# UML : Etude de la dynamique

# La dynamique

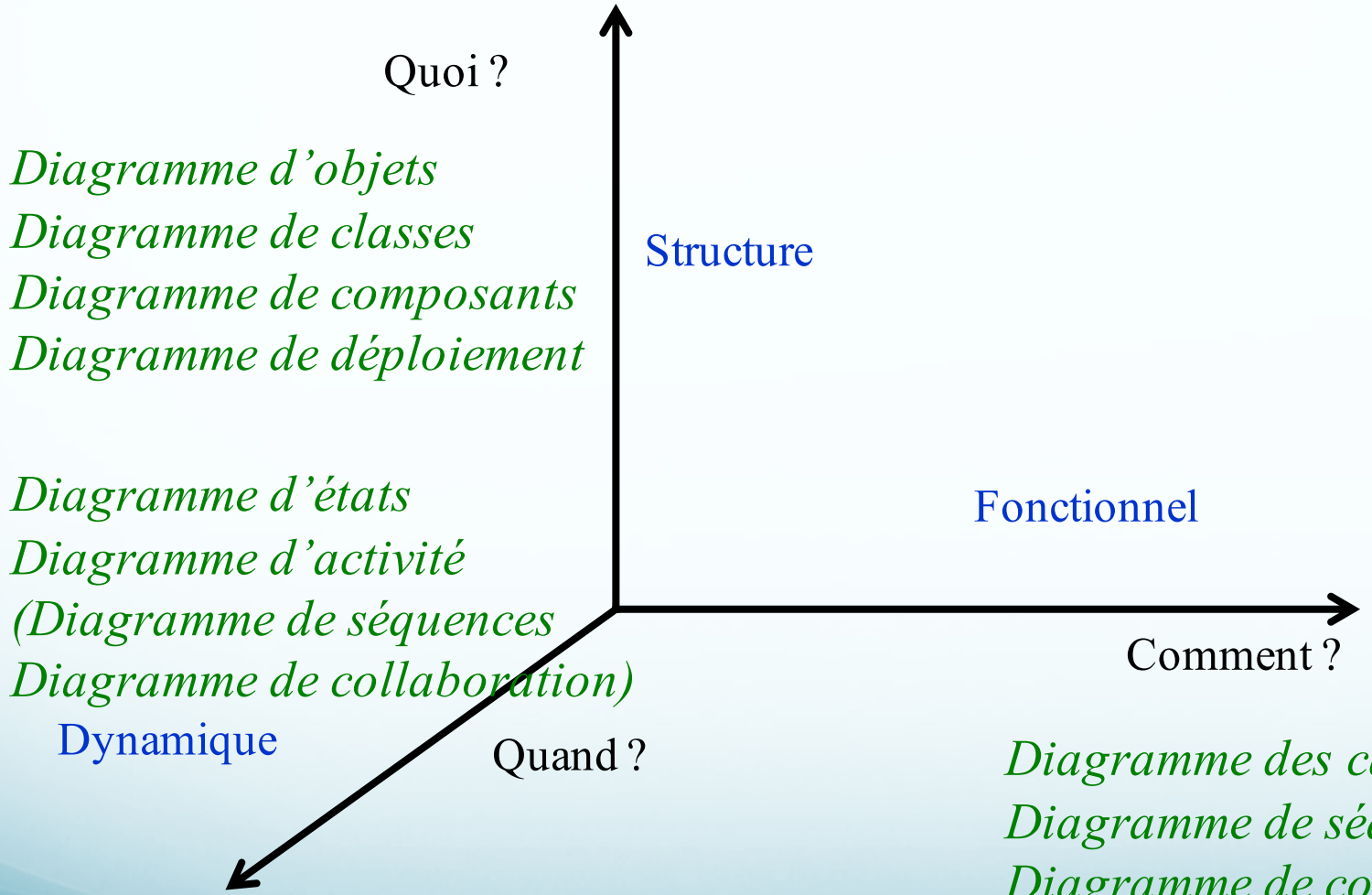



*Diagramme d'objets*
*Diagramme de classes*
*Diagramme de composants*
*Diagramme de déploiement*

Structure

*Diagramme d'états*
*Diagramme d'activité*
*(Diagramme de séquences*
*Diagramme de collaboration)*

Dynamique

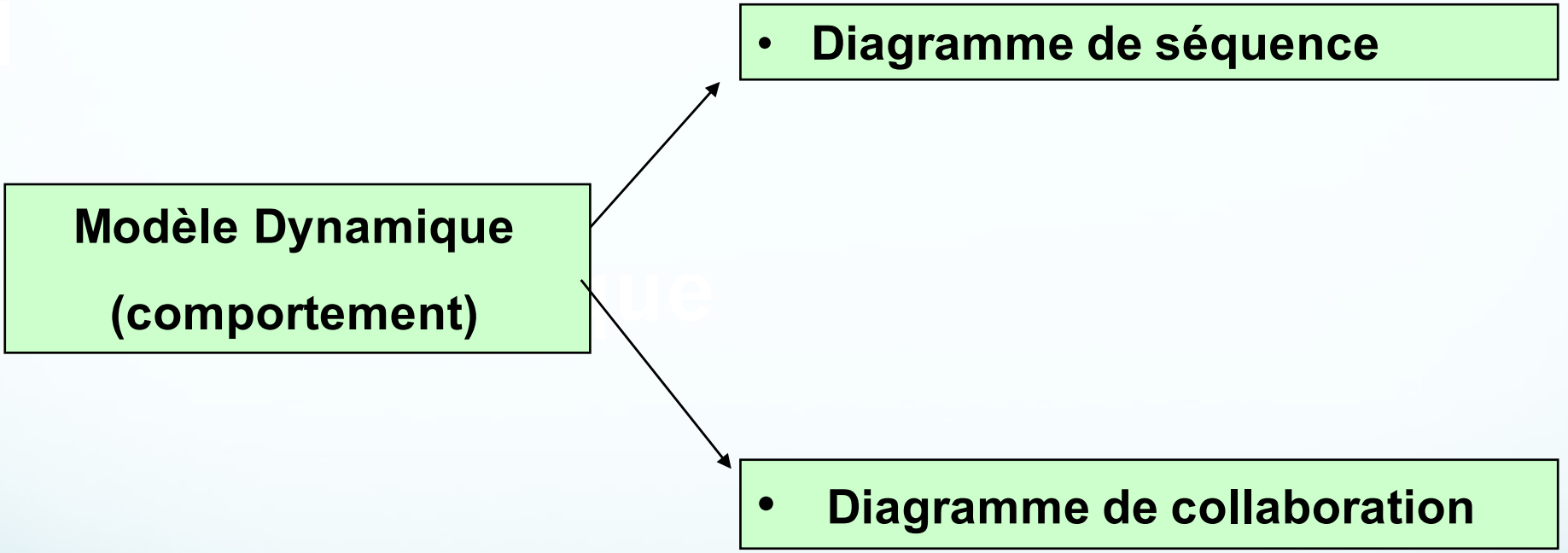
Fonctionnel

*Diagramme des cas*
*Diagramme de séquences*
*Diagramme de collaboration*

# La dynamique

- Décrire le quand ?
- Plusieurs outils UML
  - Les diagrammes de collaboration/séquence (pas de notion d'événement)
  - Les diagrammes d 'état : les différents états d'un objet, d 'un système (dynamique d'un objet)
  - Les diagrammes d 'activité : une activité du système ou d'un objet, un processus (dynamique d'un traitement : diagramme de traitement)

# Modèle dynamique



# Introduction

- C'est un diagramme qui donne une représentation graphique des interactions entre objets.

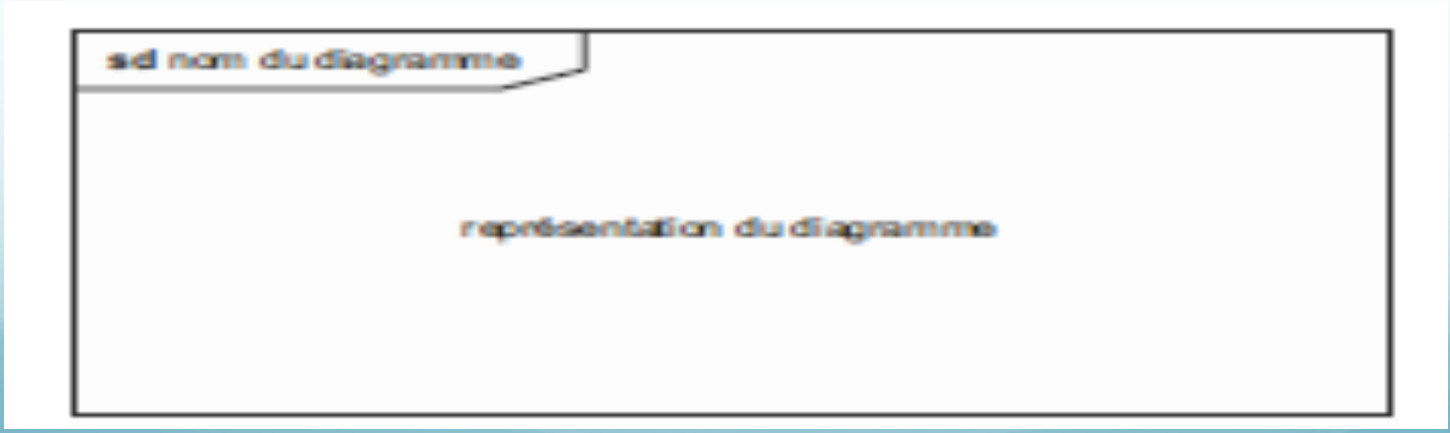
Il existe deux types de diagrammes d'interaction :

- Le diagramme de séquence (fonction du temps),
- Le diagramme de collaboration( représente les flots de données).

- Chacun de ces deux diagrammes représente la même chose vue différemment.

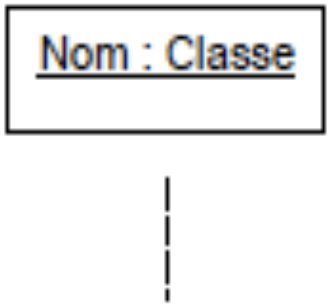
# Diagramme de séquence

- L'objectif du **diagramme de séquence est de représenter les interactions entre objets** en indiquant la chronologie des échanges.
- Cette représentation peut se réaliser par cas d'utilisation en considérant les différents scénarios associés.
- Un diagramme de séquence se représente globalement dans un grand rectangle avec indication du nom du diagramme en haut à gauche comme indiqué



# Représentation des interactions

- Un objet est matérialisé par un rectangle et une barre verticale appelée ligne de vie des objets

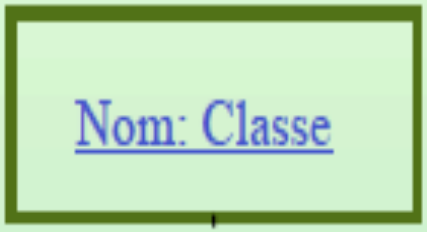



*Représentation graphique d'un objet.*

- Les objets communiquent en échangeant des messages représentés au moyen de flèches horizontales, orientées de l'émetteur du message vers le destinataire.
- L'ordre d'envoi des messages est donné par la position sur l'axe vertical

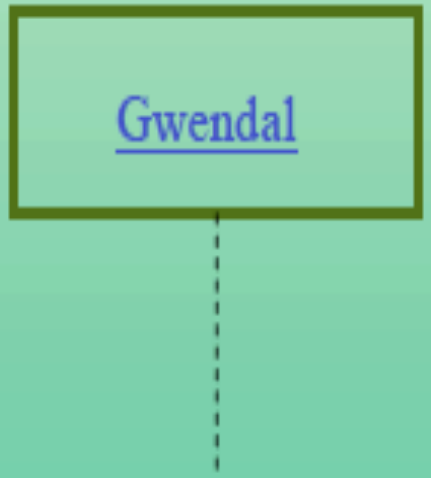
# Nommer un objet dans un diagramme de séquence :


L'objet, l'individu est connu



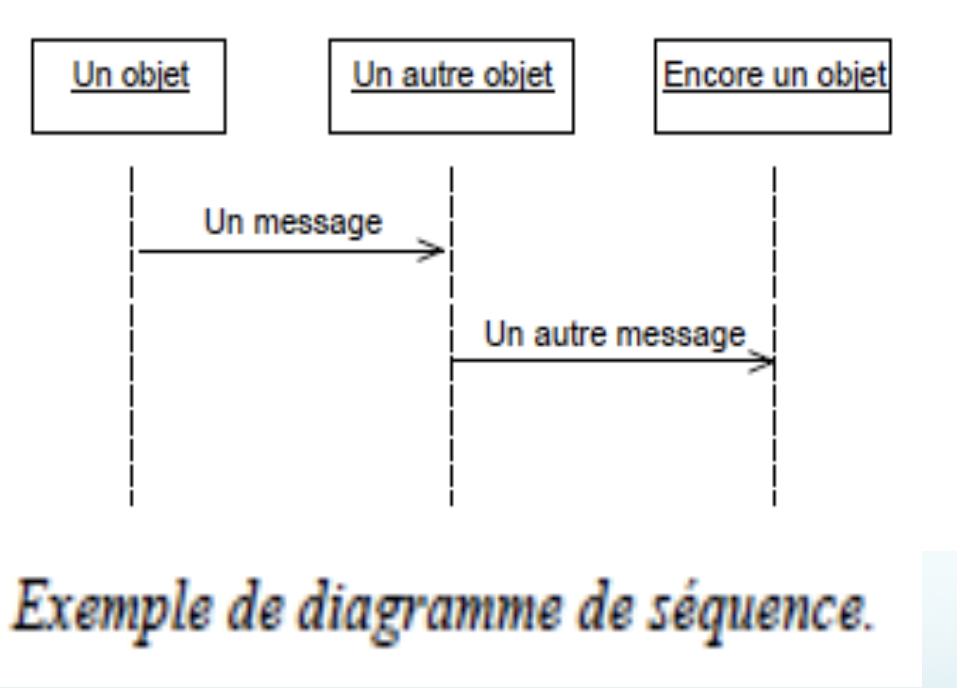


un objet non précisé de la classe indiquée




# Suite 1

- L'axe vertical peut être gradué afin d'exprimer précisément les contraintes temporelles, dans le cas par exemple de la modélisation d'un système temps réel.




*Exemple de diagramme de séquence.*

# Suite 2

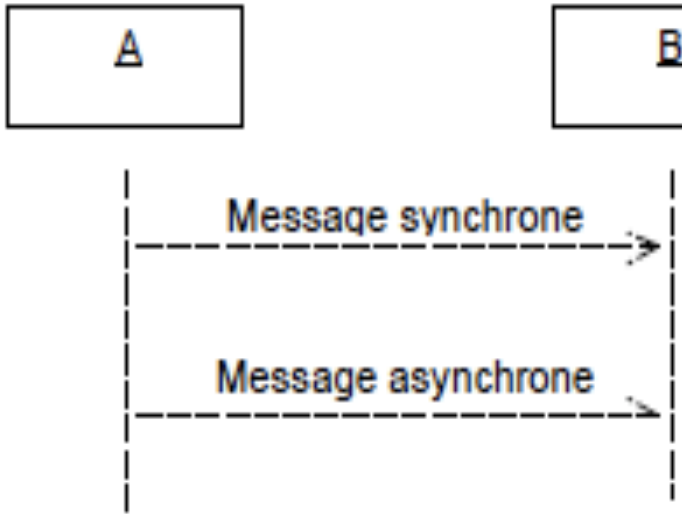
- Modélisation des **interactions entre objets** suite à un **événement externe**
- **Aspect temporel :**
  - **messages asynchrones**
  - **ou synchrones**
- **2 catégories de messages :**
  - **synchrone :** l'émetteur est bloqué jusqu'au traitement effectif du message
  - **asynchrone :** l'émetteur n'est pas bloqué, il peut poursuivre son exécution

# Suite 3

synchrone : ils sont alors symbolisés par 

asynchrone : ils sont alors symbolisés par 

- Il montre étape par étape ce qui se passe pour accomplir une fonctionnalité fournie par le système

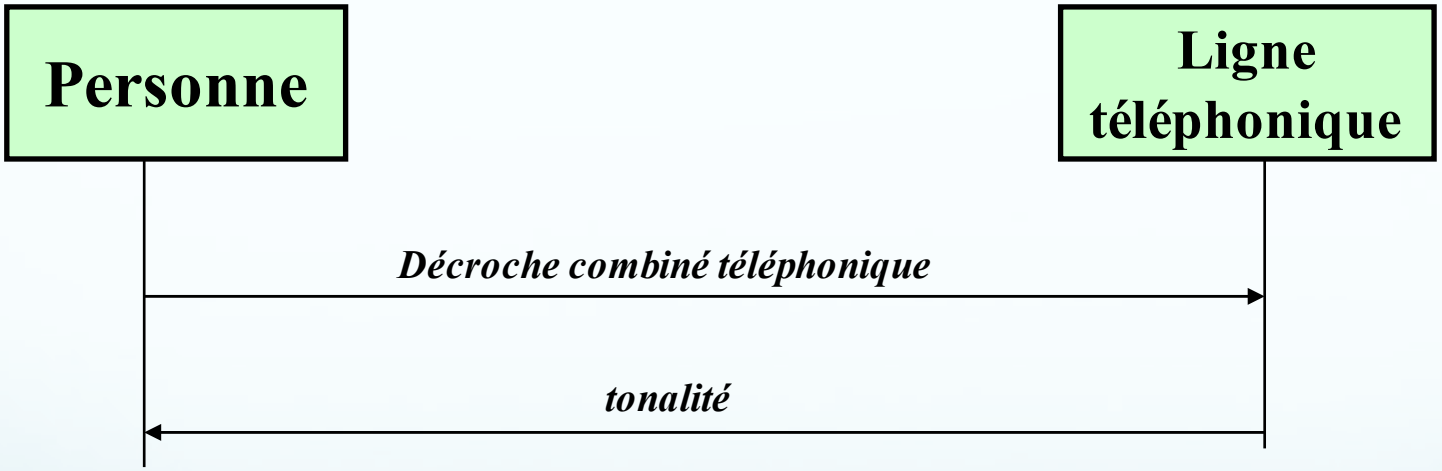



*Représentation graphique des envois de message synchrones et asynchrones.*

# Suite 4

•Un diagramme de séquence montre des interactions entre des objets selon un point de vue temporel.

•**Exemple :** décrocher un récepteur téléphonique.



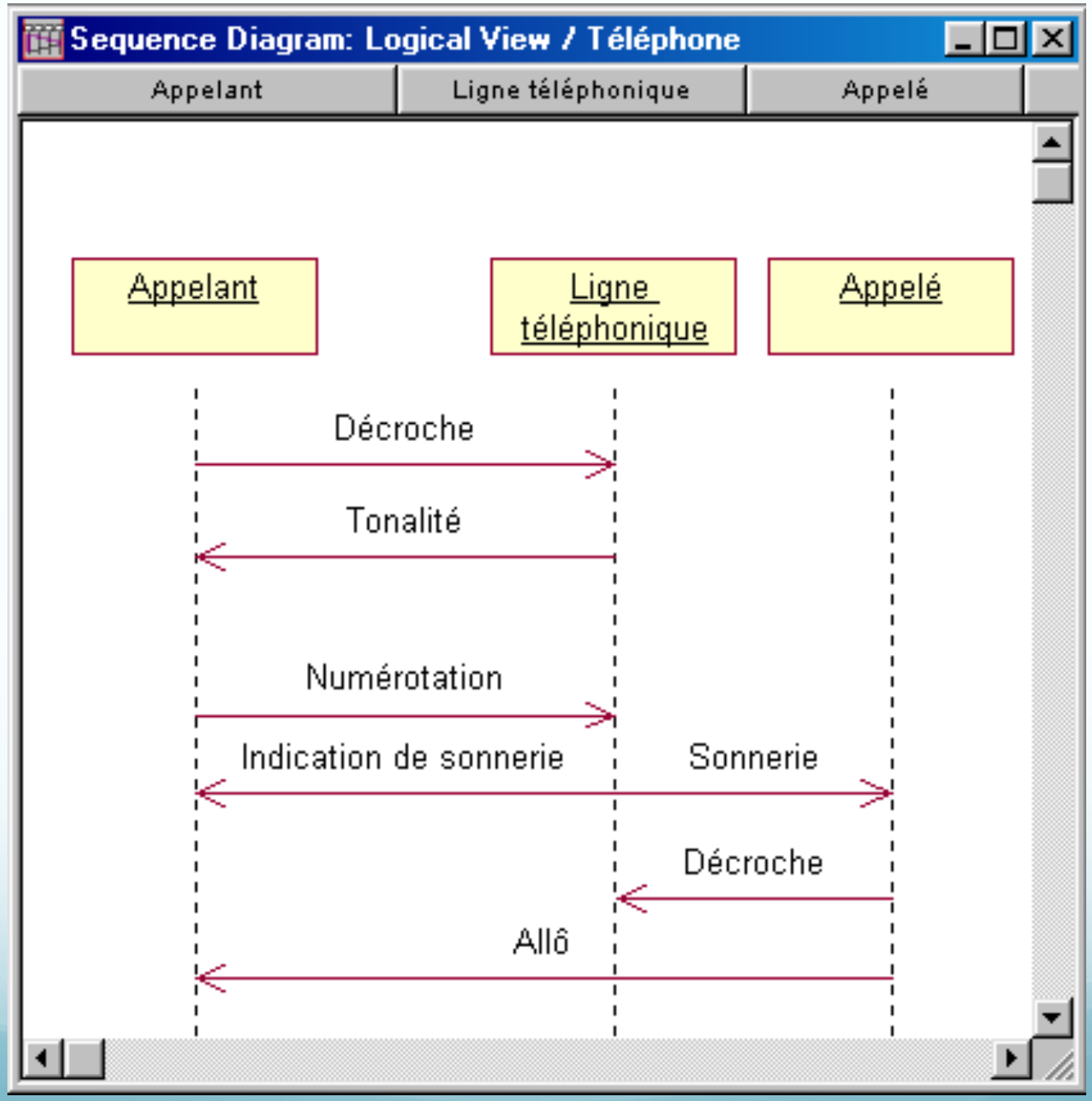

•Une personne décroche le combiné téléphonique qui a pour but d'envoyer un événement à la ligne téléphonique.

•La ligne envoi la tonalité à la personne.

# Exercice

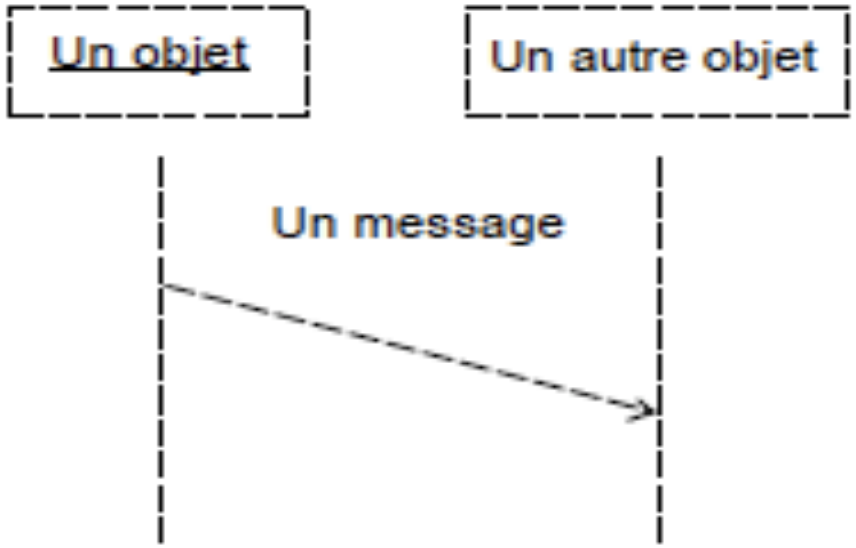
Réalisez un diagramme de séquence pour l'appel d'un utilisateur à un autre par téléphone

# Solution


Sequence Diagram: Logical View / Téléphone
Appelant
Ligne téléphonique
Appelé
Appelant
Ligne téléphonique
Appelé
Décroche
Tonalité
Numérotation
Indication de sonnerie
Sonnerie
Décroche
Allô

# Délai de propagation

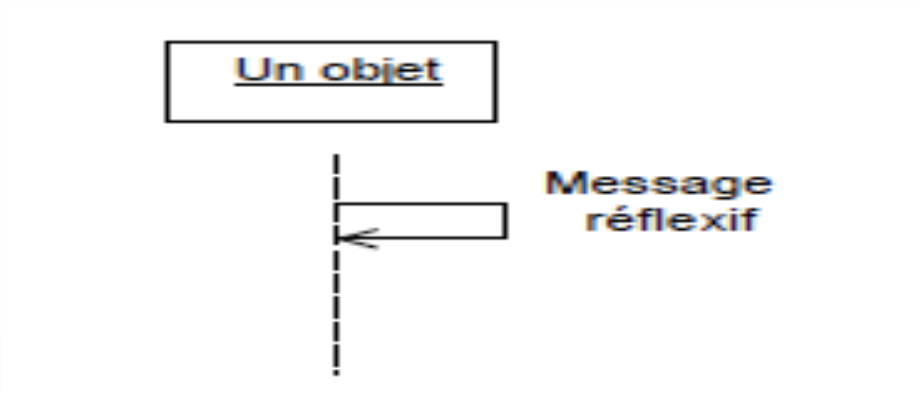
La flèche qui symbolise un message peut être représentée en oblique pour matérialiser les délais de transmission non négligeables par rapport à la dynamique générale de l'application




*Représentation d'un délai de propagation.*

# Message réflexif

Un objet peut également s'envoyer un message. Cette situation se représente par une flèche qui boucle sur la ligne de vie de l'objet.

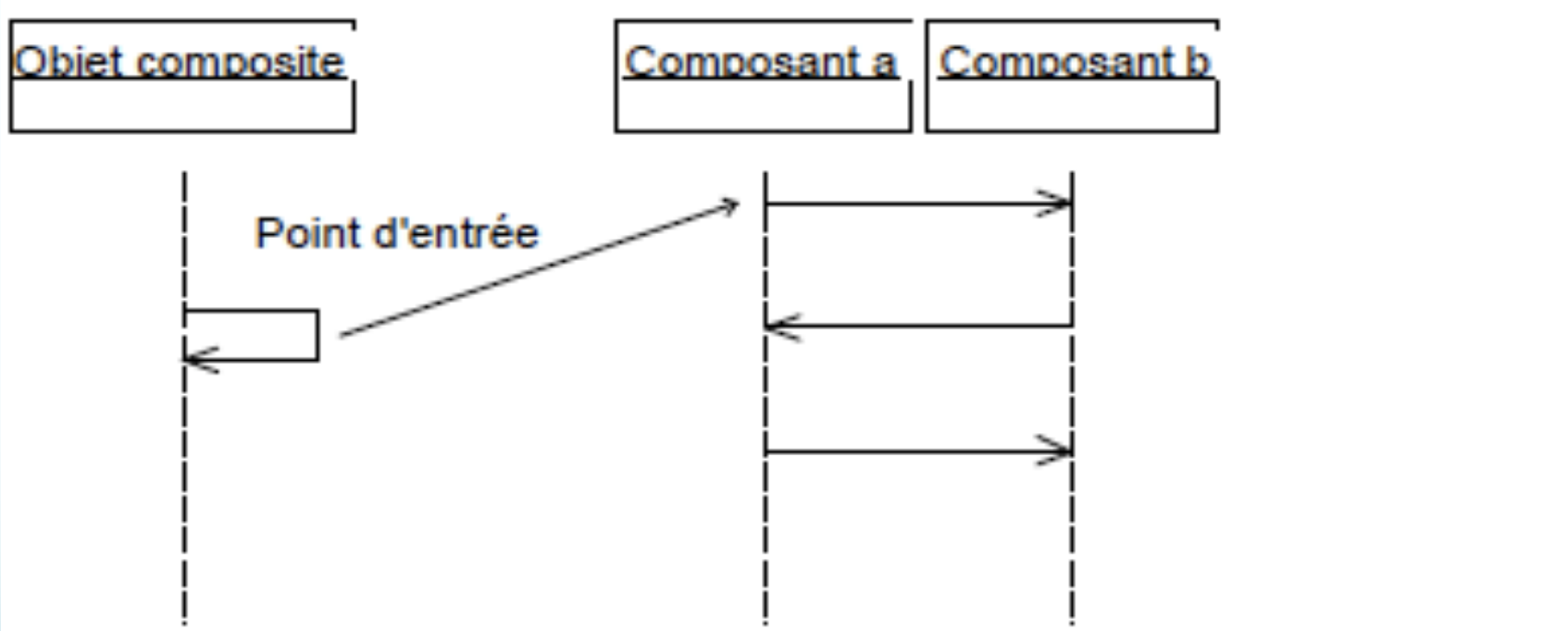



*Exemple d'envoi de message réflexif.*

Cette construction ne correspond pas toujours à un vrai message ; elle peut indiquer un point d'entrée dans une activité de plus bas niveau, qui s'exerce au sein de l'objet.

# Message réflexif (Suite)

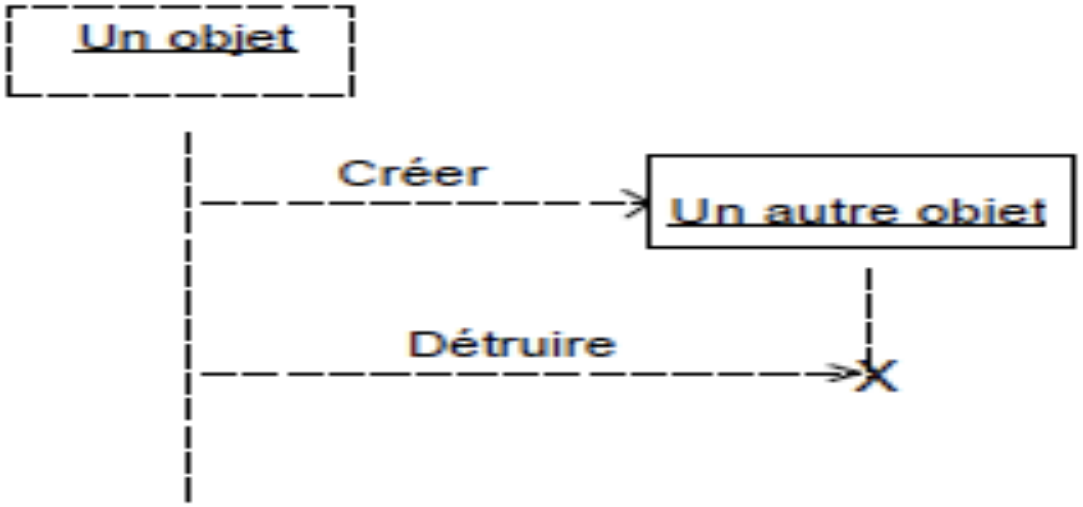
Les interactions internes (entre objets contenus par un objet composite) représentées par un message réflexif peuvent aussi être décrites dans un diagramme de séquence




*: Utilisation d'un message réflexif comme point d'entrée d'une interaction interne.*

# Message (Création/Destruction)

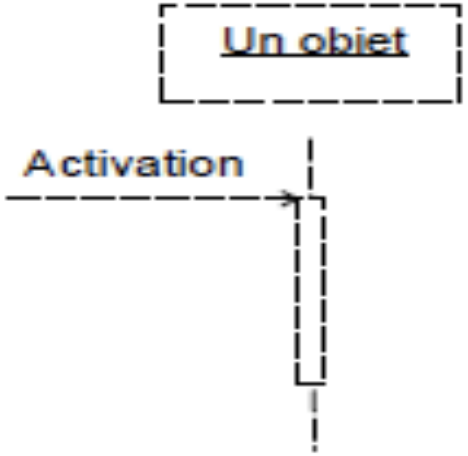
La création des objets se représente en faisant pointer le message de création sur le rectangle qui symbolise l'objet créé. La destruction est indiquée par la fin de la ligne de vie et par une lettre X soit à la hauteur du message qui cause la destruction, soit après le dernier message envoyé par un objet qui se suicide.




*Représentation de la création et de la destruction des objets.*

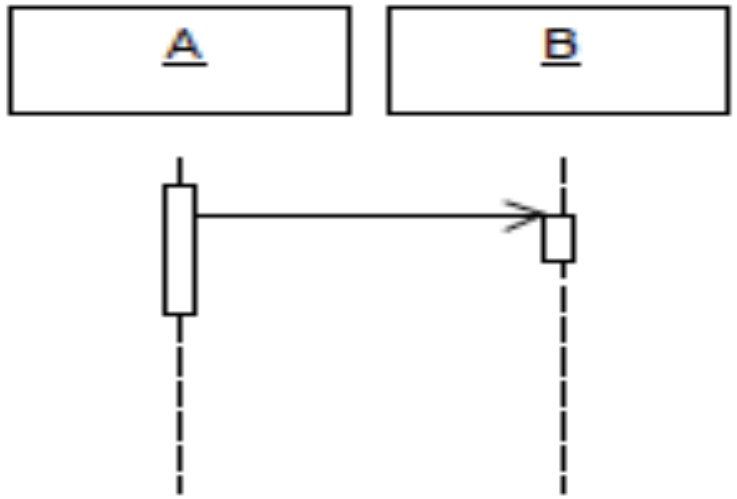
# Période d'activité

Les diagrammes de séquence permettent également de représenter les périodes d'activité des objets. Une période d'activité correspond au temps pendant lequel un objet effectue une action, soit directement, soit par l'intermédiaire d'un autre objet qui lui sert de sous-traitant. Les périodes d'activité se représentent par des bandes rectangulaires placées sur les lignes de vies. Le début et la fin d'une bande correspondent respectivement au début et à la fin d'une période d'activité.



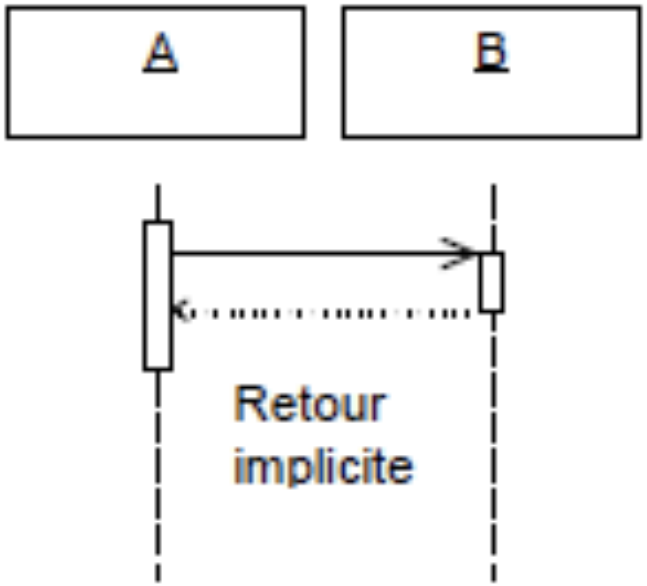

*Représentation de la période d'activité d'un objet au moyen d'une bande rectangulaire superposée à la ligne de vie de l'objet.*

Le diagramme suivant montre le cas d'un objet A qui active un autre objet B. La période d'activité de l'objet A recouvre la période d'activité de l'objet B. Dans le cas d'un appel de procédure, le flot d'exécution est passé par l'objet A à l'objet B. L'objet A est alors bloqué jusqu'à ce que l'objet B lui redonne la main



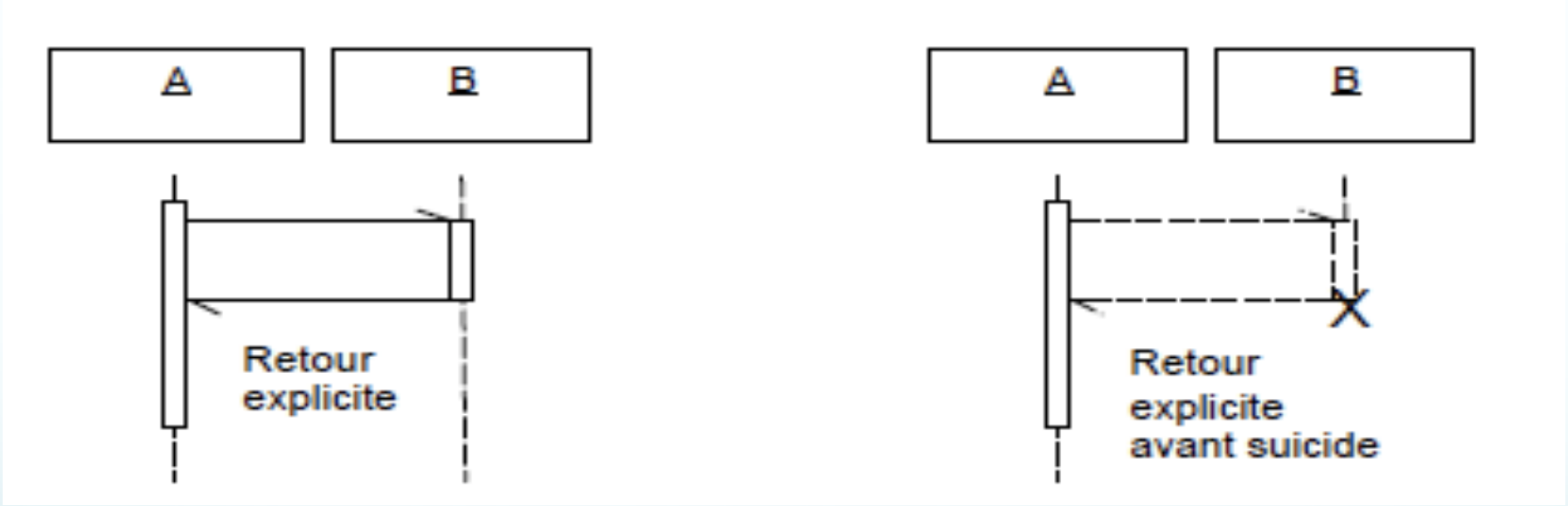

*: Exemple d'objet qui active un autre objet.*

Dans le cas d'un appel de procédure, et plus généralement dans le cas des envois synchrones, le retour en fin d'exécution de l'opération est implicite : il n'est pas nécessaire de le représenter dans les diagrammes. L'objet A reprend son exécution lorsque l'action déclenchée dans l'objet B est terminée.




*Dans le cas des envois synchrones, le retour est implicite en fin d'activité et ne nécessite pas de représentation particulière.*

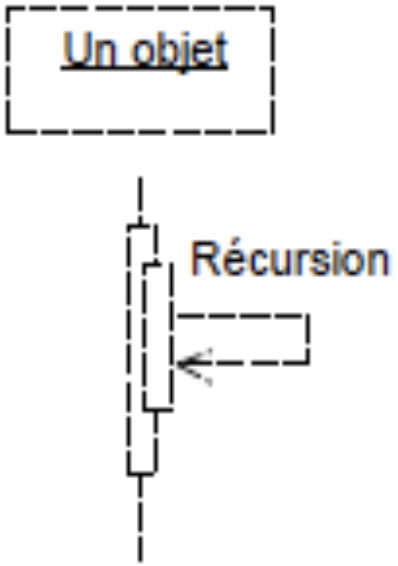
En revanche, dans le cas des envois asynchrones, le retour doit être matérialisé lorsqu'il existe. Le diagramme suivant montre un objet B initialement activé par un Objet A qui retourne un message à l'objet A avant de cesser son exécution. Il faut noter que la fin de l'activation d'un objet ne correspond pas à la fin de sa vie. Un même objet peut être activé de nombreuses fois au cours de son existence.




*Représentation explicite du retour dans le cas des envois asynchrones. La lettre*

*X symbolise une fin d'activité qui correspond également à une fin de vie.*

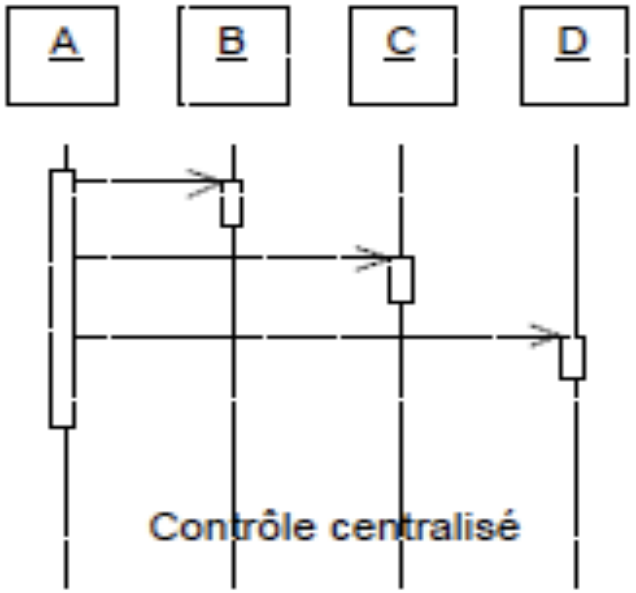
Le cas particulier des envois de messages récursifs se représente par un dédoublement de la bande rectangulaire. L'objet apparaît alors comme s'il était actif plusieurs fois.



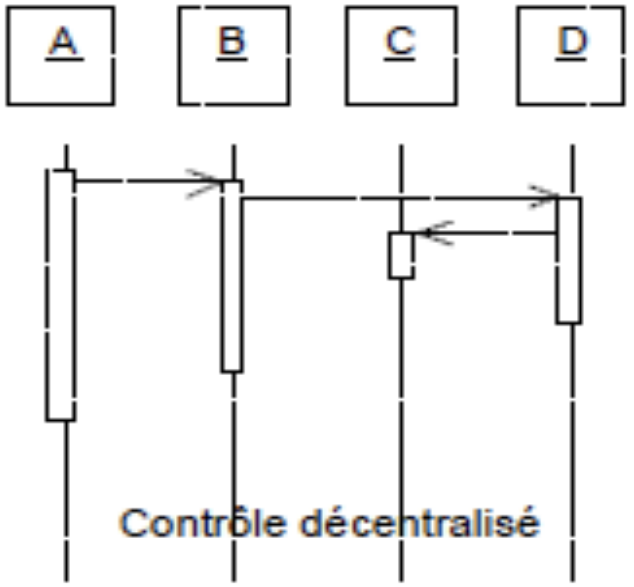

*Représentation de la récursion dans les diagrammes de séquence.*

# Structures de controle

Les formes des diagrammes de séquence reflètent indirectement les choix de structure. Les deux diagrammes suivants présentent respectivement un mode de contrôle centralisé et un mode décentralisé.




Contrôle centralisé

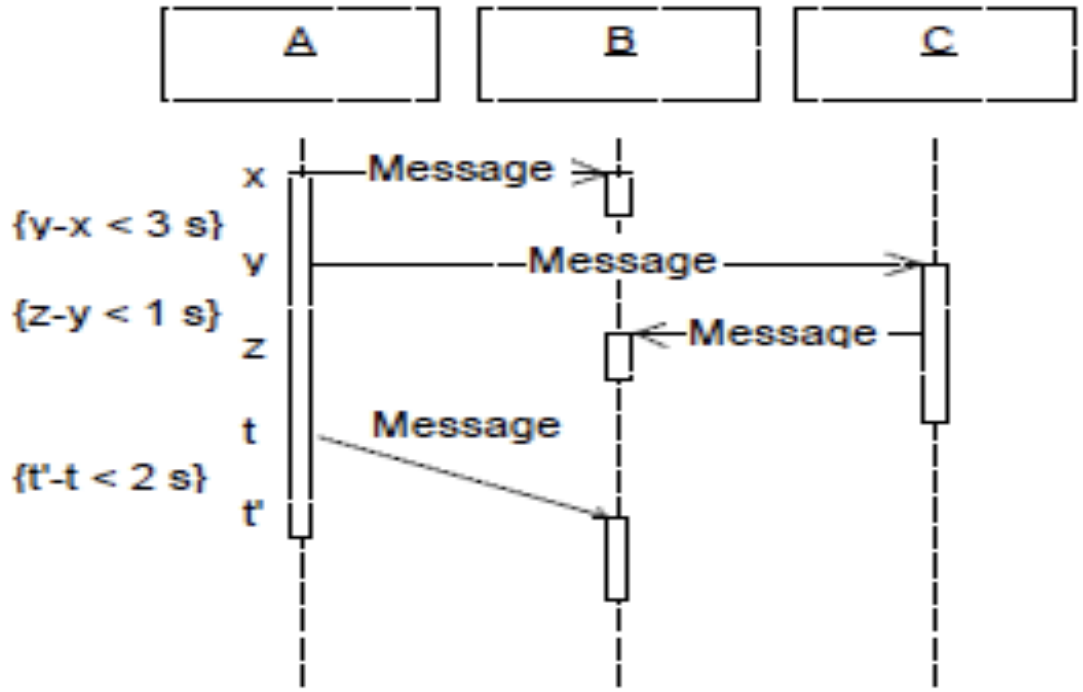



Contrôle décentralisé

*La forme des diagrammes de séquence est le reflet du mode de contrôle de l'interaction.*

# Structures de contrôle (Suite)

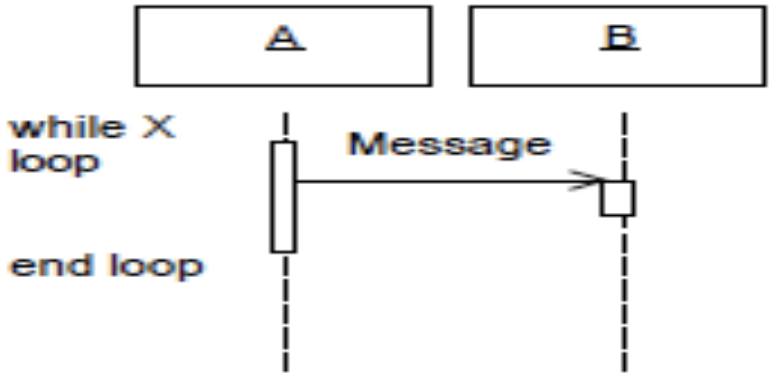
Les diagrammes de séquence peuvent être complétés par des indications textuelles, exprimées sous la forme de texte libre ou de pseudo-code. L'instant d'émission d'un message, appelé transition, peut être nommé dans le diagramme à proximité du point de départ de la flèche qui symbolise le message



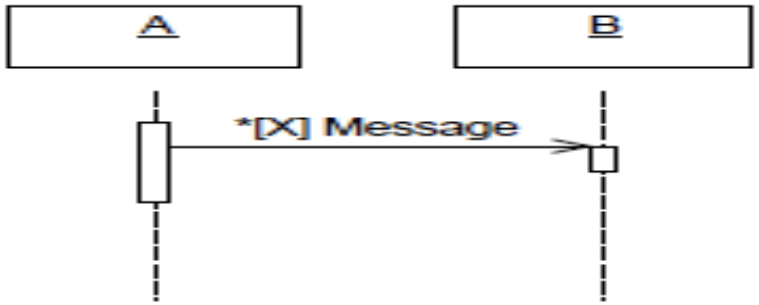

*Exemples de contraintes temporelles construites à partir de noms de transitions.*

# Structures de controle

Le diagramme suivant représente une boucle while. L'objet A envoie sans discontinuer un message à B tant que la condition X est vraie.



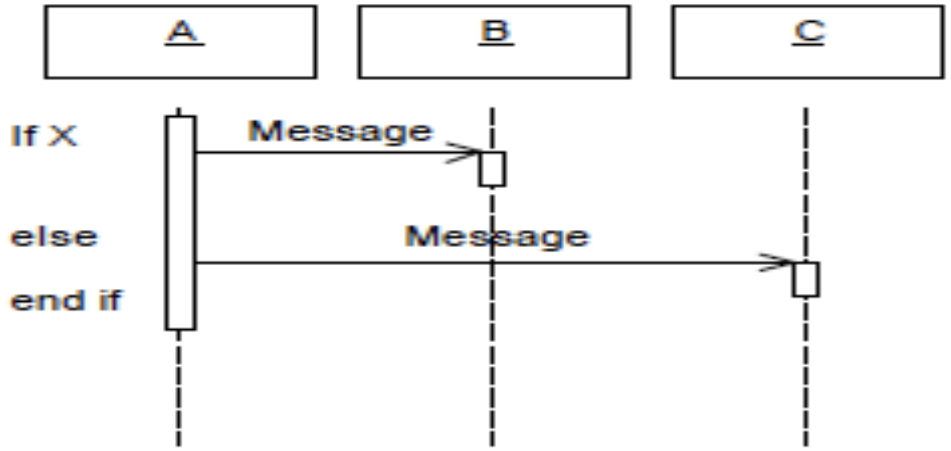

*Exemple de représentation d'une boucle* **While** *au moyen de pseudo-code.*



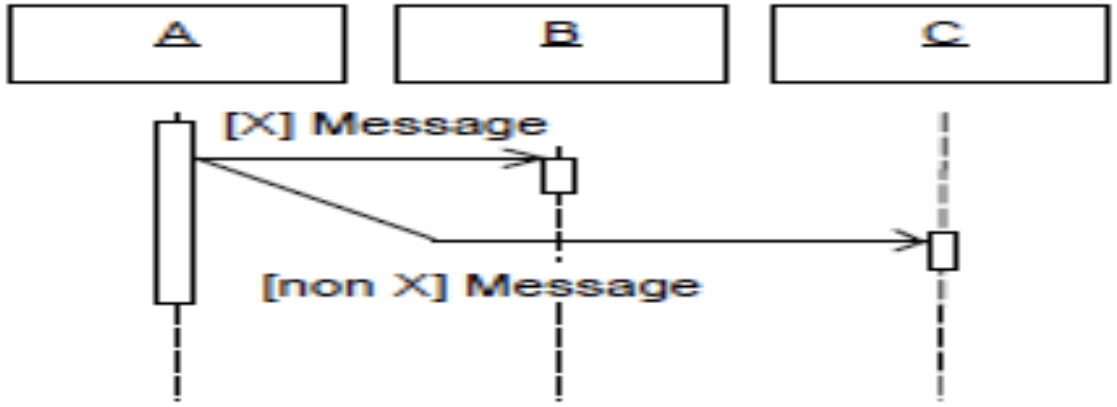

*Représentation alternative de la boucle* **While** *au moyen d'une condition placée devant le message.*

# Structures de controle

Le diagramme suivant montre que l'objet A envoie un message à l'objet B ou à l'objet C selon la condition X.




*Représentation des branchements conditionnels par l'ajout de pseudo-code.*

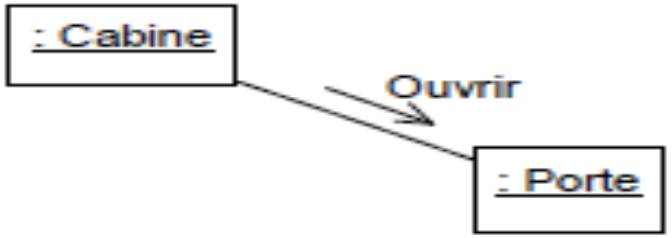



*Représentation graphique des branchements conditionnels.*

# Les Diagrammes de collaboration

Les diagrammes de collaboration montrent des interactions entre objets, en insistant plus particulièrement sur la structure spatiale statique qui permet la mise en collaboration d'un groupe d'objets.

Une interaction est réalisée par un groupe d'objets qui collaborent en échangeant des messages. Ces messages sont représentés le long des liens qui relient les objets, au moyen de flèches orientées vers le destinataire du message.




*Exemple d'interaction entre deux objets. La cabine demande à la porte de s'ouvrir.*

# L'ordre des envois de message

Dans un diagramme de collaboration, le temps n'est pas représenté de manière implicite, comme dans un diagramme de séquence, de sorte que les différents messages sont numérotés pour indiquer l'ordre des envois

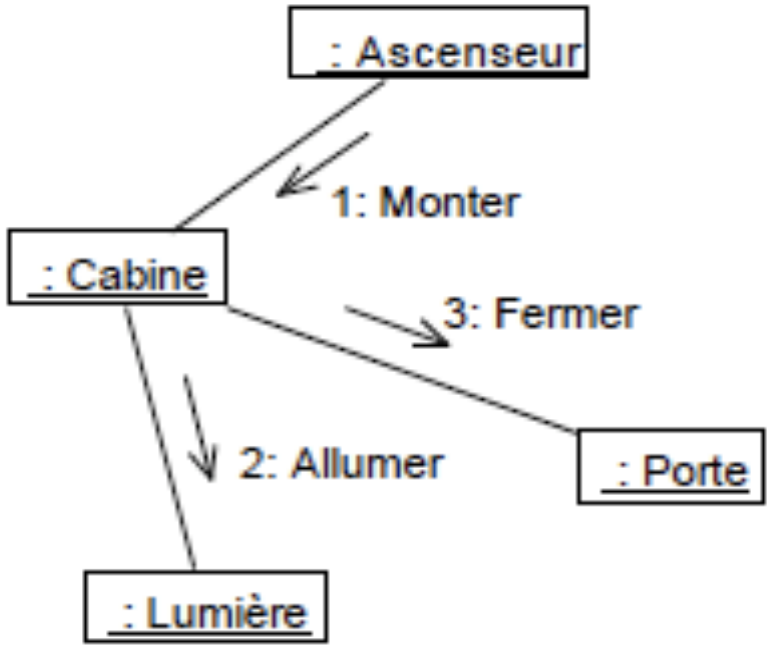



*Figure 212 : Exemple de représentation de l'ordre des envois de message.*

Les diagrammes de collaboration montrent simultanément les interactions entre les objets et les relations structurelles qui permettent ces interactions.

# Interaction entre plusieurs messages

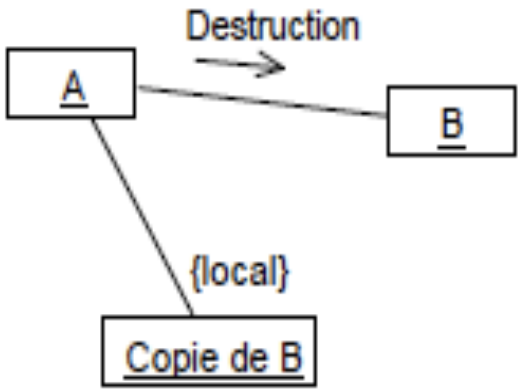



*Figure 213 : Représentation d'une interaction entre trois objets. L'objet* **B** *a été copié localement par l'objet* **A**, *dans le but de permettre l'annulation éventuelle de la destruction de* **B**.

Le diagramme suivant représente le contexte d'un mécanisme pour annuler une opération de destruction. Avant de déclencher l'opération de destruction de l'objet B, l'objet A effectue une copie locale de B de sorte que si l'opération de destruction venait à être annulée, l'objet B puisse être restitué tel qu'il était avant le déroulement de l'interaction.

# Création et destruction des objets et des liens

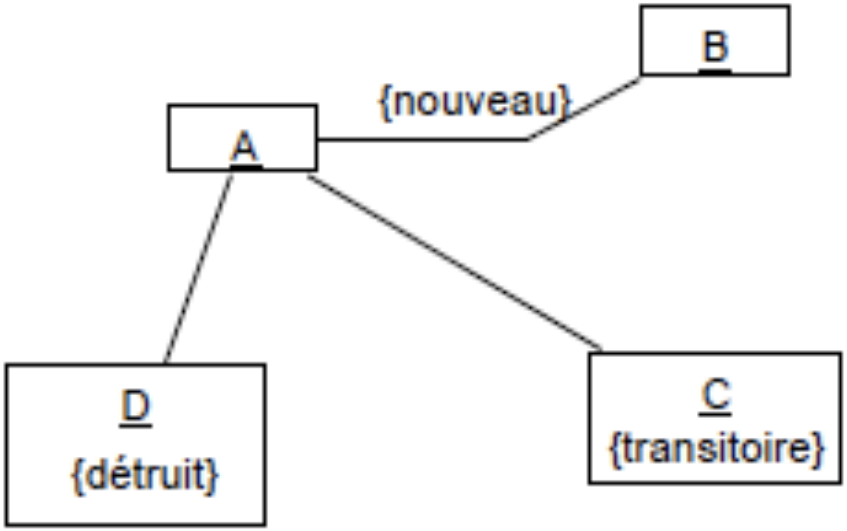



*Figure 214 : Représentation de la création et de la destruction des objets et des liens.*

Les objets et les liens créés ou détruits au cours d'une interaction peuvent respectivement porter les contraintes {nouveau} ou {nouveau}. Les objets créés, puis détruits au sein de la même interaction, sont identifiés par la contrainte {transitoire}.

# Suite

L'exemple suivant montre un instituteur qui demande à tous ses élèves de se lever ; l'itération est indiquée par le caractère * placé devant le message.

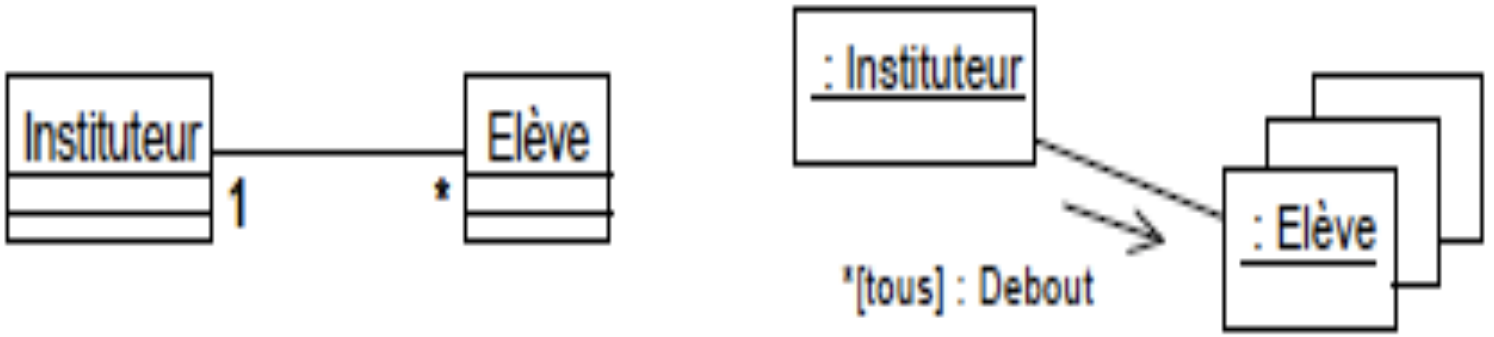



*Figure 215 : Représentation condensée d'une famille de liens, instances d'une même association.*

# La place de l'utilisateur

La notation permet de faire figurer un acteur dans un diagramme de collaboration afin de représenter le déclenchement des interactions par un élément externe au système. Grâce à cet artifice, l'interaction peut être décrite de manière plus abstraite, sans entrer dans les détails des objets de l'interface utilisateur.

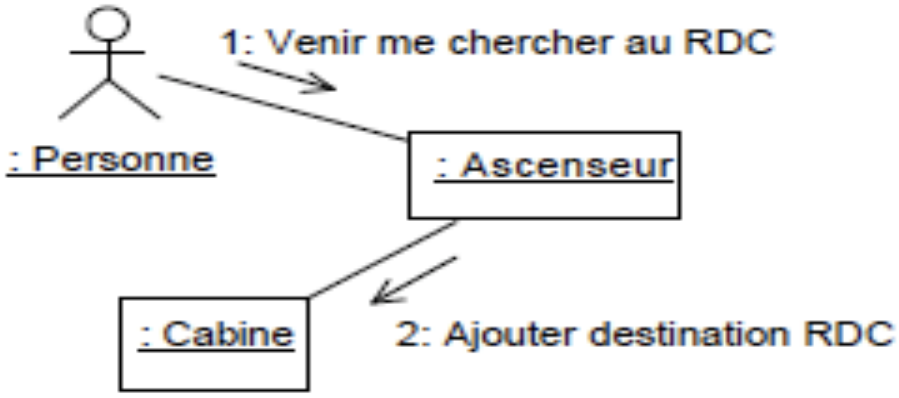



*Figure 216 : Exemple de déclenchement d'une interaction par un acteur.*

Le premier message de l'interaction est envoyé par l'acteur, représenté soit par le symbole graphique des acteurs du modèle des cas d'utilisation, soit par un objet muni d'un stéréotype qui précise sa qualité d'acteur.

# Les objets actifs

Les objets qui possèdent le flot de contrôle sont dits d'actifs. Un objet actif peut activer un objet passif pour le temps d'une opération, en lui envoyant un message. Une fois le message traité, le flot de contrôle est restitué à l'objet actif.

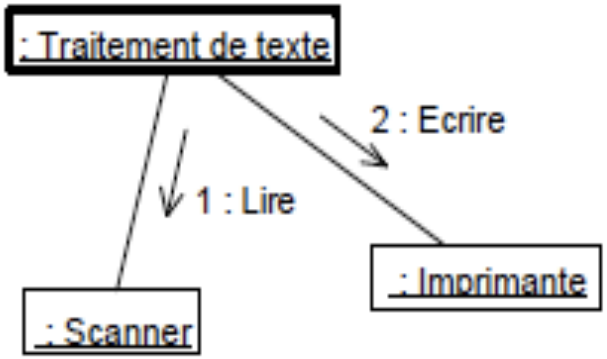



*Figure 217 : Représentation des objets actifs. Le cadre des objets actifs est plus épais que celui des objets passifs.*

Dans un environnement multitâche, plusieurs objets peuvent être actifs simultanément. Un objet actif se représente par un rectangle dont la bordure est plus épaisse que celle des objets passifs.

# Répresentation des messages

Un message se représente par une flèche placée à proximité d'un lien et dirigée vers l'objet destinataire du message.
Un message déclenche une action dans l'objet destinataire.
Les messages respectent la forme générale suivante:

synchronisation séquence ':' résultat ':=' nom arguments

Le message, ses arguments et valeurs de retour, son rang au sein de l'interaction, et diverses autres informations comme le degré d'emboîtement ou la synchronisation sont précisés lors de l'envoi.

# Synchronisation

Le point de synchronisation d'un message est exprimé sous la forme d'une séquence d'envoi de message, terminée par le caractère /

Tous les messages référencés dans cette liste doivent avoir été envoyés pour valider l'envoi du message courant.
La syntaxe d'un point de synchronisation prend la forme suivante

synchronisation ::= rang {',' synchronisation} '/'
rang ::= [entier | nom de flot d'exécution]{'.' rang}

L'entier représente le rang de l'envoi de message au sein de l'emboîtement englobant. Le nom identifie un flot d'exécution parallèle au sein d'un emboîtement. Ainsi, l'envoi de message 3.1.3 suit immédiatement l'envoi 3.1.2 au sein de l'emboîtement 3.1, alors que l'envoi 3.1. a est effectué simultanément à l'envoi 3.1.b

Dans l'exemple suivant, le message Message est envoyé lorsque les envois A.1 et B.3 ont été satisfaits.


*Figure 218 : Exemple de représentation de la synchronisation entre flots d'exécution parallèles.*

# Séquence

La séquence indique le niveau d'emboîtement de l'envoi de message au sein de l'interaction. La séquence est constituée d'une suite de termes séparés par des points. Chaque séquence possède la syntaxe suivante :

séquence ::= rang [récurrence]

La récurrence représente l'itération et les branchements conditionnels ; elle prend les formes suivantes :

récurrence ::= '*' '[' clause d'itération ']' bloc
ou
récurrence ::= '[' clause de condition ']' bloc

La clause d'itération est optionnelle ; elle est exprimée dans un format libre :

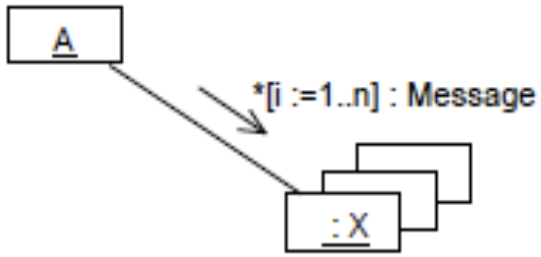



*Figure 219 : Exemple de représentation de l'itération.*

La notation de l'itération sous-entend l'envoi séquentiel des messages contenus par le bloc. L'envoi parallèle (également appelé diffusion) est matérialisé par la suite de caractères *||.

# Séquence (Suite)

La clause de condition valide ou non l'envoi des messages contenus par le bloc.
La clause de condition est exprimée dans un format libre :

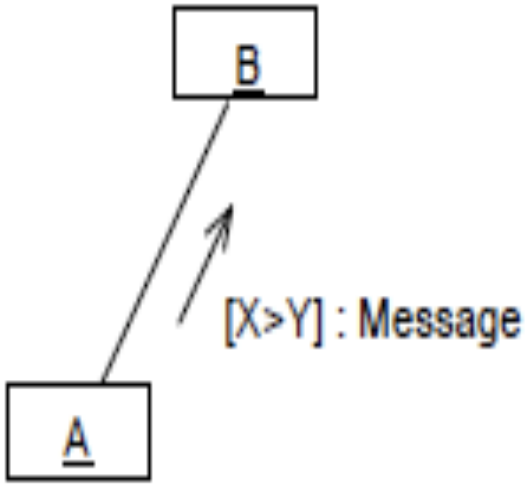



*Figure 220 : Exemple de représentation d'un envoi de message conditionnel.*

# Résultats et noms

Résultat: Le résultat est constitué d'une liste de valeurs retournées par le message.
Ces valeurs peuvent être utilisées comme paramètres des autres messages compris dans l'interaction.

Ce champ n'existe pas en l'absence de valeurs retournées. Le format de ce champ est libre :

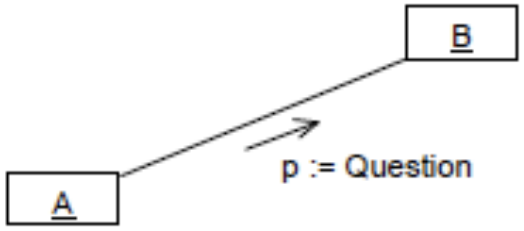



*Figure 221 : Représentation de la récupération d'un résultat.*

Nom: Il s'agit du nom du message. Souvent ce nom correspond à une opération définie dans la classe de l'objet destinataire du message.

# Exemples

Les expressions suivantes donnent quelques exemples de la syntaxe d'envoi des messages :

```
4 : Afficher (x, y) -- message simple

3.3.1 : Afficher (x, y) -- message imbriqué

4.2 : âge := Soustraire (Aujourd'hui,
DateDeNaissance)
-- message imbriqué avec valeur retournée

[Age >= 18 ans] 6.2 : Voter () -- message
conditionnel

4.a, b.6 / c.1 : Allumer (Lampe) --
synchronisation avec d'autres flots d'exécution

1 * : Laver () -- itération

3.a, 3.b / 4 *||[i := 1..n] : Eteindre () --
itération parallèle
```

# Syntaxe

- **pré / [cond] séq *|| [iter] : r := msg (par)**
- **pré** : prédécesseurs (liste de numéros de séquence de messages): Indique que le message ne sera envoyé que lorsque tous ses prédécesseurs le seront
- **[cond]** : expression booléenne: Permet de conditionner l'envoi du message
- **séq** : numéro de séquence du message: Indique le rang du message, c'est-à-dire son numéro d'ordre par rapport aux autres messages.

  Il est possible de représenter le niveau d'emboîtement des messages et leur précédence, à l'aide de chiffres séparés par des points.
  Exemple : l'envoi du message 1.3.5 suit immédiatement celui du message 1.3.4 et ces deux messages font partie du flot (de la famille de messages) 1.3.

  Pour représenter l'envoi simultané de deux messages, il suffit de les indexer par une lettre.
  Exemple : l'envoi des messages 1.3.a et 1.3.b est simultané.
- **iter** : récurrence du message: Permet de spécifier l'envoi séquentiel (ou en parallèle, avec "||") de messages. Il est aussi possible de spécifier qu'un message est récurrent en n'utilisant que "*" ou "*||".
- **r** : valeur de retour du message: Permet d'affecter la valeur de retour d'un message
- **msg** : nom du message.
- **par** : paramètres du message

# Exemples

- `3 : bonjour()`
- `[heure = midi] 1 : manger()`
- `1.3.6 * : ouvrir()`
- `3 / *|||[i := 1..5] : fermer()`
- `1.3,2.1 / [t < 10s] 2.5 : age := demanderAge(nom,prenom)`
- ```
  1.3 / [disk full] 1.7.a * : deleteTempFiles()
  1.3 / [disk full] 1.7.b : reduceSwapFile(20%)
  ```

  .
  ```

# Exemples

- `3 : bonjour()`
  **Ce message a pour numéro de séquence "3 ».**

- `[heure = midi] 1 : manger()`

- `1.3.6 * : ouvrir()`

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

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

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

# Exemples

- `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()`

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

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

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

# Exemples

- `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.**

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

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

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

  .
  ```

# Exemples

- `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.**

- `3 / *||[i := 1..5] : fermer()`
  **Représente l'envoi en parallèle de 5 messages. Ces messages ne seront envoyés qu'après l'envoi du message 3.**

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

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

# Exemples

- `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.**

- `3 / *||[i := 1..5] : fermer()`
  **Représente l'envoi en parallèle de 5 messages. Ces messages ne seront envoyés qu'après l'envoi du message 3.**

- `1.3,2.1 / [t < 10s] 2.5 : age := demanderAge(nom,prenom)`
  **Ce message (numéro 2.5) ne sera envoyé qu'après les messages 1.3 et 2.1, et que si "t < 10s". La valeur de retour est affectée à age. Le message prend nom et prenom en paramètres**

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

# Exemples

- `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.**

- `3 / *||[i := 1..5] : fermer()`
  **Représente l'envoi en parallèle de 5 messages. Ces messages ne seront envoyés qu'après l'envoi du message 3.**

- `1.3,2.1 / [t < 10s] 2.5 : age := demanderAge(nom,prenom)`
  **Ce message (numéro 2.5) ne sera envoyé qu'après les messages 1.3 et 2.1, et que si "t < 10s". La valeur de retour est affectée à age. Le message prend nom et prenom en paramètres**

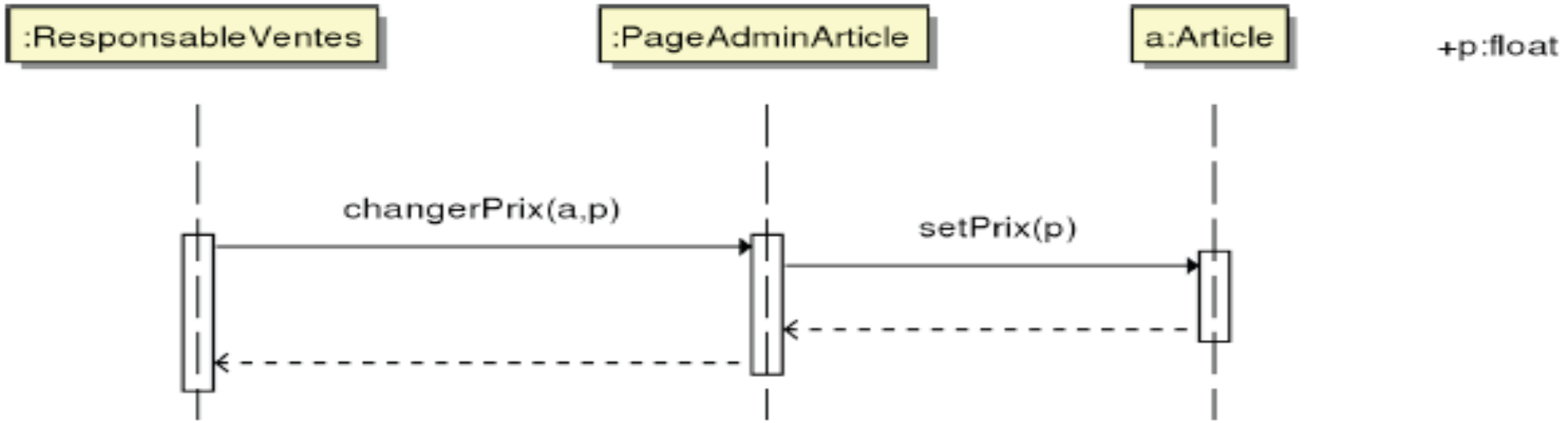
- `1.3 / [disk full] 1.7.a * : deleteTempFiles()`
  `1.3 / [disk full] 1.7.b : reduceSwapFile(20%)`

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

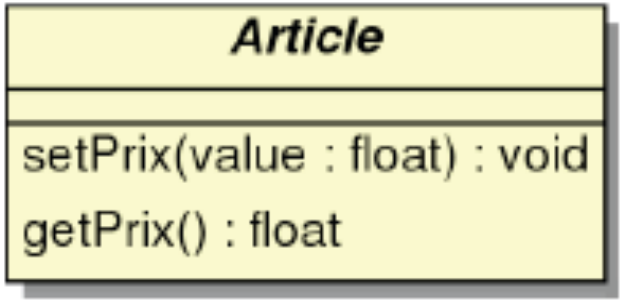
# Quelques exemples d’interaction entre les diagramme

## Cas d'utilisation :

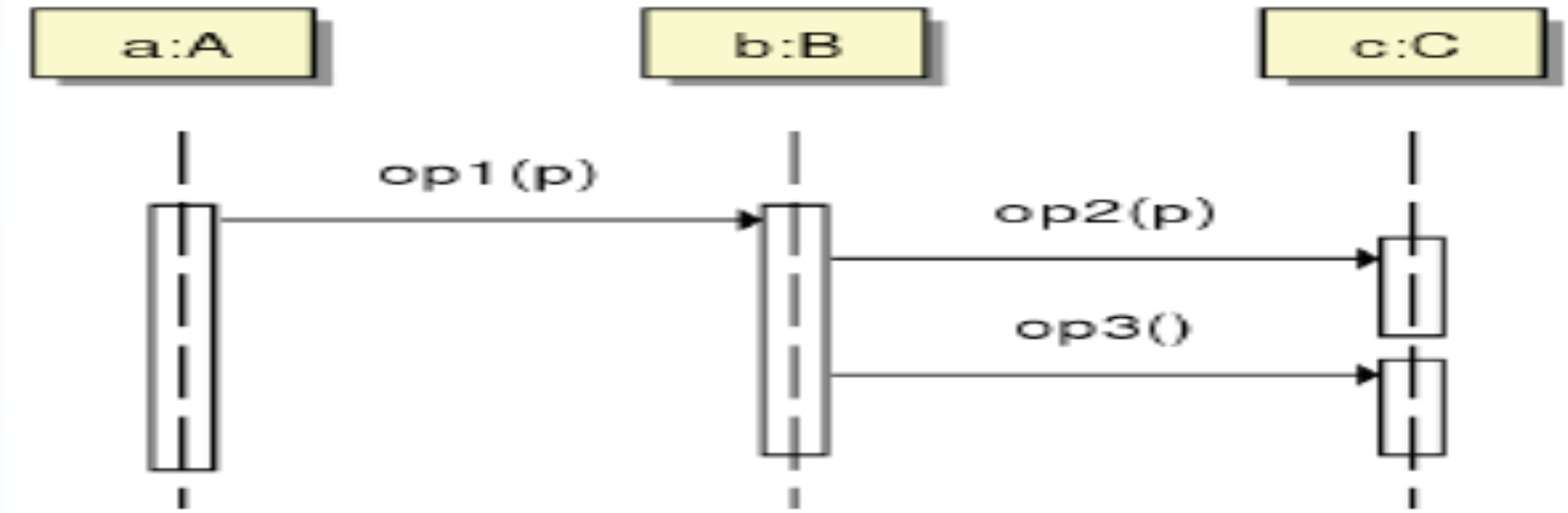

## Diagramme de séquences correspondant :



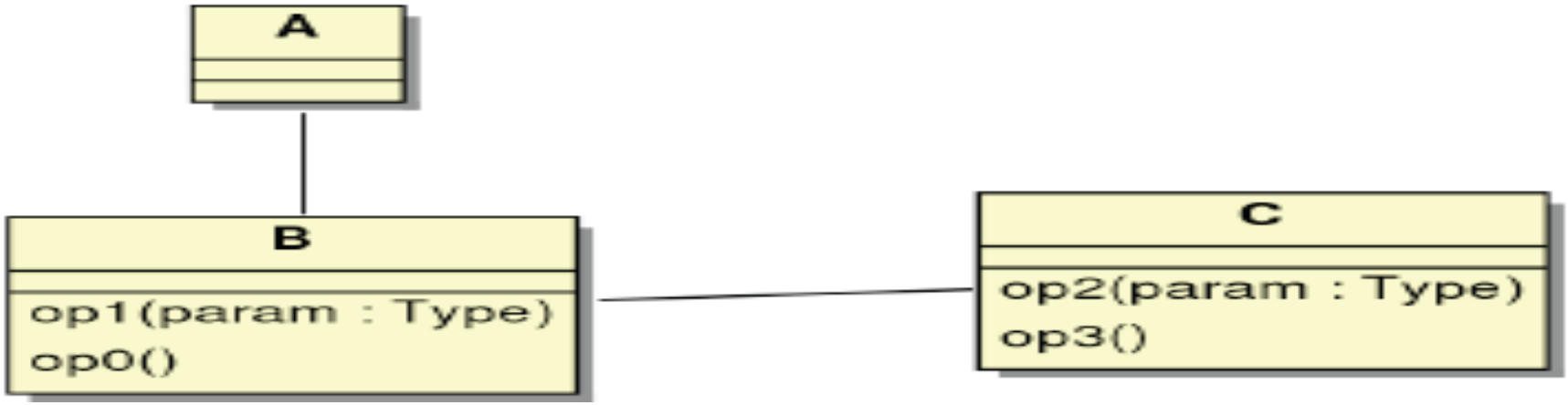

## Opérations nécessaires dans le diagramme de classes :



# Message synchrones et Diagramme de classes




Envoyer un message et attendre la réponse pour poursuivre son activité revient à invoquer une méthode et attendre le retour pour poursuivre ses traitements.



# Message asynchrone et Diagramme de classes


Les signaux sont des objets dont la classe est stéréotypée « signal » et dont les attributs (porteurs d'information) correspondent aux paramètres du message.