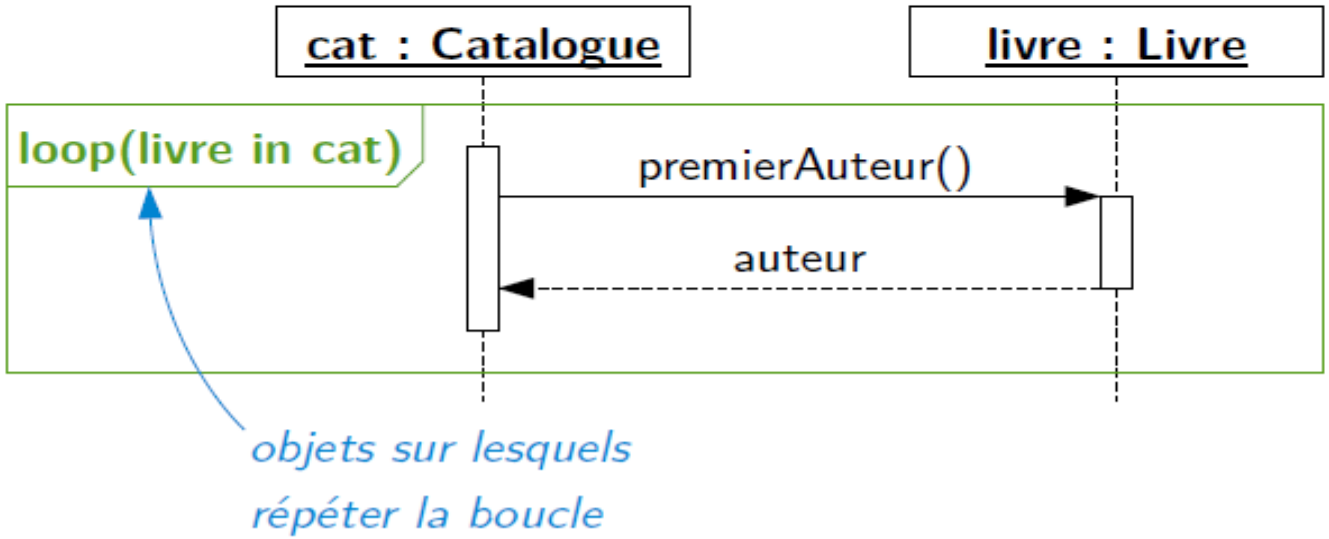


FRAGMENT D'INTERACTION AVEC OPÉRATEUR « LOOP »

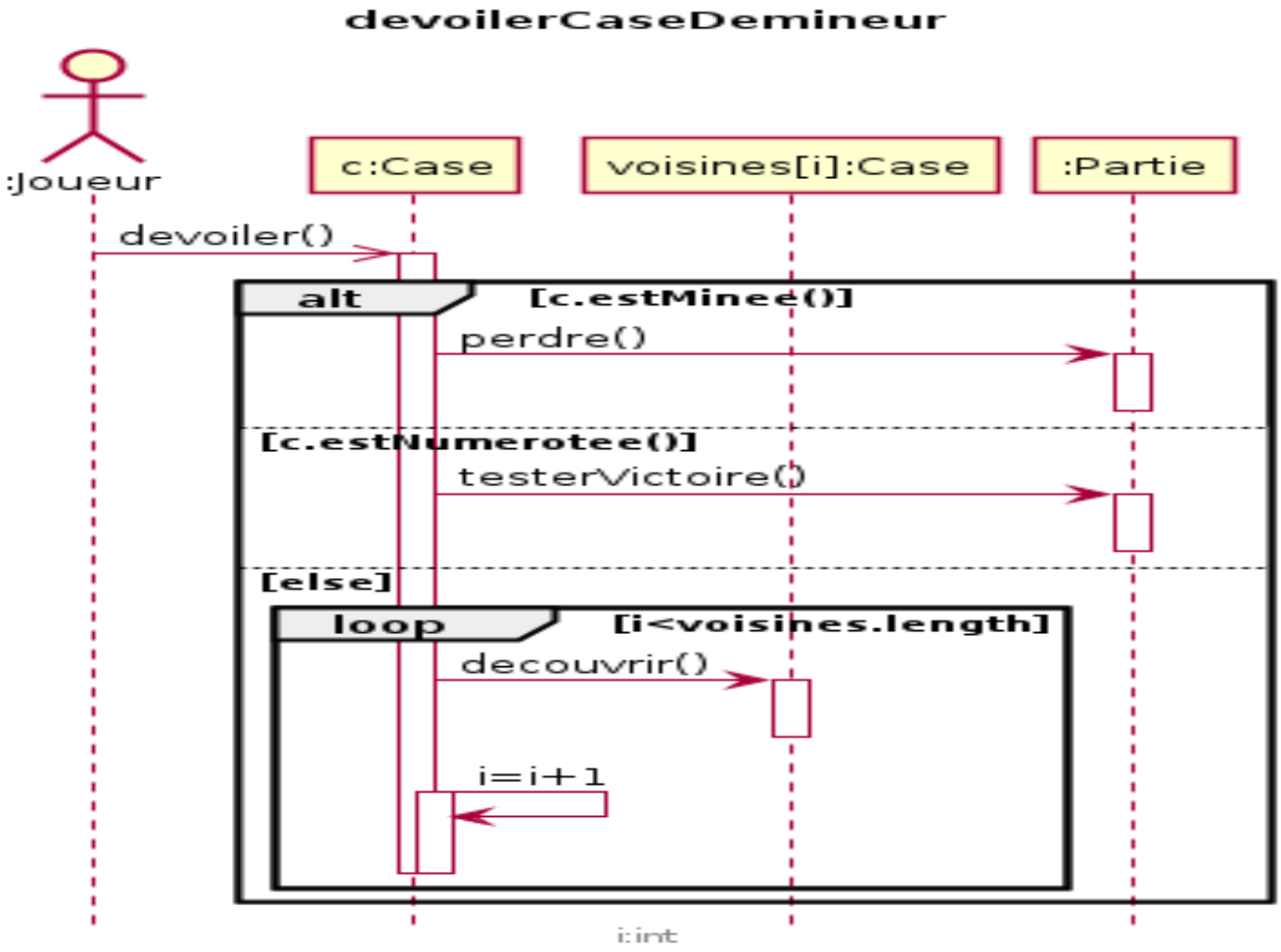
Principe : Répéter un enchaînement de messages

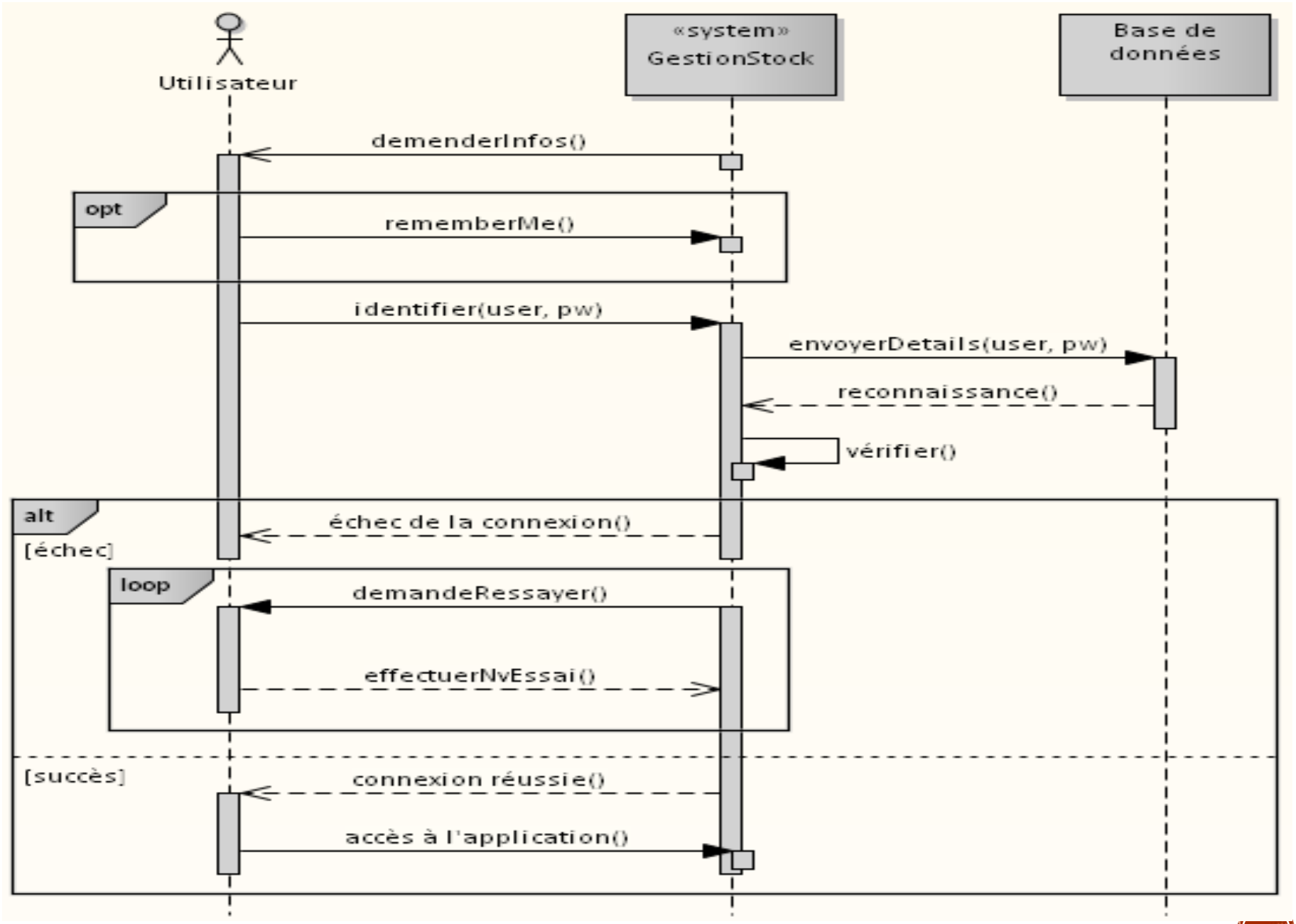
Notation :

- Note
- Bloc de boucle **loop**



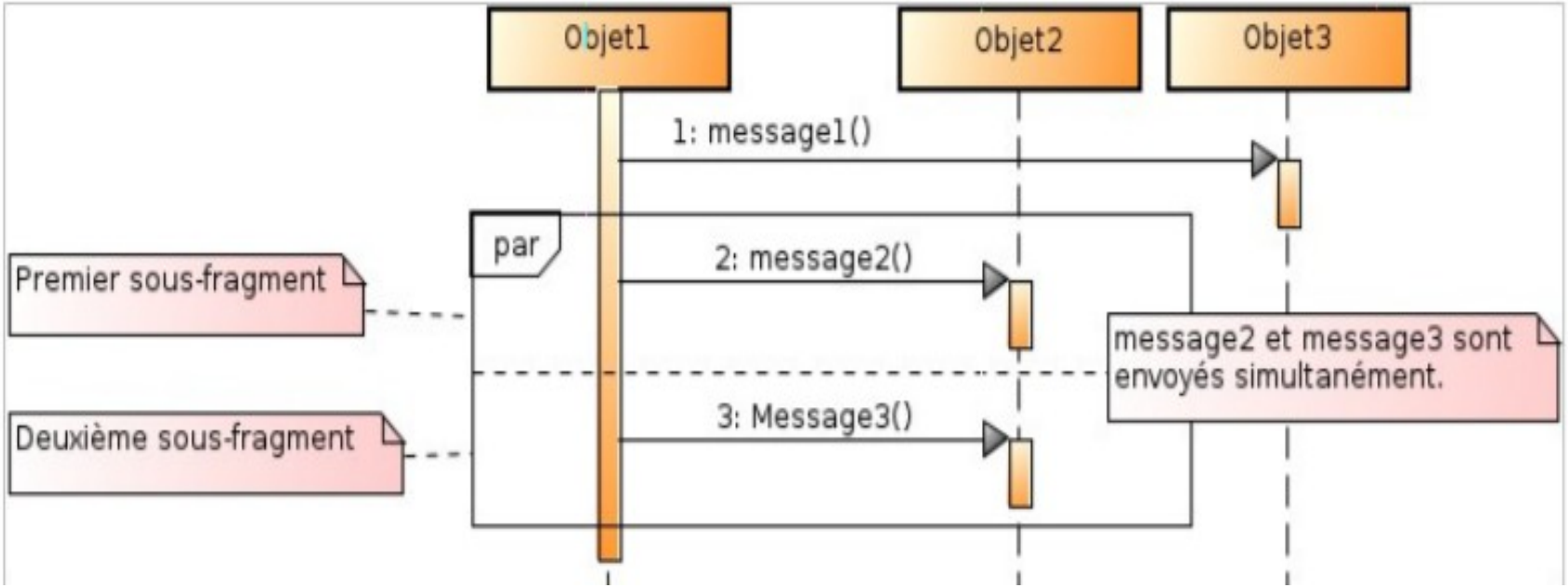
EXEMPLE


devoilerCaseDemineur
:Joueur
c:Case
voisines[i]:Case
:Partie
devoiler()
alt
[c.estMinee()]
perdre()
[c.estNumerotee()]
testerVictoire()
[else]
loop
[i<voisines.length]
decouvrir()
i=i+1
i:int


Utilisateur
«system»
GestionStock
Base de
données
demenderInfos()
opt
rememberMe()
identifier(user, pw)
envoyerDetails(user, pw)
reconnaissance()
vérifier()
alt
échec de la connexion()
[échec]
loop
demandeRessayer()
effectuerNvEssai()
[succès]
connexion réussie()
accès à l'application()

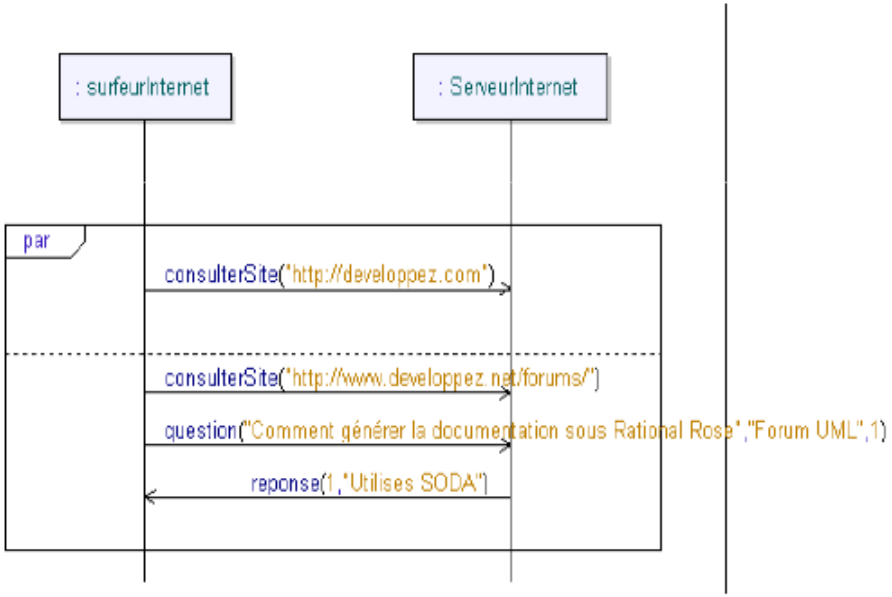
FRAGMENT D'INTERACTION AVEC OPÉRATEUR « PAR »

- 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).

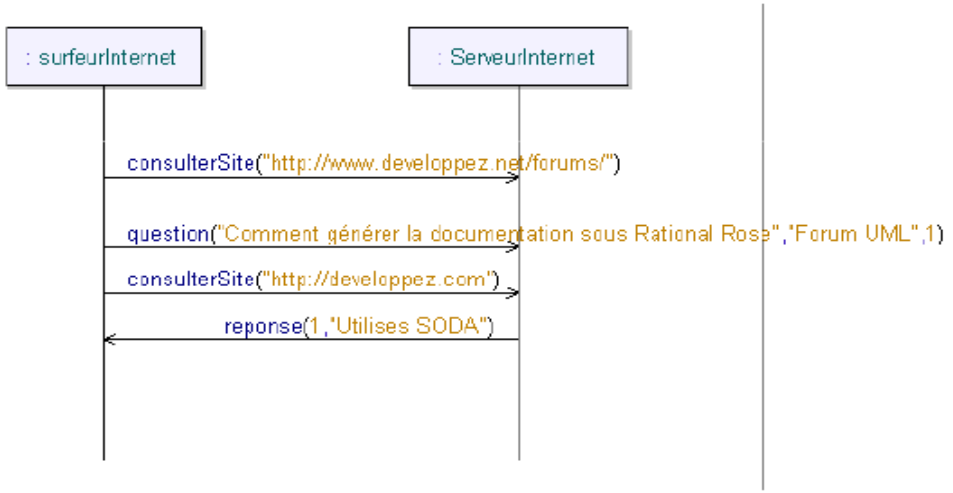


OPÉRATEUR « PAR »

- Représente des interactions ayant lieu en **parallèle. Les interactions des** différents opérandes (les deux branches de notre opérateur ci-dessous) peuvent donc se mélanger, s'intercaler, dans la mesure où l'ordre imposé dans chaque opérande est respecté.



Un développeur ou surfeur internet peut consulter en parallèle, soit le site http://www.developpez.com soit le site http://www.developpez.net/forums/ sans préférence d'ordre (il peut commencer par consulter les forums puis les cours, soit l'inverse).



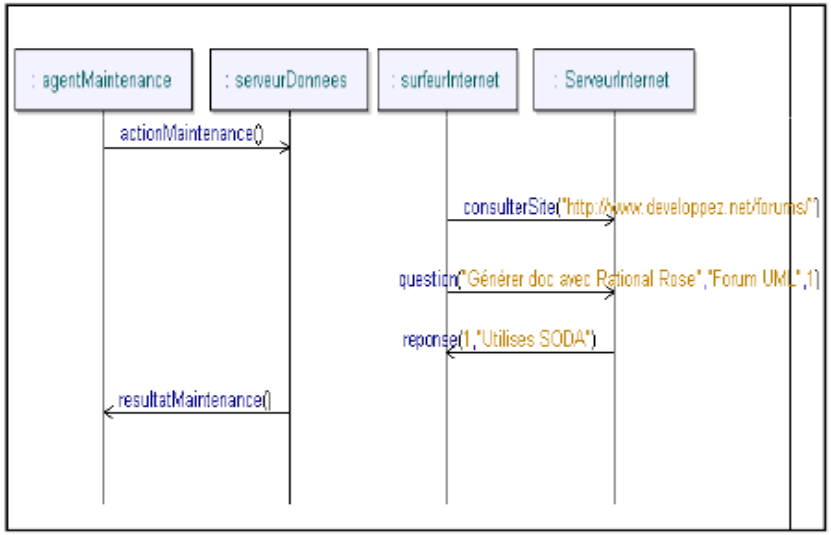
C'est une illustration du diagramme de séquence: l'ordre de l'envoi des messages est respecté: Consulter site forum avant question avant réponse.

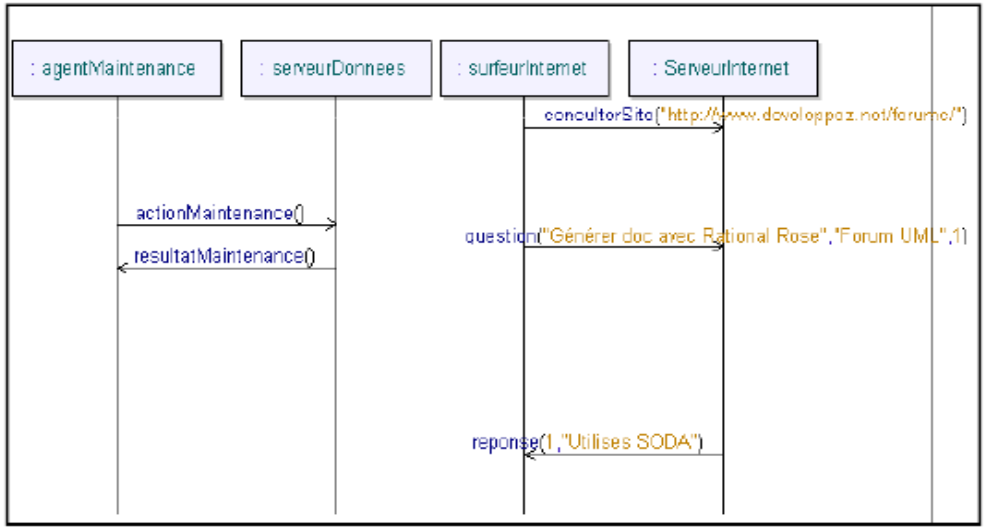
OPÉRATEURS DE FLUX DE CONTRÔLE

- **seq** : indique que le fragment est composé de plusieurs sous fragments qui peuvent s'exécuter dans n'importe quel ordre (mais pas en même temps).
- **critical** : pour les fragments qui doivent se dérouler sans être interrompus. Utilisé dans un fragment par ou seq. Indique que les messages de fragment ne doivent pas être entrelacés avec d'autres messages.
- **strict** : Il existe au moins deux fragments d'opérande. Les fragments doivent se produire dans l'ordre donné.
- **ref** : permet de faire appel à un autre diagramme de séquence.

L'OPÉRATEUR « SEQ »

- Il existe au moins deux fragments d'opérande. Les messages impliquant les mêmes lignes de vie doivent se produire dans l'ordre des fragments. Lorsqu'ils n'impliquent pas les mêmes lignes de vie, les messages des différents fragments peuvent être entrelacés en parallèle.

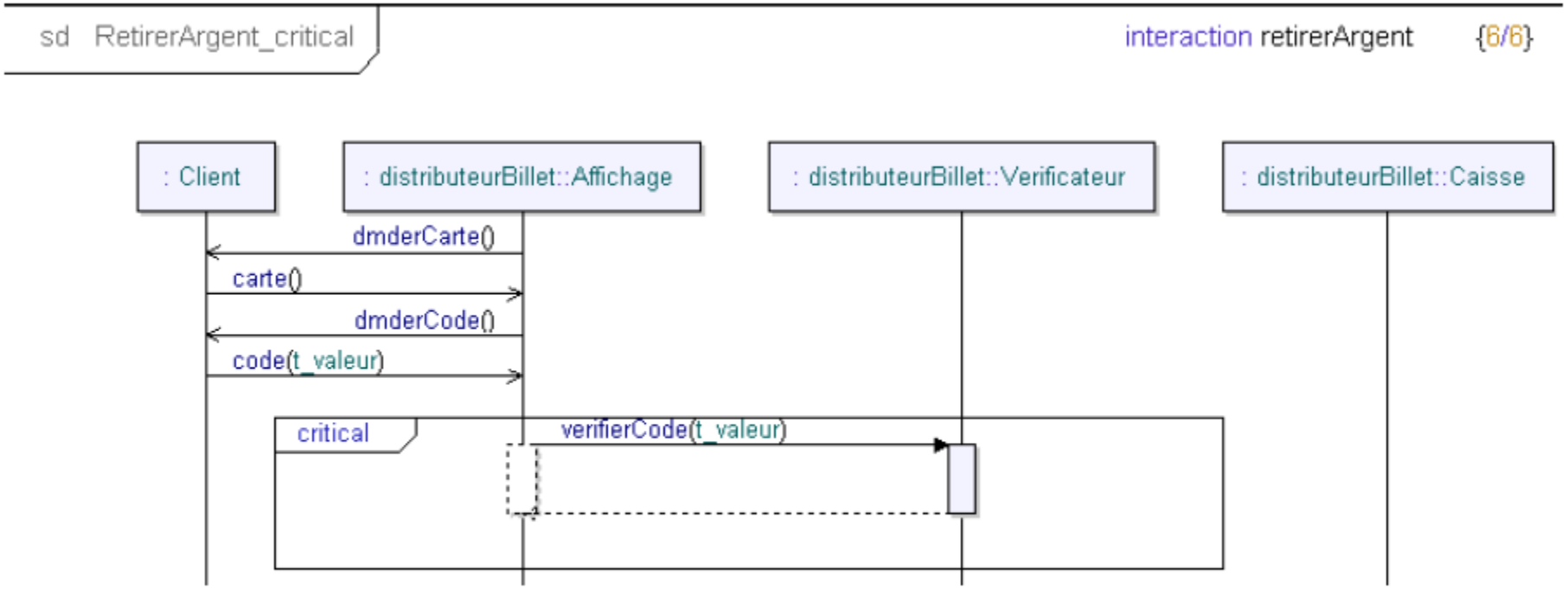




Le « sequencing" est le "mode" de description par défaut des diagrammes de séquence. Ici, l'interprétation de ce diagramme est : - resultatMaintenance arrive après actionMaintenance, - question arrive après consulterSite, - reponse arrive après question. Les deux diagrammes ci-dessus sont donc équivalents.

L'OPÉRATEUR « CRITICAL »

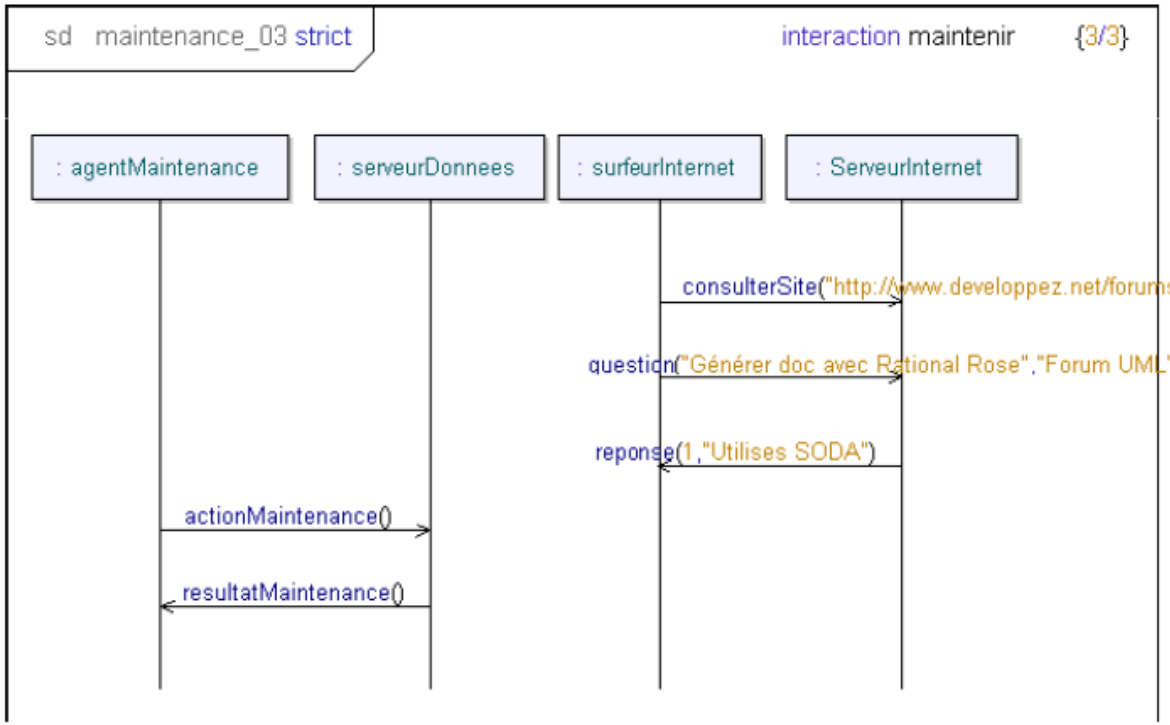
- désigne une **section critique. Une section critique permet** d'indiquer que les interactions décrites dans cet opérateur ne peuvent pas être interrompues par d'autres interactions décrites dans le diagramme. Utilisé dans un fragment Par ou Seq. Indique que les messages de fragment ne doivent pas



On ne souhaite pas que le client puisse obtenir des billets avec un code erroné

L'OPÉRATEUR « STRICT »

- Il existe au moins deux fragments d'opérande. Les fragments doivent se produire dans l'ordre donné. Cet opérateur est à opposer à l'opérateur **" Seq" qui** notifiera que les interactions qui s'opèrent entre des entités indépendantes n'ont pas d'ordre particulier.

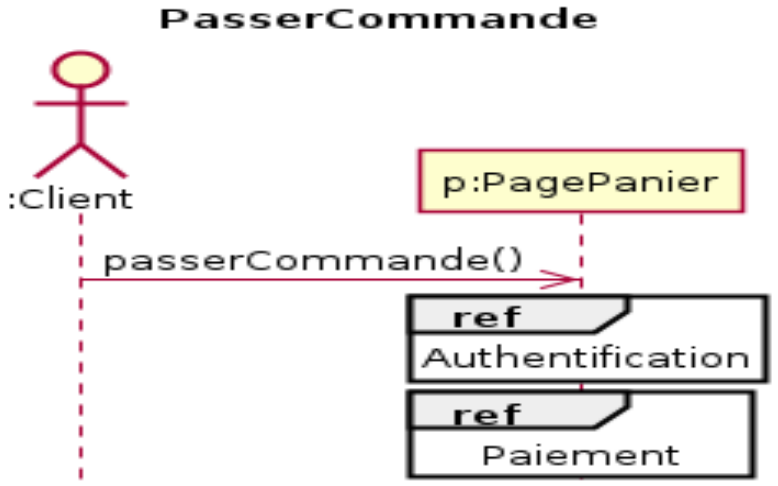


L'interprétation de ce diagramme est :
-resultatMaintenance arrive après actionMaintenance,
-question arrive après consulterSite,
-reponse arrive après question.
-- resultatMaintenance arrive après reponse.

Il y a donc un ordre imposé dans cette séquence entre les deux groupes d'intéractions.

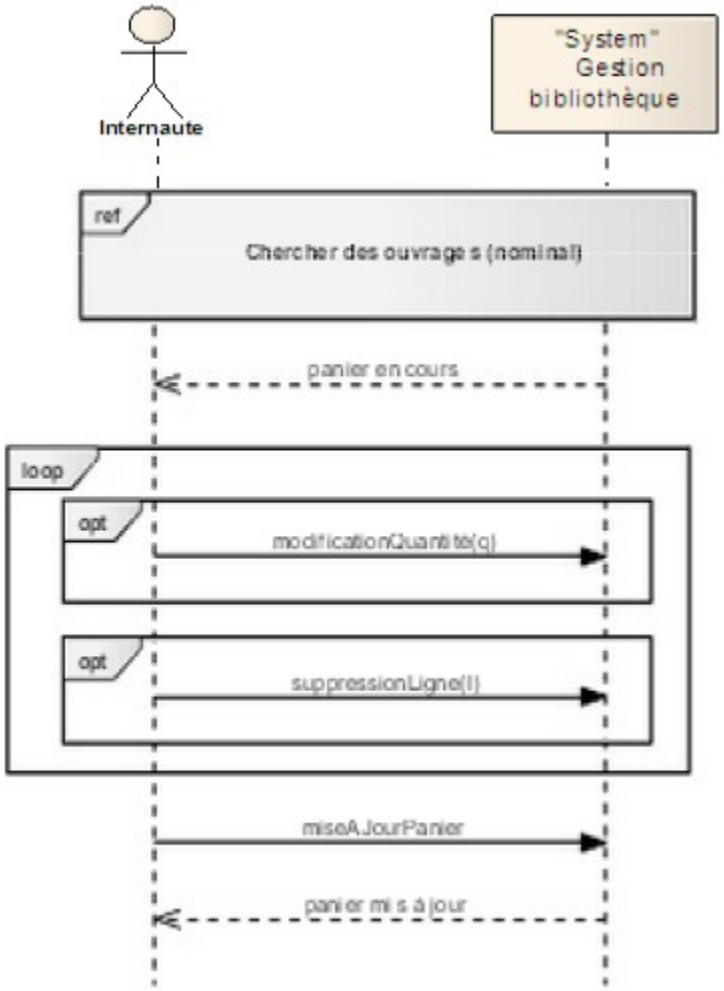
RÉUTILISATION DE SÉQUENCES « REF »

- Un fragment ref permet d'indiquer la réutilisation d'un diagramme de séquences défini par ailleurs. Ce sont des pointeurs ou raccourcis vers d'autres diagrammes de séquence existants. Cela équivaut à copier le contenu du diagramme de séquence pointé en lieu et place de la référence. Cela permet de factoriser des parties de comportement utilisées dans plusieurs scénario.
- En supposant qu'il existe un diagramme intitulé Authentification et un autre Paiement, on peut établir le diagramme suivant :



L'OPÉRATEUR « REF »

- **Diagramme de séquence système de « Gérer son panier »**
- Le renvoi au diagramme de séquences « Chercher des ouvrages » en début de diagramme, montre un exemple de remplissage du panier suite à une recherche réussie.
- Ce diagramme illustre les actions répétées de modification de quantité et de suppression de lignes.



OPÉRATEURS D'INTERPRÉTATION DE LA SÉQUENCE

- **ignore et considere** : pour les fragments facultatifs ou obligatoires.
  - **consider** : Spécifie une liste des messages que ce fragment décrit. D'autres messages peuvent se produire dans le système en cours d'exécution, mais ils ne sont pas significatifs quant aux objectifs de cette description.
  - **ignore** : Liste des messages que ce fragment ne décrit pas. Ils peuvent se produire dans le système en cours d'exécution, mais ils ne sont pas significatifs quant aux objectifs de cette description.
- **assert** : Pour les fragments dont on connaît à l'avance les paramètres du message (exemple : après la saisie des 4 chiffres d'un code, la saisie suivante sera obligatoirement la touche « Entrée »). Le fragment d'opérande spécifie les seules séquences valides. Généralement utilisé dans un fragment Consider ou Ignore.
- **neg** : pour indiquer que la séquence à l'intérieur du fragment n'est pas valide et ne doit pas se produire. Généralement utilisé dans un fragment Consider ou Ignore qui désigne un ensemble d'interactions invalides.