III. Diagramme de séquences

1) Définition :

Un diagramme de séquences décrit comment les éléments d'un système interagissent entre eux et avec les acteurs du système. Cette interaction prend la forme d'une chronologie de messages échangés entre les objets (instances de classes ou autres entités appartenant au système) et les acteurs.

De manière générale, chaque diagramme de séquences décrit un cas d'utilisation :

- d'un point de vue interne du fonctionnement du système
- au niveau de l'instance
- suivant des messages échangés de façon chronologique entre les acteurs et les objets du système

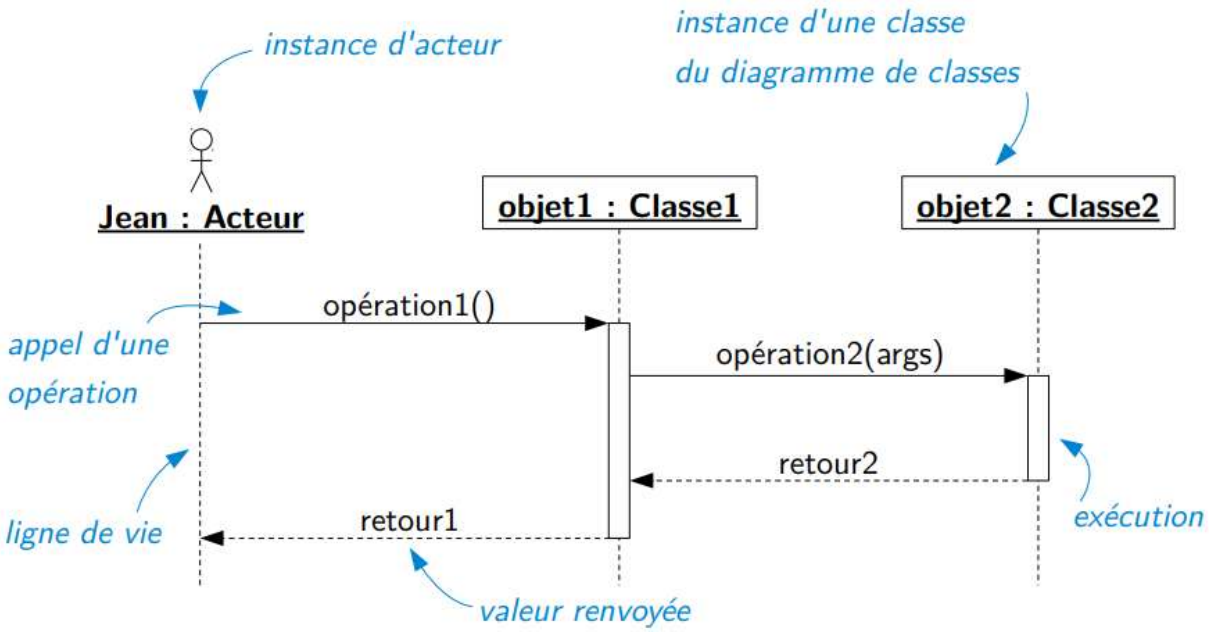
2) Eléments d'un diagramme de séquences :

a. Vu générale d'un diagramme de séquences

Un diagramme de séquence est constitué de :

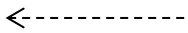
- Acteur : celui qui réalise le cas d'utilisation et donc qui lance l'action souhaitée
- Objets (instances) : il s'agit de tous les objets impliqués dans les opérations qui permettent de réaliser dans un ordre précis le cas d'utilisation
- Messages : il s'agit d'appel d'opérations et éventuellement de valeurs de retour (à l'issue d'une opération)

Chaque acteur et objet est représenté avec une ligne de vie. Les messages sont échangés entre les lignes de vie. Les messages (appels d'opérations) sont représentés par des flèches. Le temps d'exécution d'une opération est représenté par un rectangle (on parle aussi de *période d'activité*).



Remarques :

- Une opération peut contenir des arguments
- Une réponse à une opération est toujours représentée par une flèche en pointillée :



- La réponse peut être simple (valeur de retour uniquement) ou détaillée suivant la syntaxe:

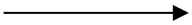
```
attributCible = nomMessageSynchroneInitial(LISTE_PARAMS) :
valeurRetour
```

- Les messages de retour sont optionnels
- Le nom des objets peut être omis sur le diagramme
- Il est possible de numéroter les messages *(certains outils le font comme StarUML)*

b. Types de messages

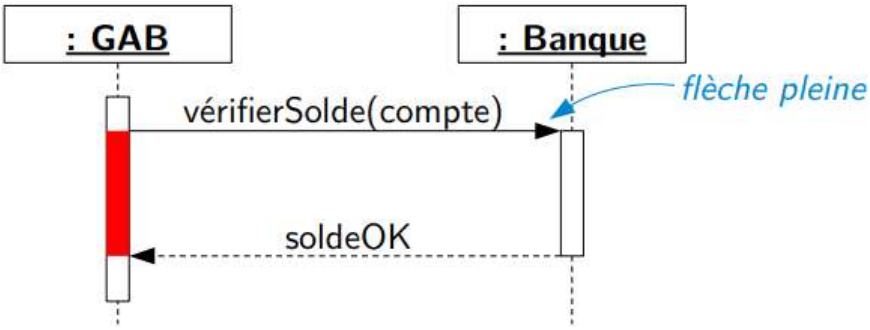
Parmi les messages qu'on peut avoir sur un diagramme de séquence :

- **Message synchrone** : est représenté par une flèche pleine. Ici, l'émetteur du message est bloqué en attendant la fin de l'exécution de l'opération.

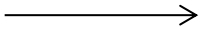


Les messages synchrones attendent toujours la fin de l'opération avant de reprendre leur comportement. On utilise le plus souvent un message de retour (en pointillés) marquant la reprise du contrôle par l'objet émetteur du message synchrone.

Exemple :

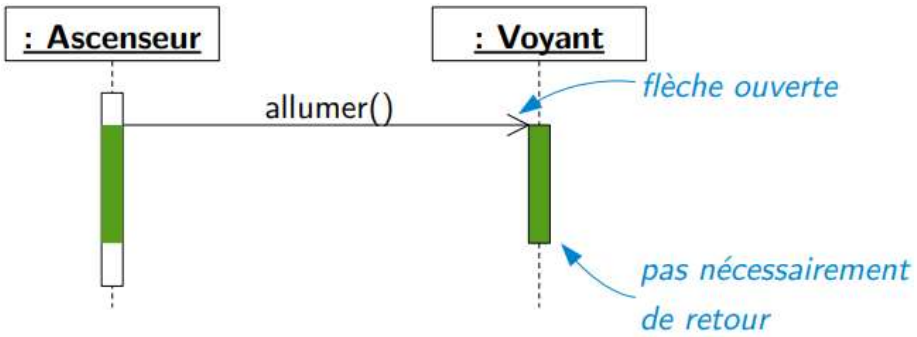


- **Message asynchrone** : est représenté par une flèche ouverte.

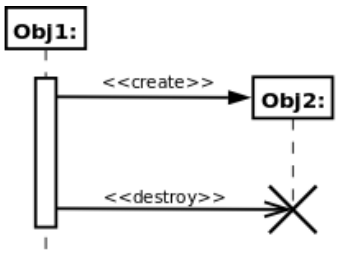


Avec les messages asynchrones, l'émetteur n'attend pas de réponse du destinataire, mais poursuit ses processus sans interruption.

Exemple :

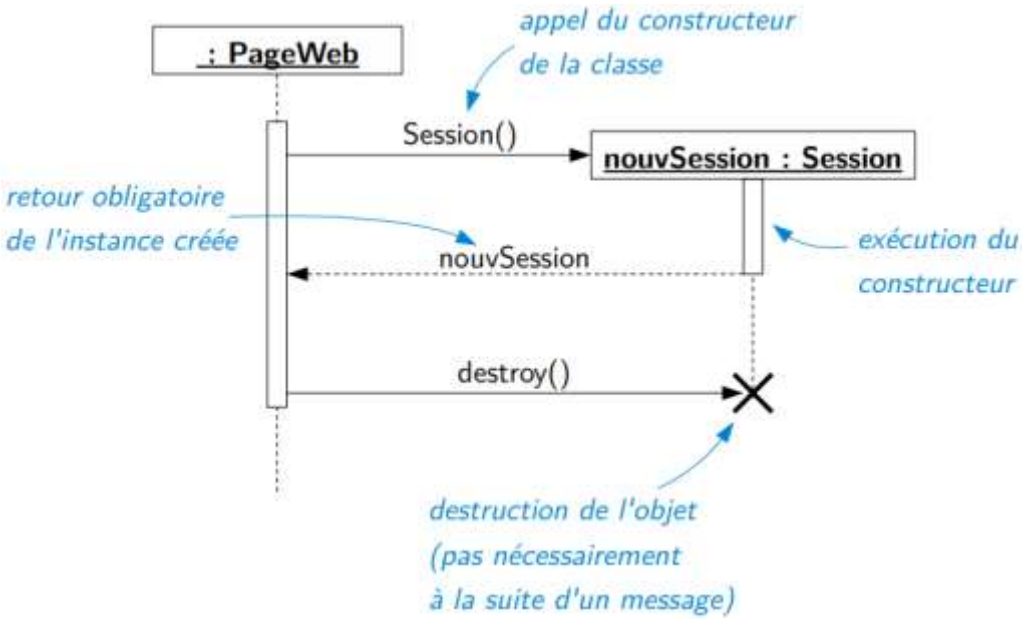


- **Message de création et de destruction** (***create / destroy***) : est un type de message qui signale une nouvelle instance d'une ligne de vie. Le système crée un nouvel objet dans le diagramme de séquence. La création de l'objet est matérialisée par une flèche qui pointe sur le sommet d'une ligne de vie.



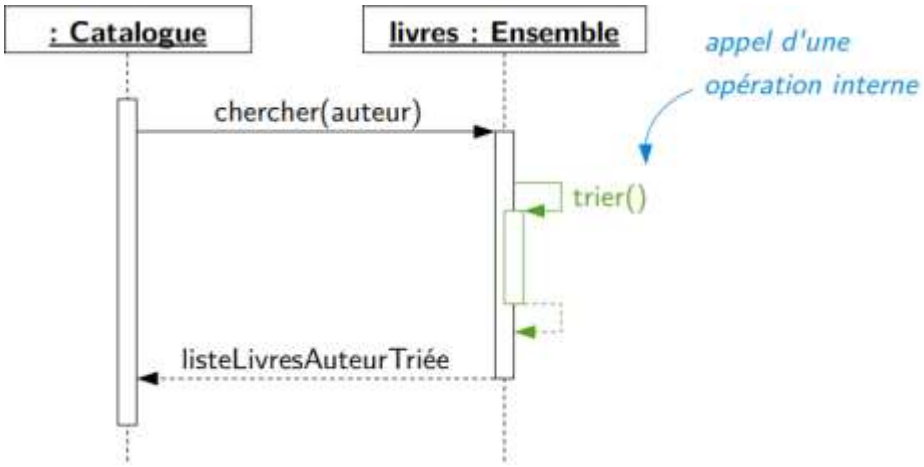
La destruction d'un objet est matérialisée par une croix qui marque la fin de la ligne de vie de l'objet. La destruction d'un objet n'est pas nécessairement consécutive à la réception d'un message.

Exemple :



- **Message réflexif** : un message réflexif est un message dont l'émetteur et le destinataire sont le même objet

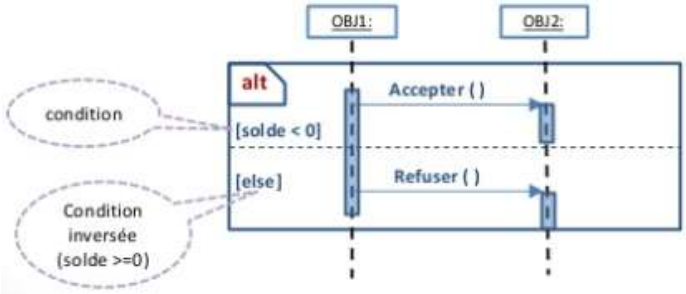
Exemple :



c. Les fragments combinés :

- **Fragment alt** : il s'agit d'un opérateur conditionnel. Il permet de montrer les différentes alternatives d'exécution suivant des conditions précises. Les différentes alternatives sont spécifiées dans des zones délimitées par des pointillés.
    - Les conditions sont spécifiées entre crochets dans chaque zone
    - On peut utiliser une clause **[else]** (ou on spécifie les autres conditions)

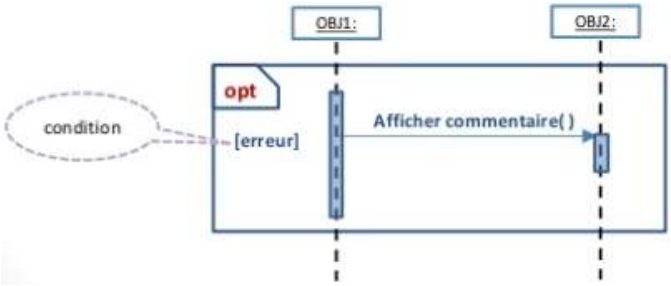
  Exemple :



  Cet opérateur est l'équivalent de l'opérateur conditionnel ***if*** ou ***switch***.

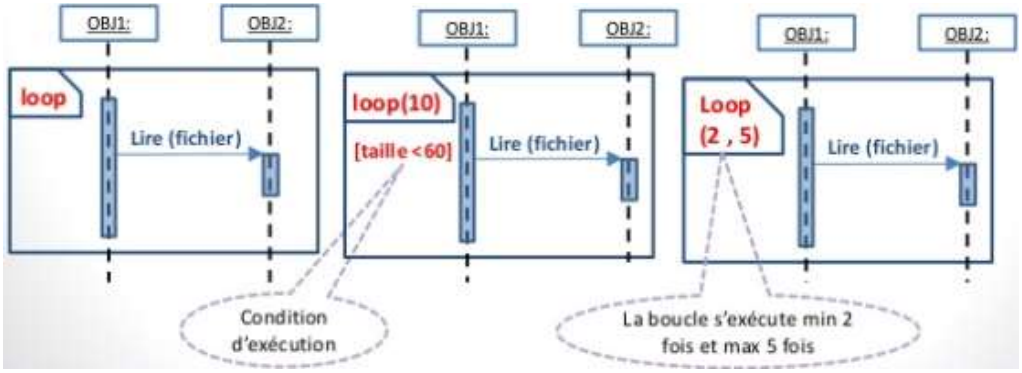
- **Fragment opt :** il représente un comportement qui peut se produire ou pas (il s'agit d'une option). Il est l'équivalent d'un if sans else.

  Exemple :



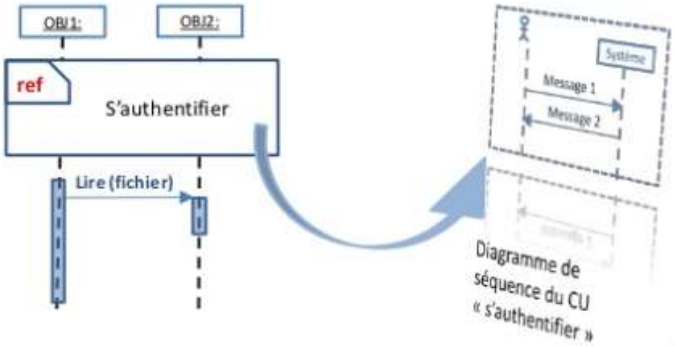
- **Fragment loop :** il décrit un ensemble d'interactions qui s'exécutent en boucle. La condition pour rester dans la boucle est spécifiée entre crochet. On peut aussi spécifier le nombre de répétition exacte ou le nombre minimum et maximum de répétition entre parenthèse

Exemples :



- **Fragment ref :** il est utilisé pour faire référence à un autre diagramme de séquences existant. Son rôle est de factoriser des parties de comportements utilisées dans plusieurs scénarios (et donc plusieurs cas d'utilisation). Il permet aussi d'alléger le diagramme (dans le cas d'un scénario complexe par exemple).

  Exemple :



Remarque :

Il est possible de combiner plusieurs fragments dans le même diagramme.

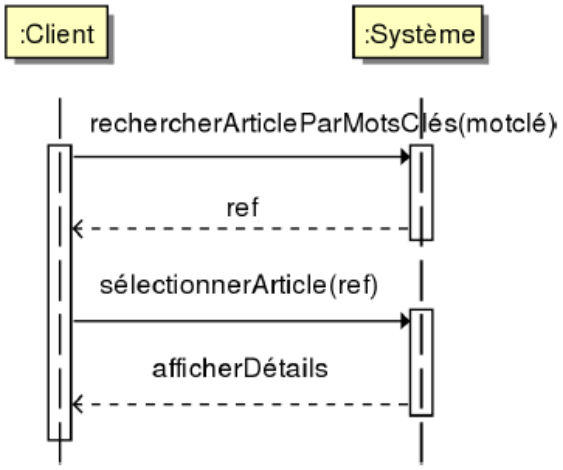
3) Diagramme de séquences système (DSS)

Un DSS permet de spécifier l'enchainement des opérations entre les acteurs et le système tel que dans un scénario d'un cas d'utilisation. Le système est considéré ici comme un tout :

- On décrit ses interactions avec les acteurs
- On utilise une classe ***système*** qui (à part les acteurs) donnera lieu à une seule ligne de vie dans le DSS

Les DSS permettent d'alléger les diagrammes de séquences et avancer plus vite dans la conception d'application. Ils peuvent être enrichis par la suite.

Exemple :



4) Exemple d'application

Un site web permet de gérer des catalogues de produits pour la vente. Nous voulons représenter le cas d'utilisation « ajouter article ».

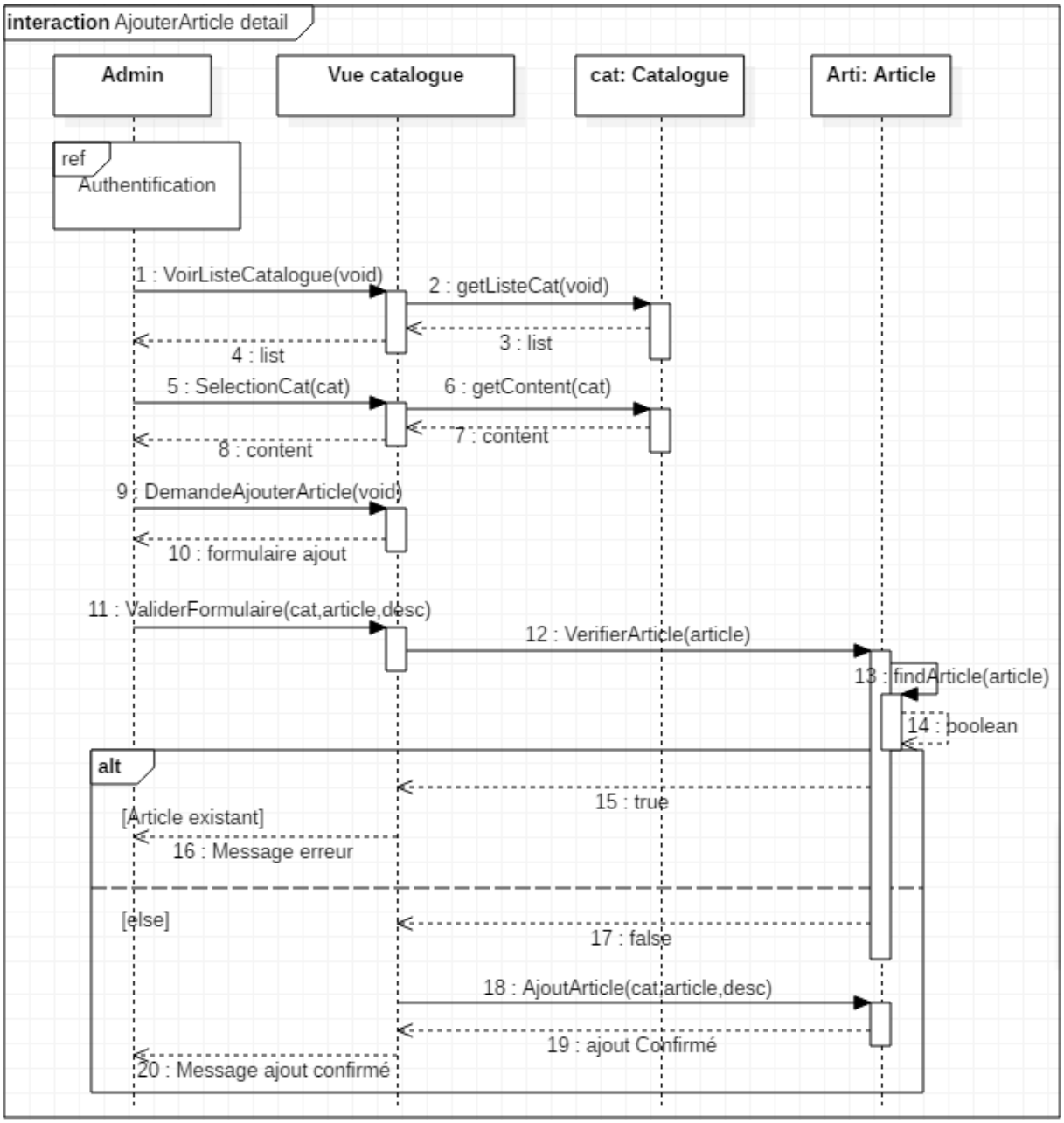
Pour ajouter un article, l'administrateur du site doit d'abord consulter la liste des catalogues après authentification, il choisit ensuite de saisir un nouvel article dans un catalogue. Si l'article saisi existe déjà, le système le notifiera à l'administrateur, sinon l'article sera ajouté et un message de confirmation sera affiché à l'admin.

Voici le diagramme de séquence de cet énoncé :

- On considère **Catalogue** et **Article** des classes du système.
- La **vue catalogue** est au fait l'interface (page web ou autre) entre l'utilisateur et le système qui permet d'accéder aux catalogues.

Remarques :

- Dans le cas de l'utilisation du modèle MVC, il faut ajouter un objet **contrôleur** entre la vue et la classe catalogue
- Dans le diagramme ci-dessous, on peut également ajouter un objet **BD** (base de données) et montrer ainsi l'interaction des objets du système avec la base de données (récupération de données, ajout,…)
- Pour simplifier, la vérification des erreurs ***après remplissage du formulaire*** ne figure pas dans le diagramme ci-dessous (*sinon, il faut mettre un fragment alt après l'opération 11, pour faire le contrôle d'erreurs, dans le cas de champs vides*).


interaction AjouterArticle detail
Admin
Vue catalogue
cat: Catalogue
Arti: Article
ref
Authentification
1 : VoirListeCatalogue(void)
2 : getListeCat(void)
3 : list
4 : list
5 : SelectionCat(cat)
6 : getContent(cat)
7 : content
8 : content
9 : DemandeAjouterArticle(void)
10 : formulaire ajout
11 : ValiderFormulaire(cat,article,desc)
12 : VerifierArticle(article)
13 : findArticle(article)
14 : boolean
alt
15 : true
[Article existant]
16 : Message erreur
[else]
17 : false
18 : AjoutArticle(cat,article,desc)
19 : ajout Confirmé
20 : Message ajout confirmé

Voici aussi le DSS correspondant au même énoncé :