

Diagrammes de séquences

Diagrammes de communication

(et un zeste de diagramme des cas d'utilisation)



Isabelle BLASQUEZ
@iblasquez

2024



Isabelle BLASQUEZ



[@iblasquez](https://twitter.com/iblasquez)

Enseignement : Génie Logiciel

Recherche : Développement logiciel agile



ICSTUG #IUTAgile

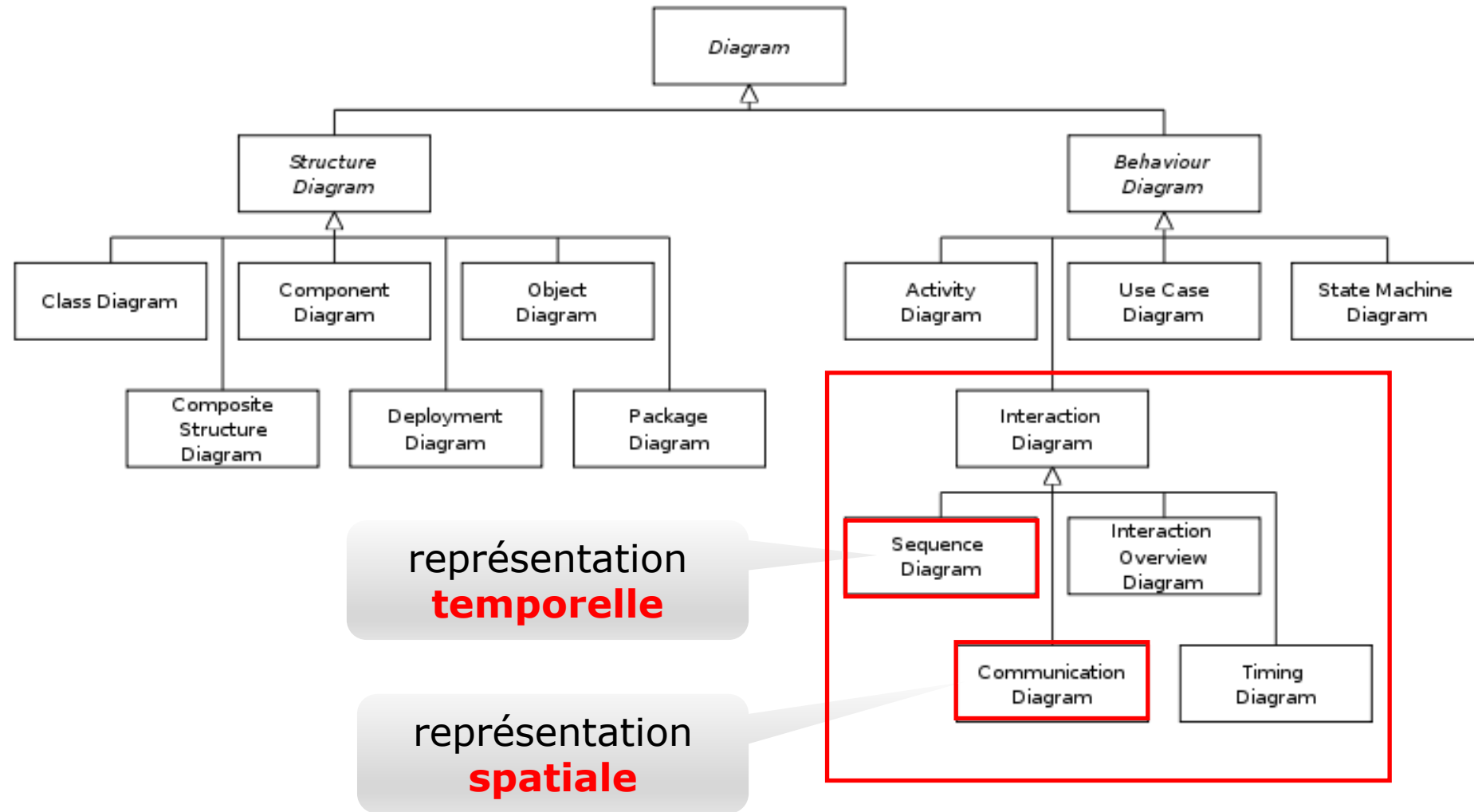


CodeWeek. 

**#Software
Craftsmanship**

 **IUSEOMIX** LIMOUSIN

De l'interaction avec le diagramme de séquence (et de communication)

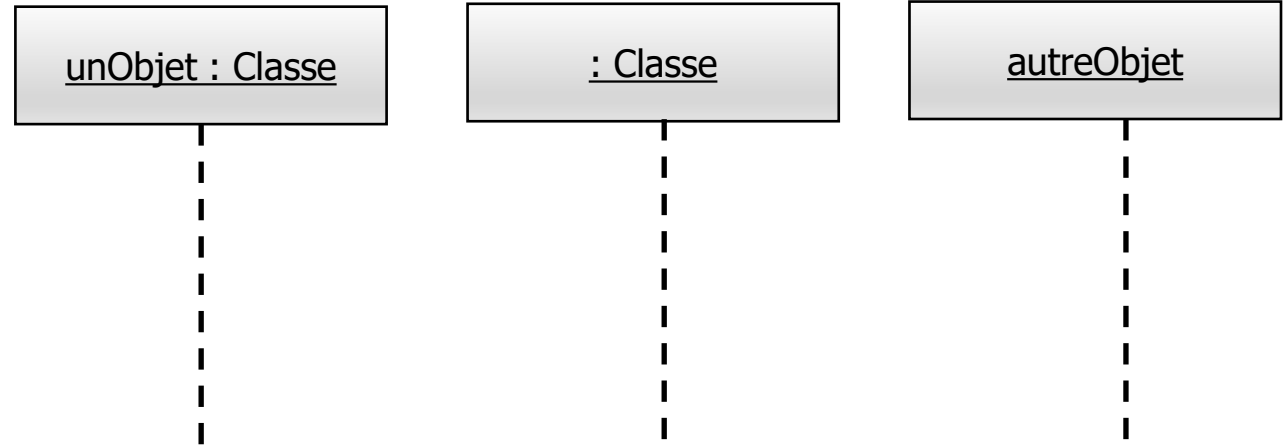


Représentation graphique

Introduction au Diagramme de Séquence (DS)

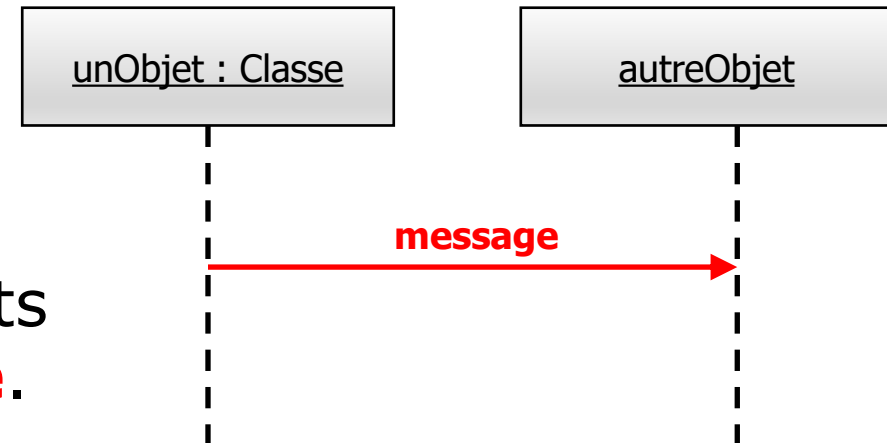
Le diagramme de séquence est un **diagramme d'interaction**.

Un objet est représenté par un **rectangle**



Sa ligne de vie est représentée par une **ligne en pointillée**

Une interaction modélise un **comportement dynamique** entre objets qui se traduit par un envoi de **message**.



Notion de message

Message : communication unidirectionnelle entre objets
qui transporte l'information avec l'intention de déclencher
une activité chez le récepteur.

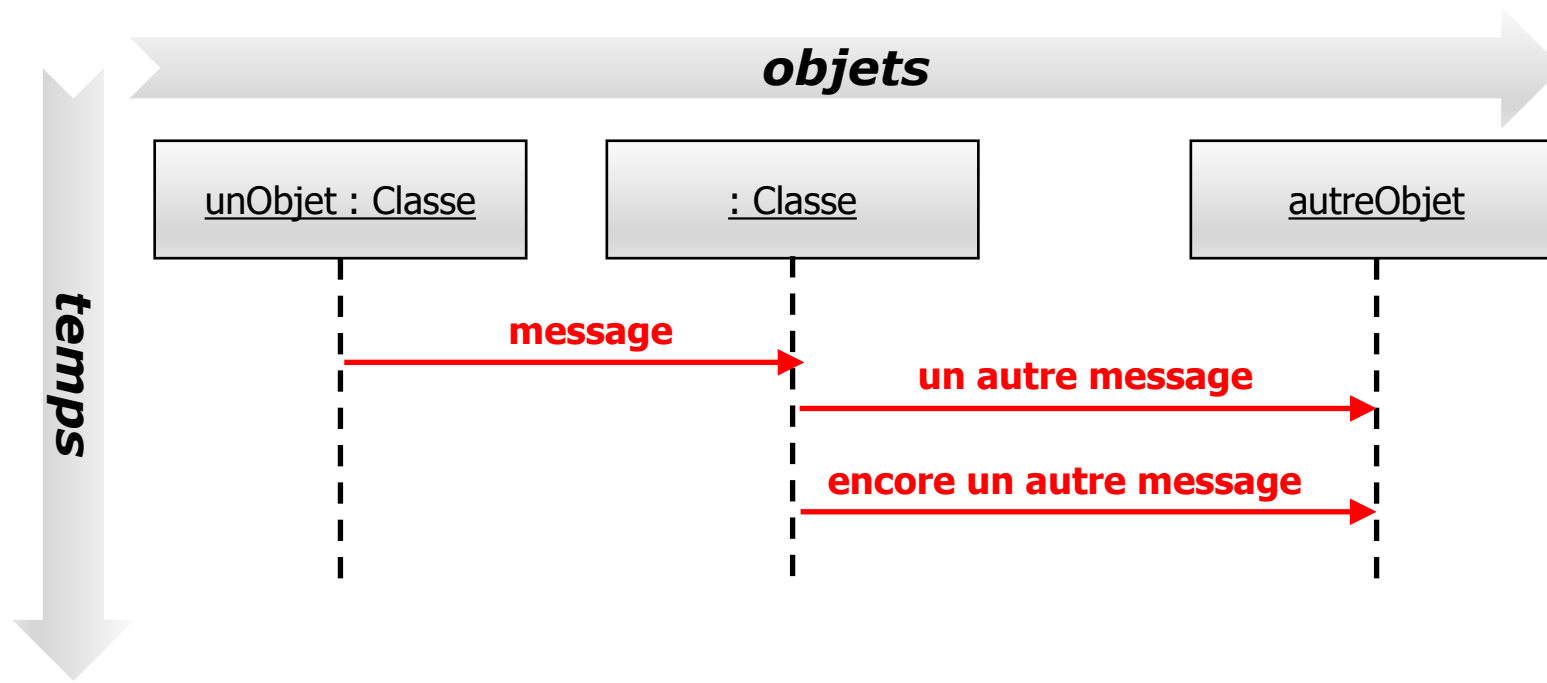
Un **message simple** : 

Expression syntaxique complète d'un **message** :

Paramètre de retour := numéro de séquence : nom du message (Liste des paramètres)

Dimension temporelle du DS

Un diagramme de séquence est constitué d'une **séquence d'interactions** respectant un **point de vue temporel**.

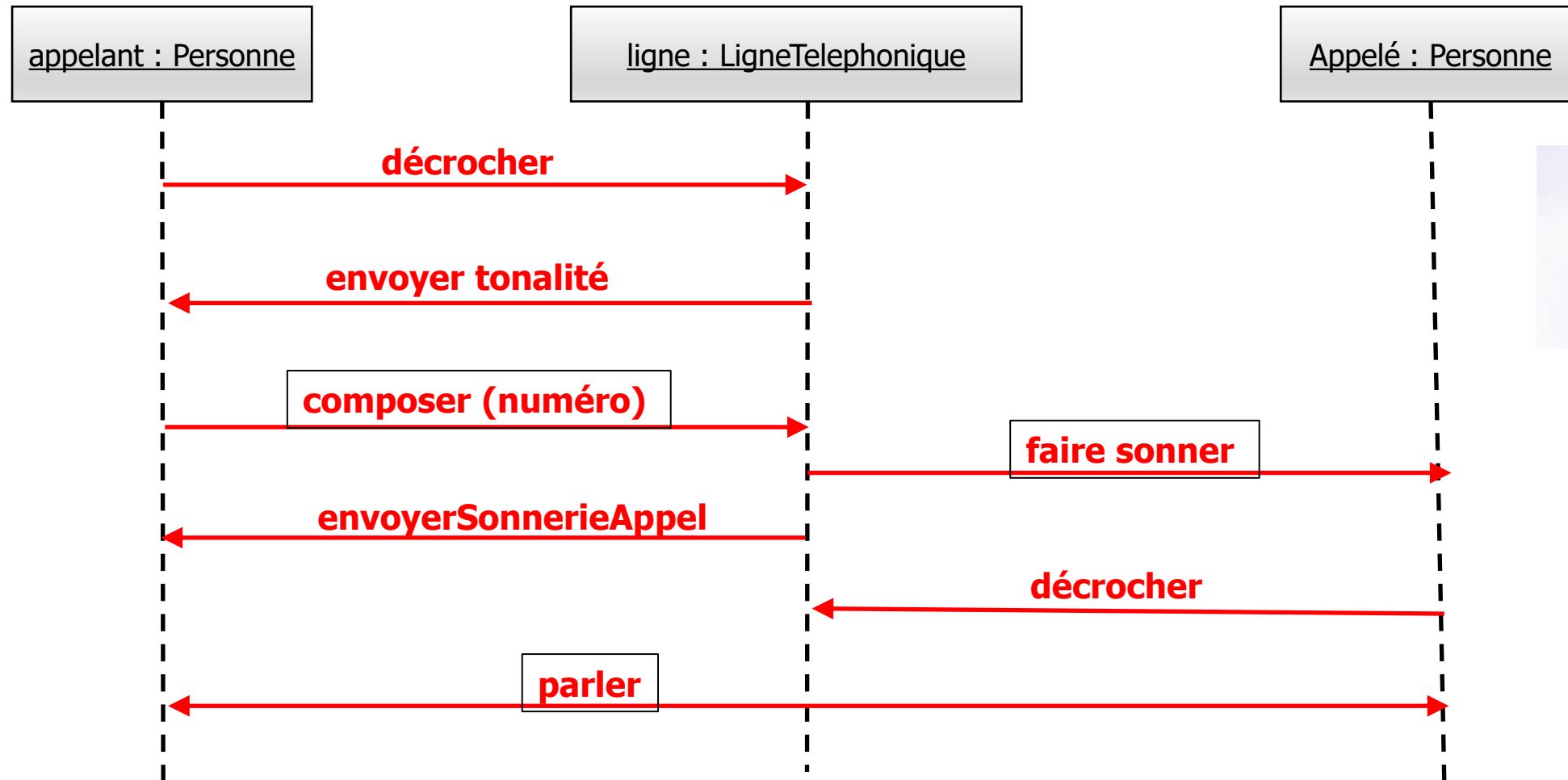


Un diagramme de séquence a **deux dimensions**

- Dimension verticale : **le temps** (ordre indique ordre d'envoi des messages)
- Dimension horizontale : **les objets** (peu importe l'ordre)

Exemple de diagramme de séquence

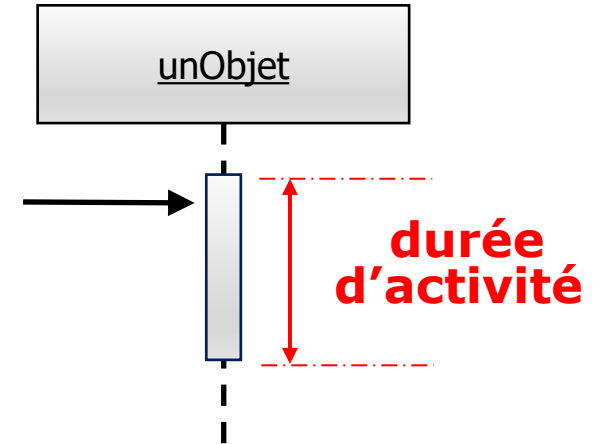
Exemple d'une communication téléphonique sur un téléphone *vintage*
(flot nominal => où tout se passe bien)



Les activations (ou Focus of Control)

Une période d'activité est représentée par une bande rectangulaire placée sur la ligne de vie.

Elle correspond au temps pendant lequel un objet effectue une action.



La période d'activité est également appelée
« activation » ou « focus of control ».

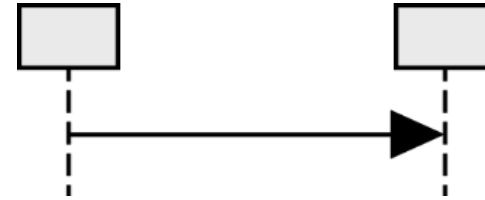
(période sans interruption durant laquelle les différents messages s'enchaînent)

La représentation du focus est optionnelle.

Les messages (1/3) : les classiques ...

Message synchrone (CALL)

→ bloque l'émetteur qui attend jusqu'à la réception d'une réponse (Exemple : appel d'une *opération*)
(le plus couramment utilisé)



Message asynchrone (SEND)

→ l'émetteur continue sans attendre la réponse du récepteur (émetteur non bloqué). C'est un *signal*



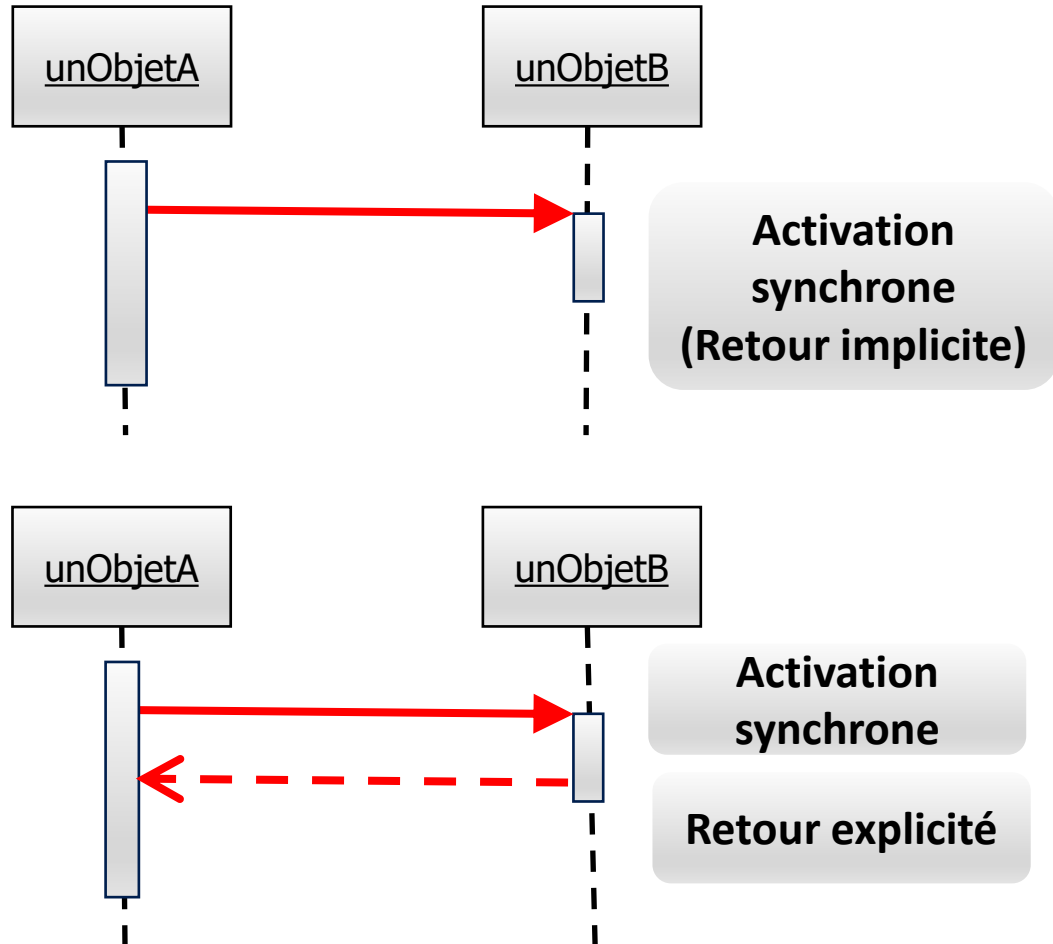
Message de retour (RETURN)

→ pas forcément représenté si le retour est explicite (comme par exemple pour message synchrone)

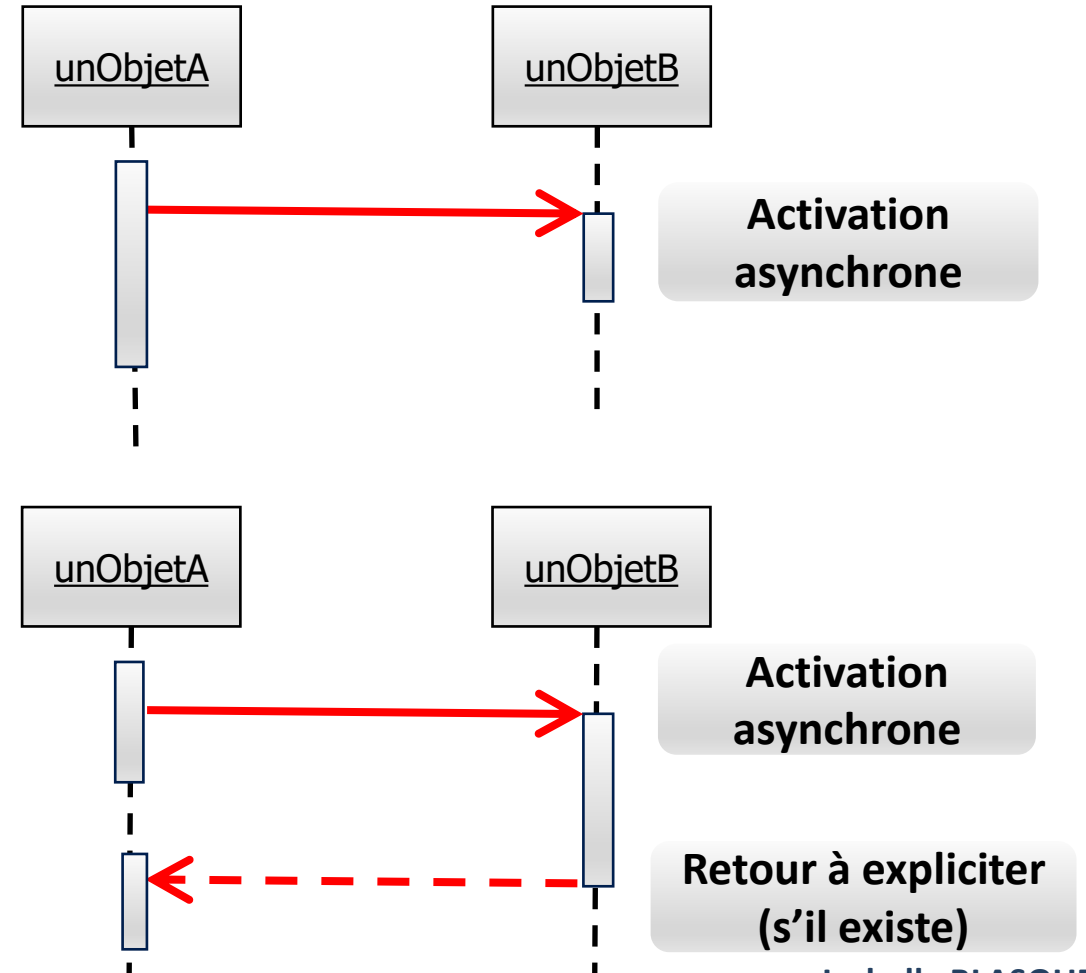


Le retour dans les messages synchrone & asynchrone

Message synchrone (CALL)

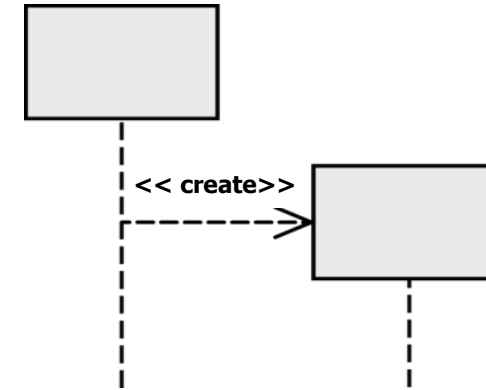
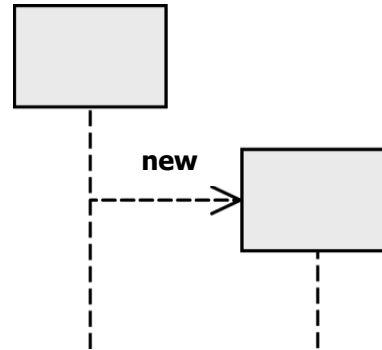


Message asynchrone (SEND)

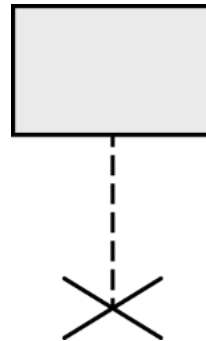


Les messages (2/3) : création et destruction d'objet

Création d'objet



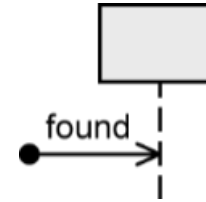
Destruction d'objet



Les messages (3/3) : moins couramment utilisés

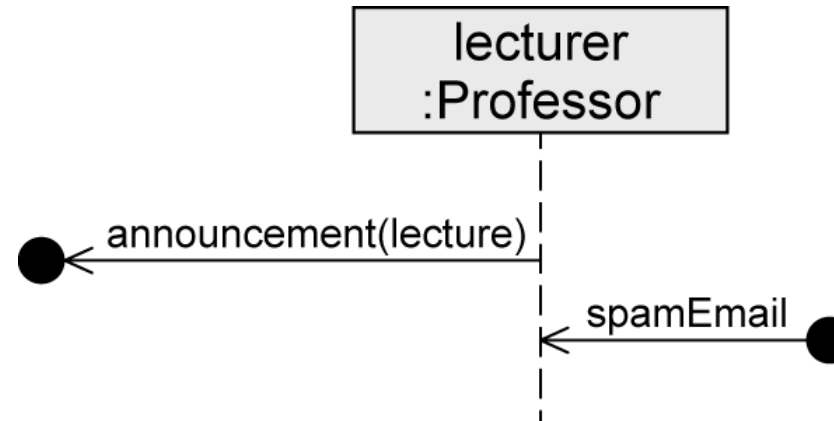
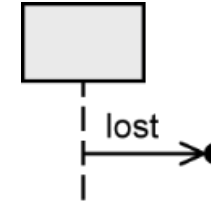
Message trouvé (*found*)

→ l'émetteur du message n'est pas connu



Message perdu (*lost*)

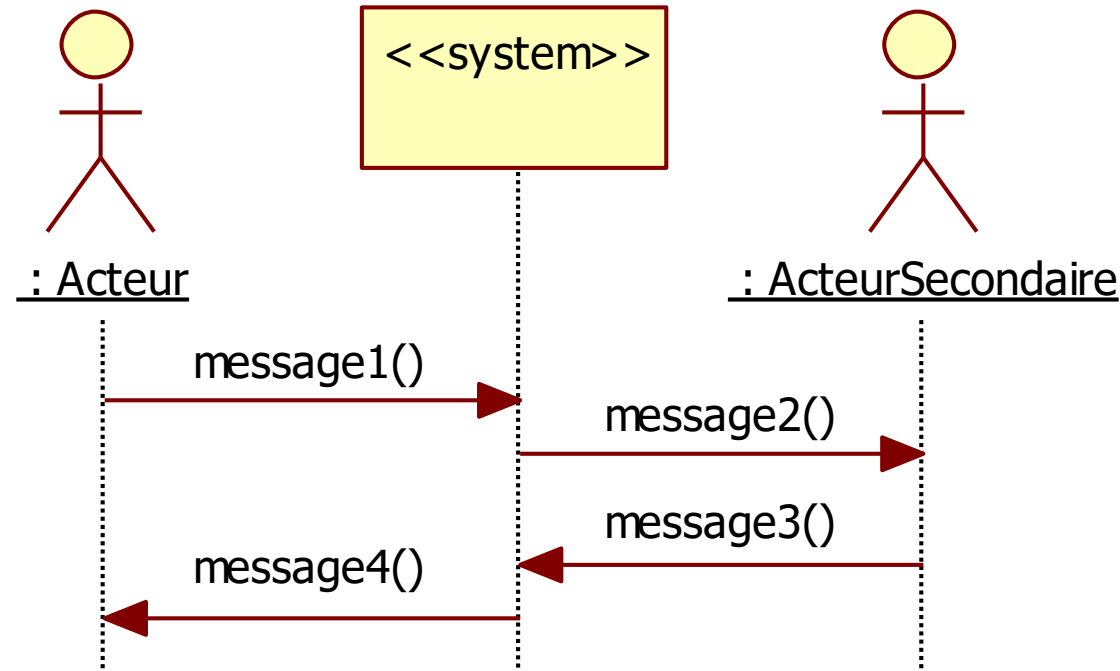
→ le récepteur du message n'est pas connu



Le diagramme de Séquence Système (DSS)

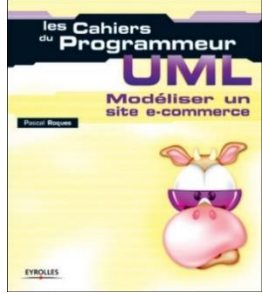
Diagrammes de Séquence Système (DSS)

Proposé par Larman, les **Diagrammes de Séquence Système (DSS)** montrent les **acteurs** qui interagissent directement avec le système.

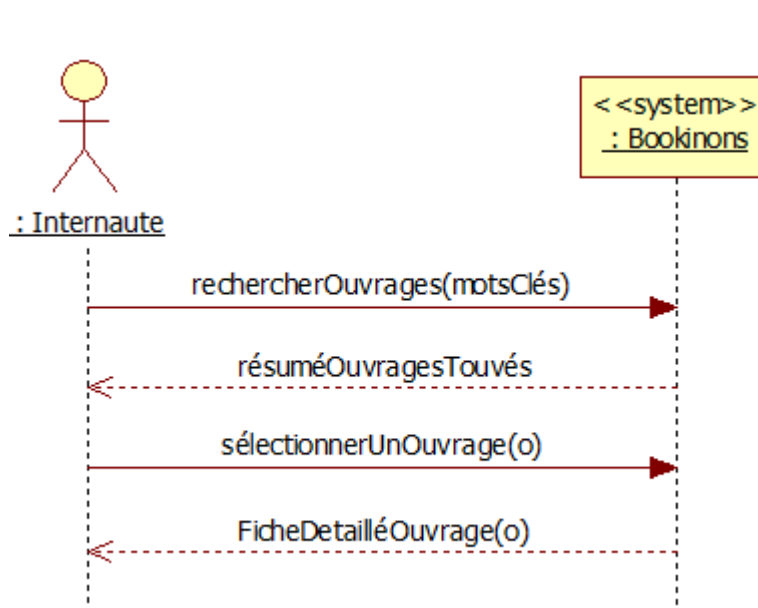


Le **système** informatique est vu comme une **boîte noire** car son comportement est **décrit de l'extérieur** (peut-être utilisé durant une phase d'analyse ... la boîte pourra ensuite être ouverte en phase de conception !)

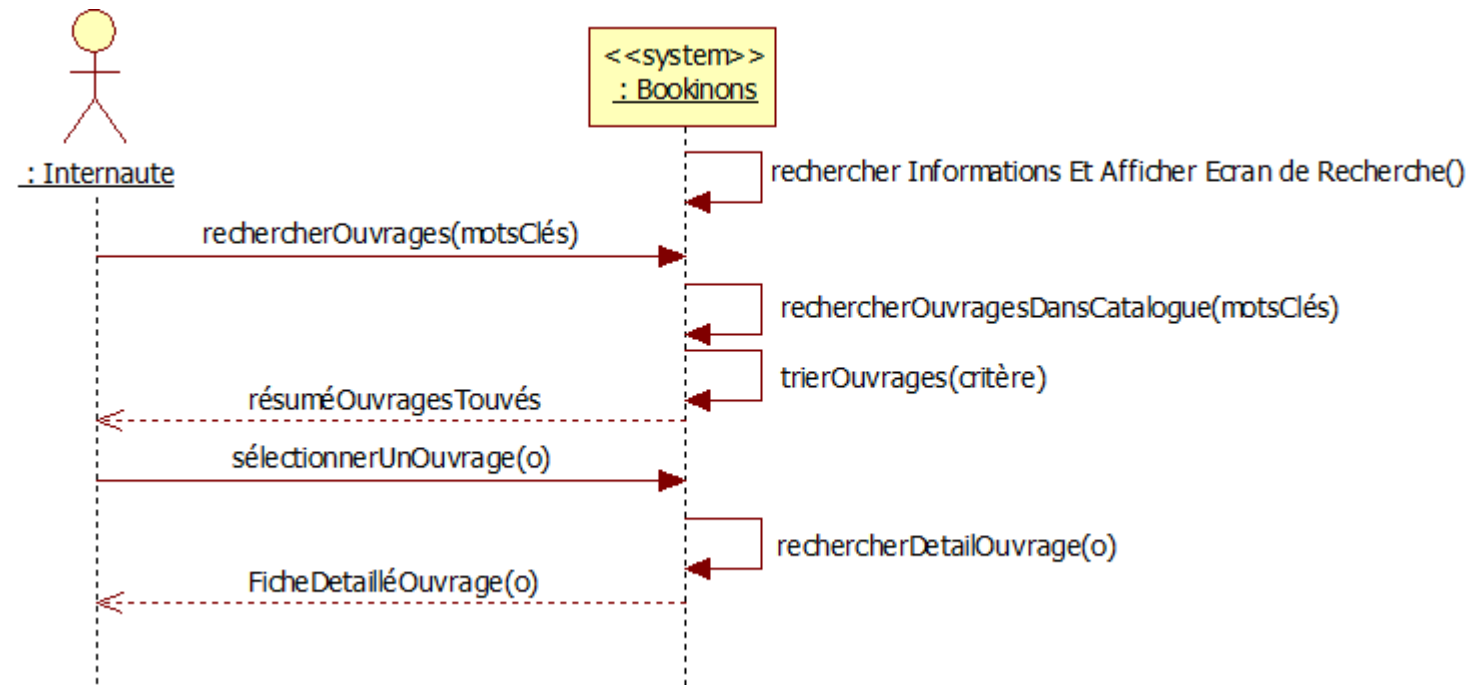
DSS Rechercher un ouvrage



Sans action interne



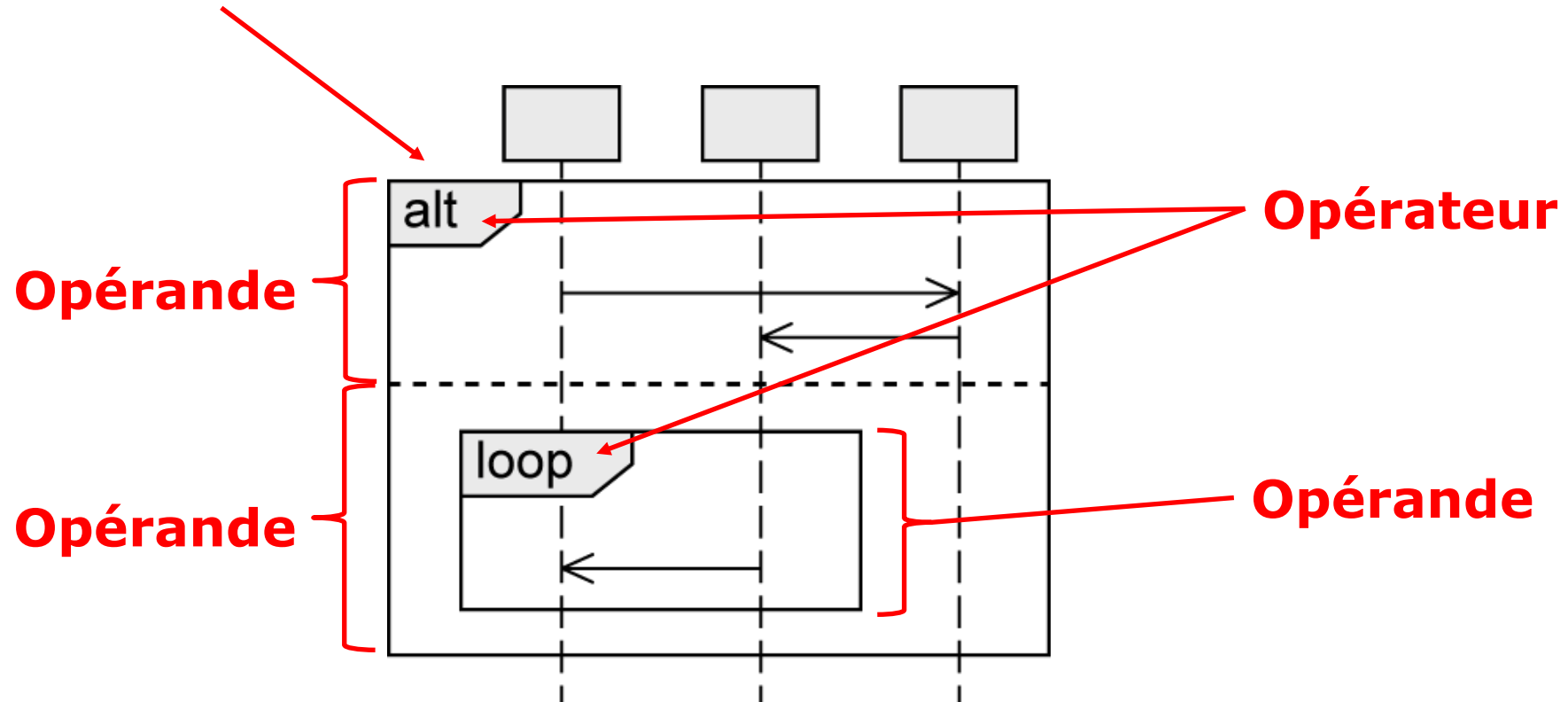
Avec action interne (message réflexif)



Fragment combiné

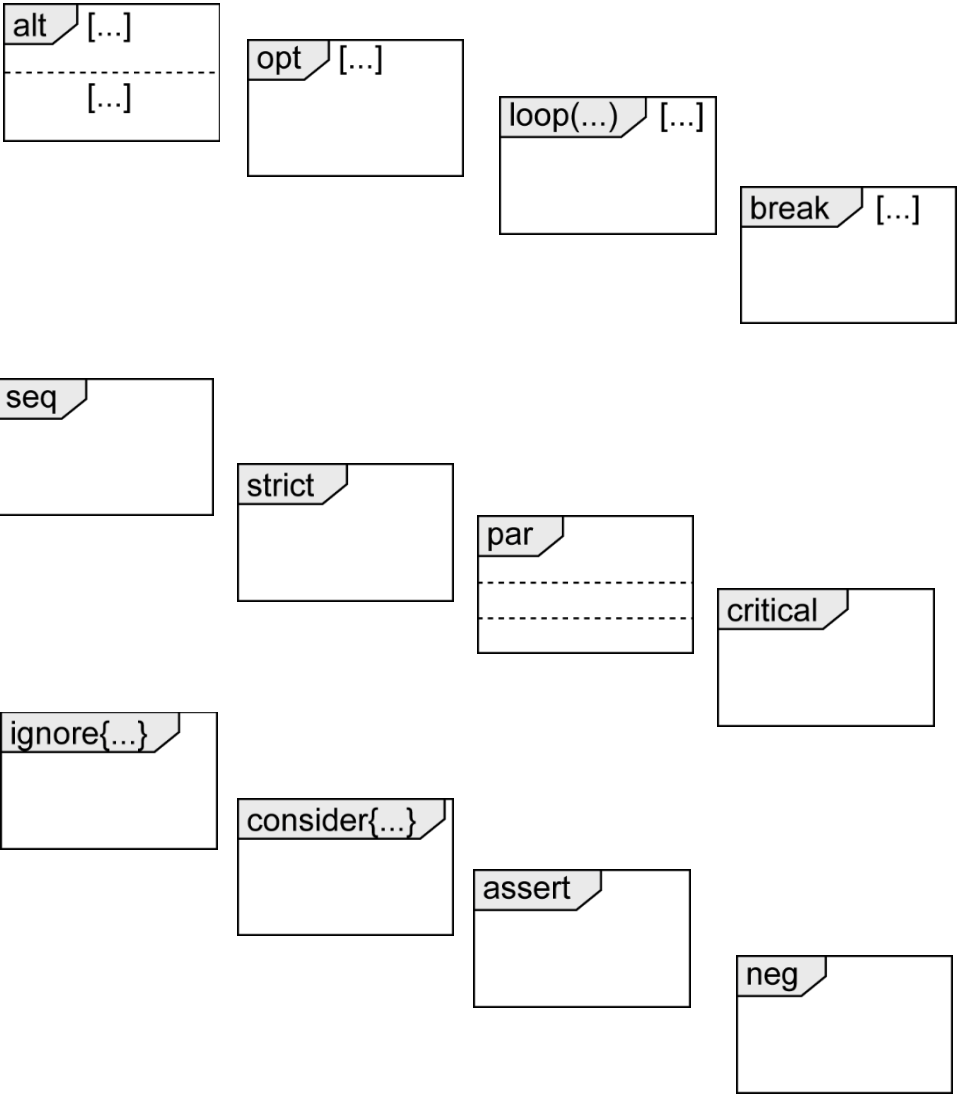
Fragment combiné : Présentation

Fragment combiné



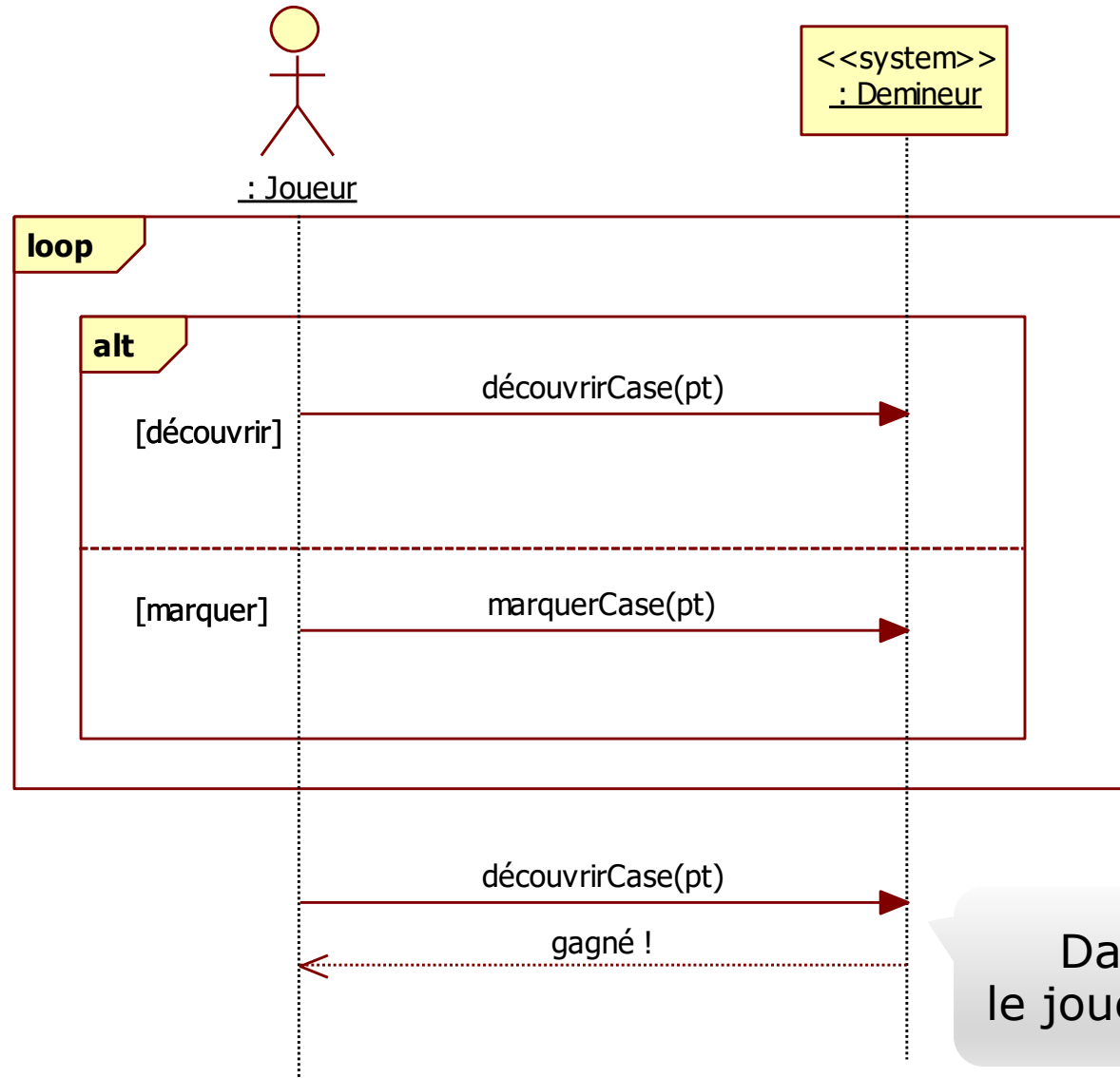
Liste des 12 opérateurs

	Opérateur	
Choix et boucles	alt	Alternative
	opt	Option
	loop	Boucle (répétition)
	break	Exception
Parallélisation et ordre d'envoi	seq	Weak sequencing
	strict	Strict sequencing
	par	Parallèle (execution concurrente)
	critical	Section critique
Contrôle envoi de message	ignore	Ignorer (messages non présents dans le fragment combiné)
	consider	Considérer (interactions à prendre en compte dans la séquence)
	assert	Assertion (le fragment combine est une assertion)
	neg	Negative (interaction invalide)



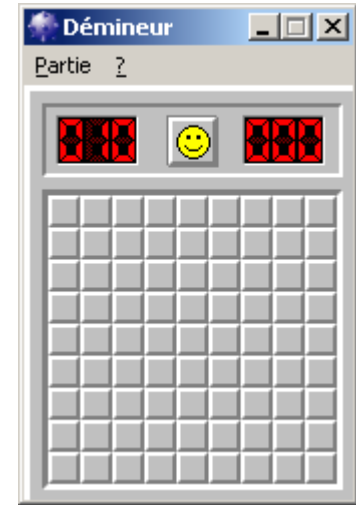
Fragments combinés : exemple

Diagramme séquence système simplifié
pour le flot de base du UC «Jouer une partie de démineur»



Le joueur passe son temps
à découvrir ou à marquer
des cases ...

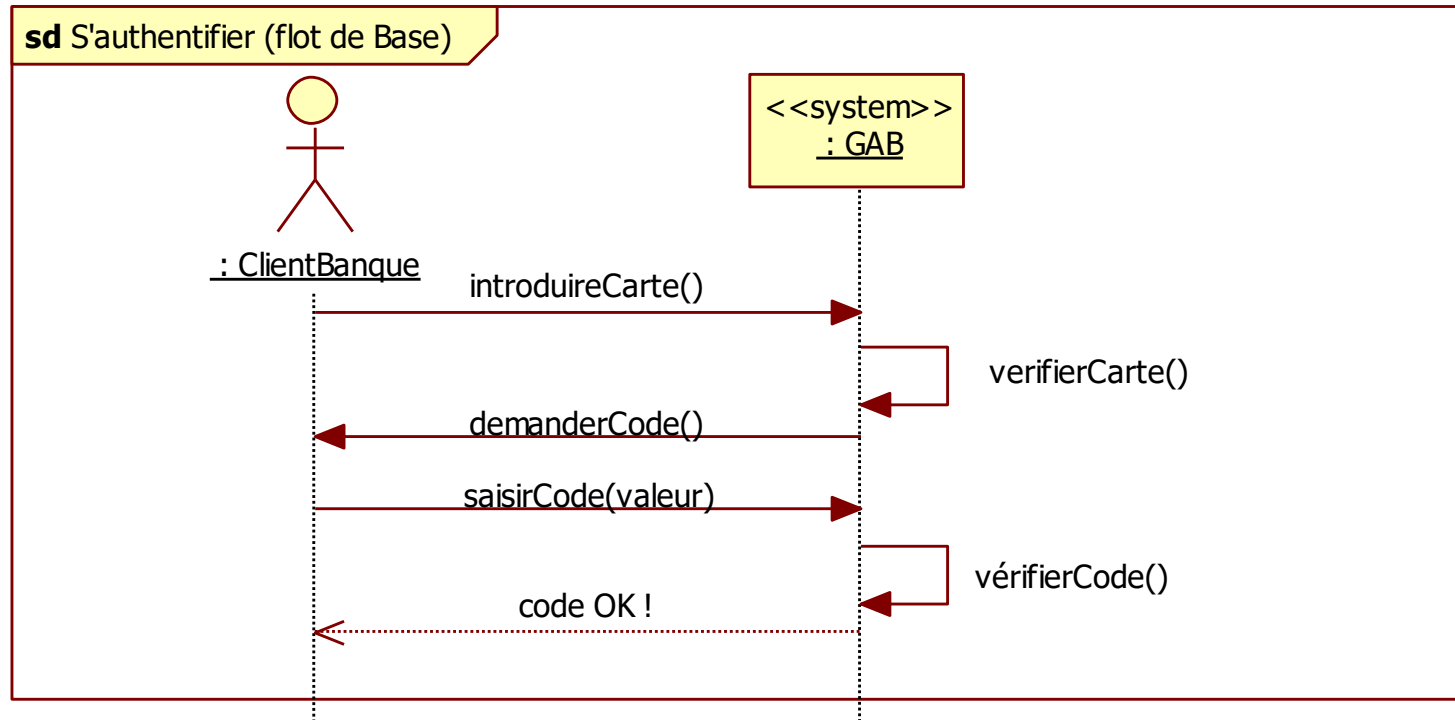
Dans le flot de base,
le joueur finit par gagner...



Cadre de diagramme & référence

Présentation de la notion de cadre de diagramme

Un diagramme peut être inclus dans un **cadre de diagramme** (opérateur **sd**)

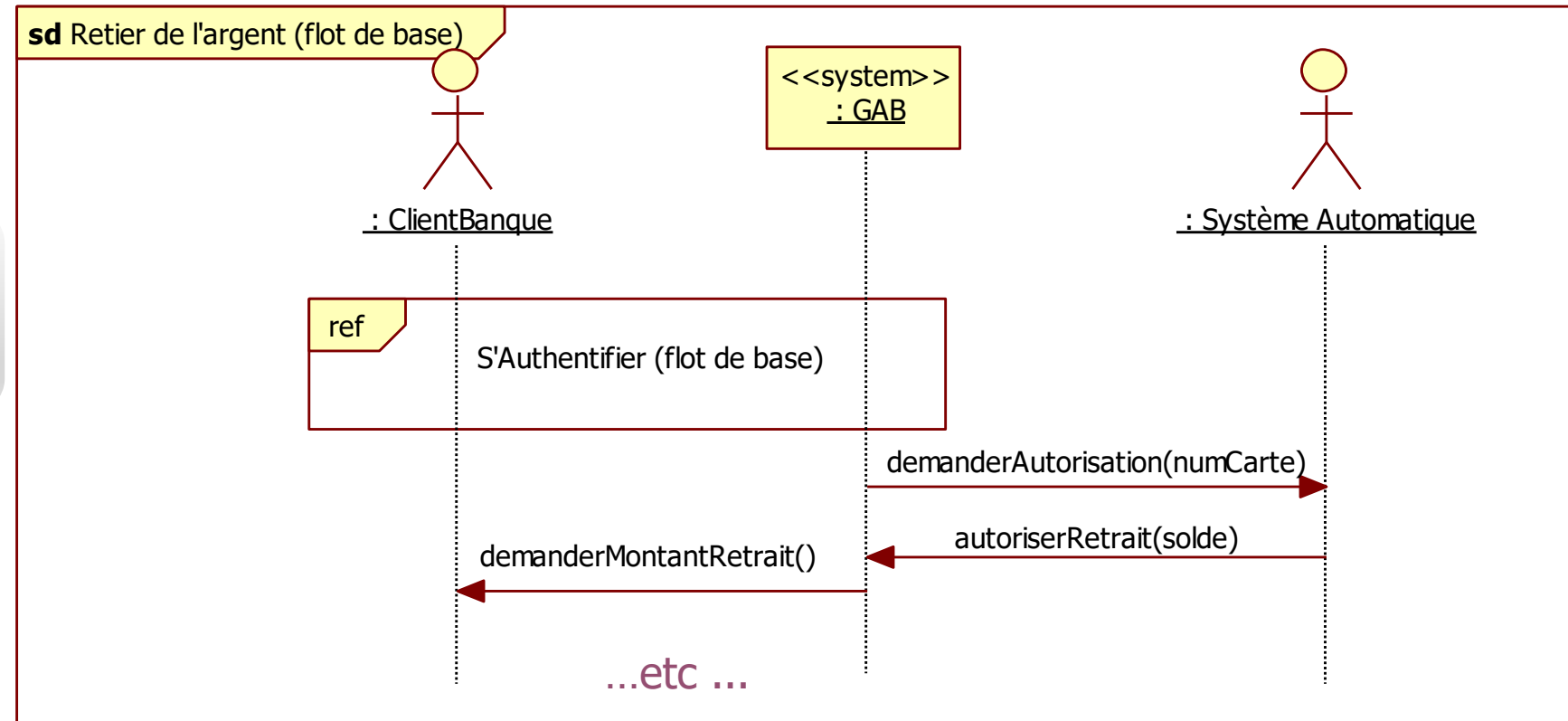


sd est le type du diagramme et signifie **sequence diagram**
Le cadre n'est pas obligatoire lorsque le contexte est clair.

Référencement d'interactions avec l'opérateur `ref`

Il est possible de factoriser des parties de comportement et d'alléger un diagramme de séquence en utilisant un cadre de diagramme ayant **ref** comme mot clé

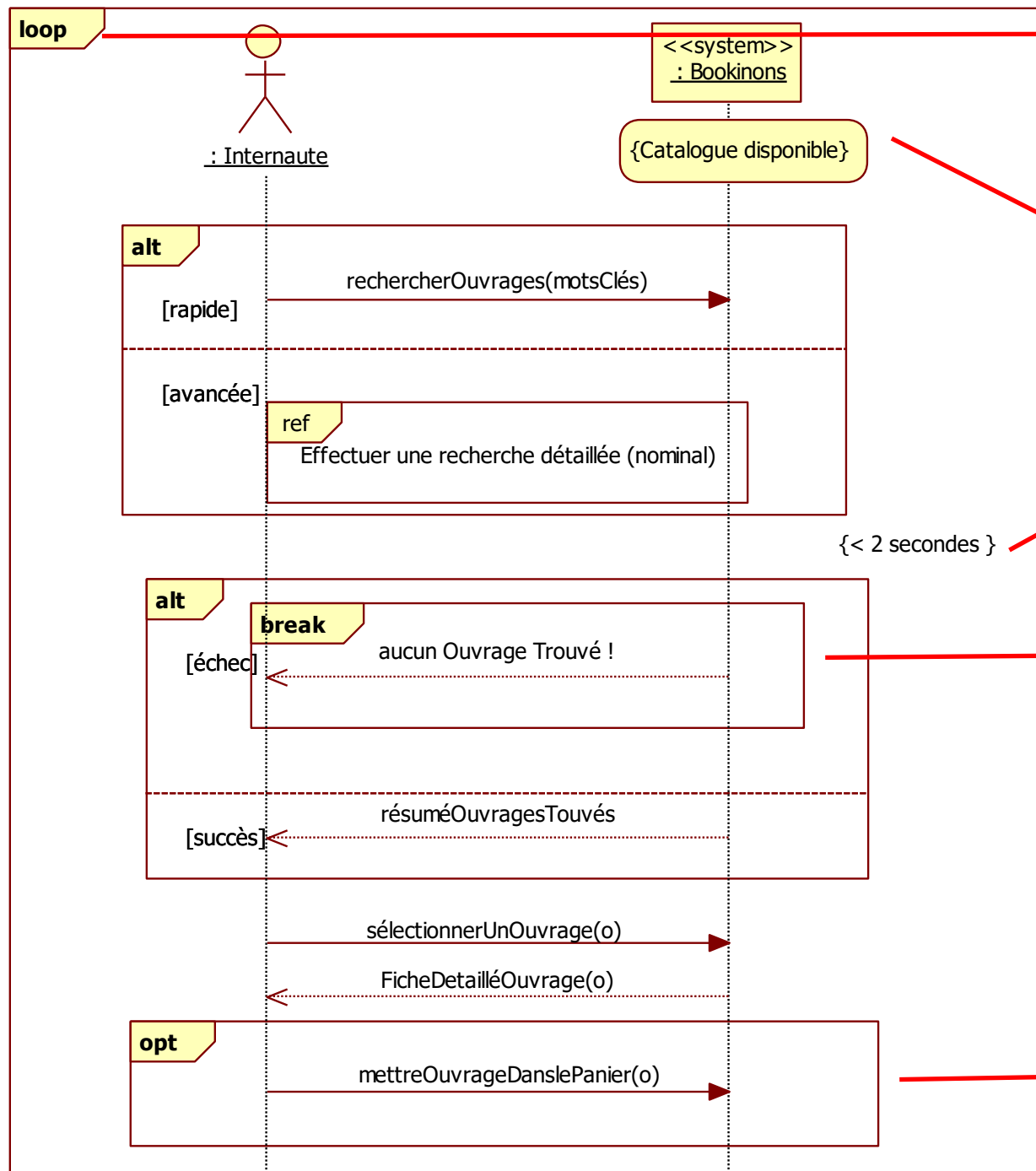
Le diagramme de séquence *Retirer de l'argent* fait **réf**érence au diagramme de séquence *S'authentifier*



Une **réf**érence (*interaction occurrence*) peut être vue comme un pointeur ou un raccourci vers un autre diagramme de séquence existant

Diagramme de séquence :

Exemple (notation UML 2.0)



La recherche d'ouvrage est répétable à volonté

Pré-condition rajoutée sous forme d'une **contrainte d'état**

Contrainte de temps

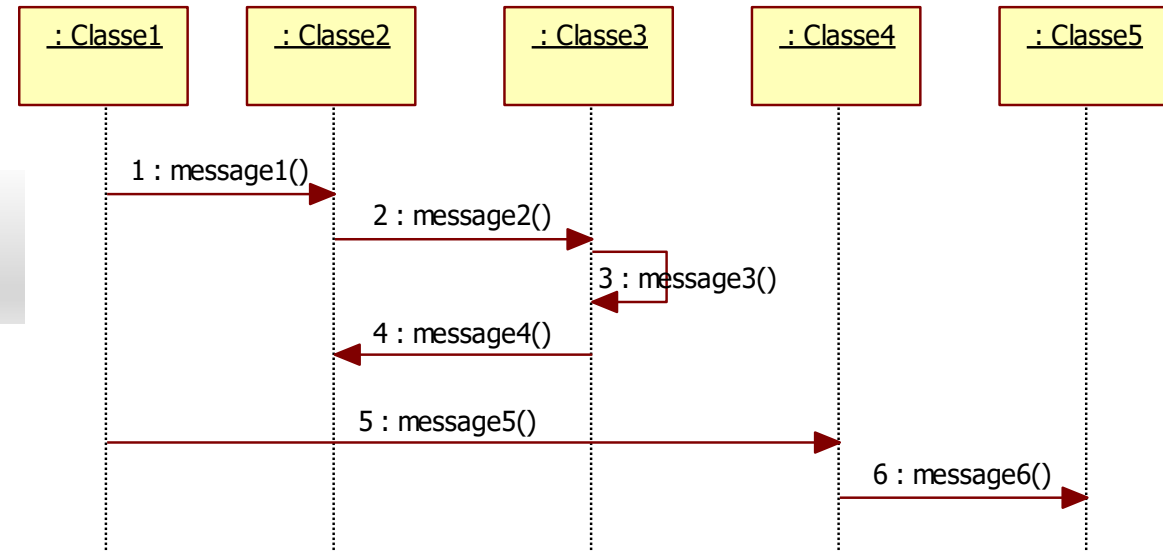
Exception : un break interrompt la séquence (*sélection et mise dans le panier ne seront pas possible si passage dans le break*)

On peut éventuellement (*optionnellement*) mettre l'ouvrage trouvé dans le panier

Diagramme de communication

Diagramme Séquence vs Diagramme Communication

Diagramme de séquence
représentation **temporelle**



*Les AGL permettent de
convertir automatiquement
un diagramme en un autre*

Diagramme de communication
représentation **spatiale** (structurelle)

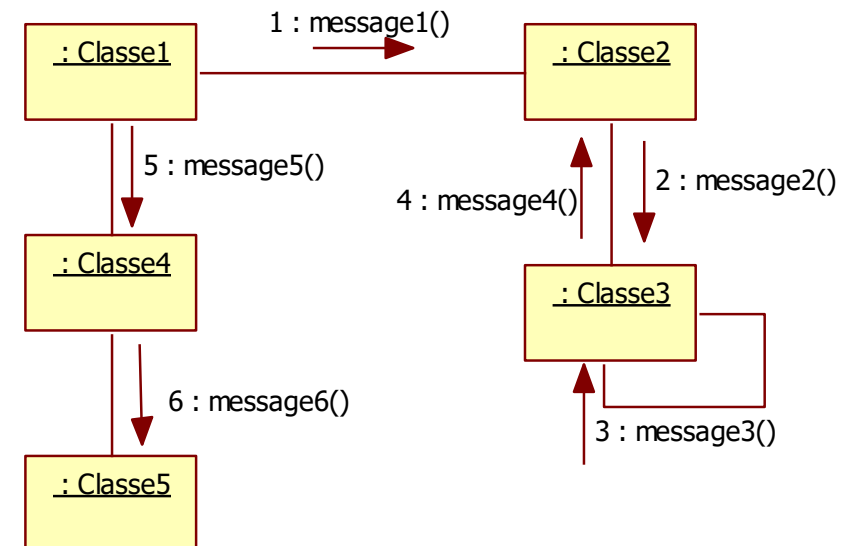
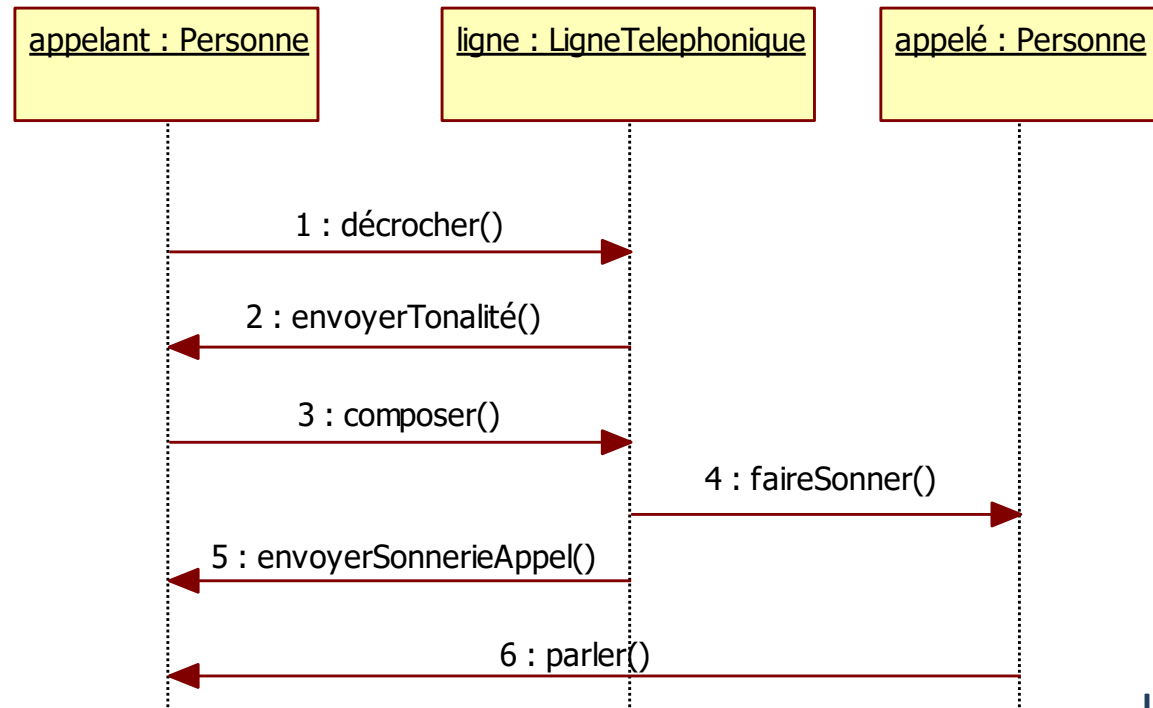
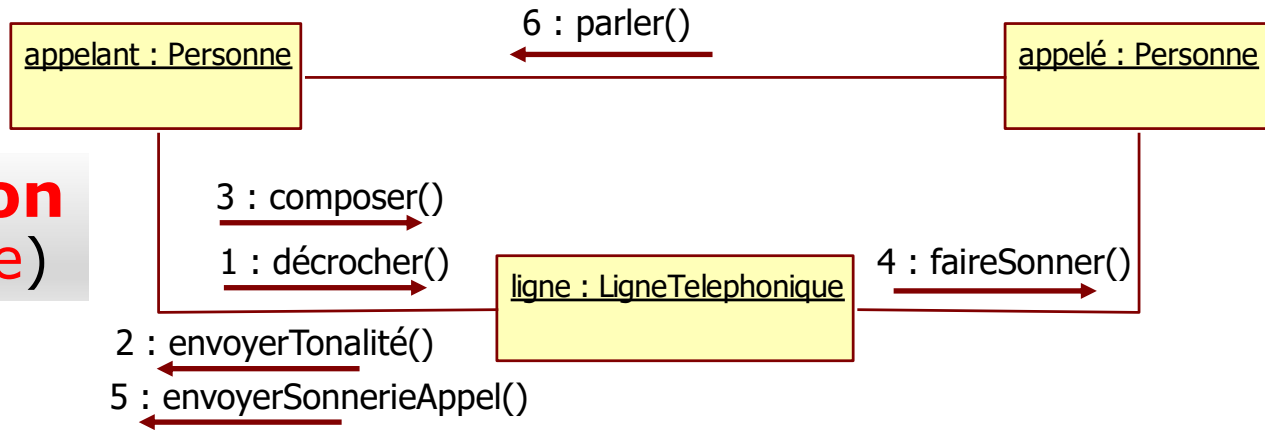


Illustration de la communication téléphonique

Diagramme de communication
représentation spatiale (structurelle)



Diagramme de séquence
représentation temporelle





Petit Exercice

Mise en application

Du diagramme de séquence



Tirer un missile depuis le vaisseau



Scénario possible pour la fonctionnalité
tirer un missile depuis le vaisseau d'un jeu Space Invaders

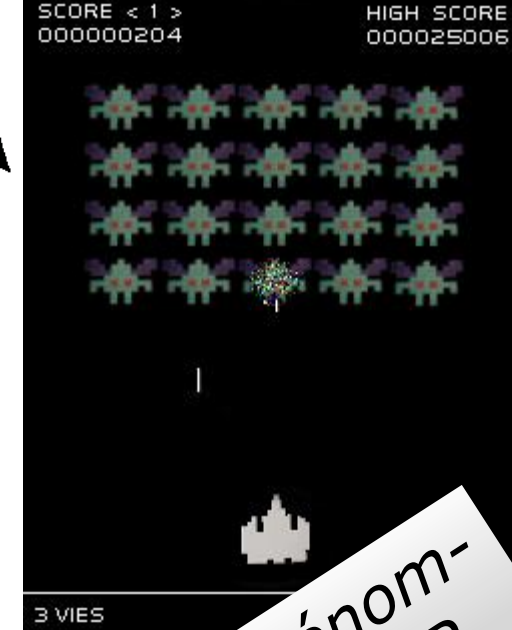
1. Le joueur demande au jeu Space Invaders de tirer un missile (en appuyant sur la touche ESPACE par exemple)

2. Le vaisseau tire un missile (en deux temps) :

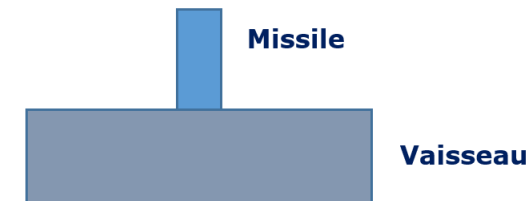
2.1 Le vaisseau calcule d'abord la position du missile de manière, lors du tir, à pouvoir positionner le missile en son milieu.

2.2 Le vaisseau émet ensuite le missile à la position souhaitée de manière à se retrouver dans la configuration ci-contre

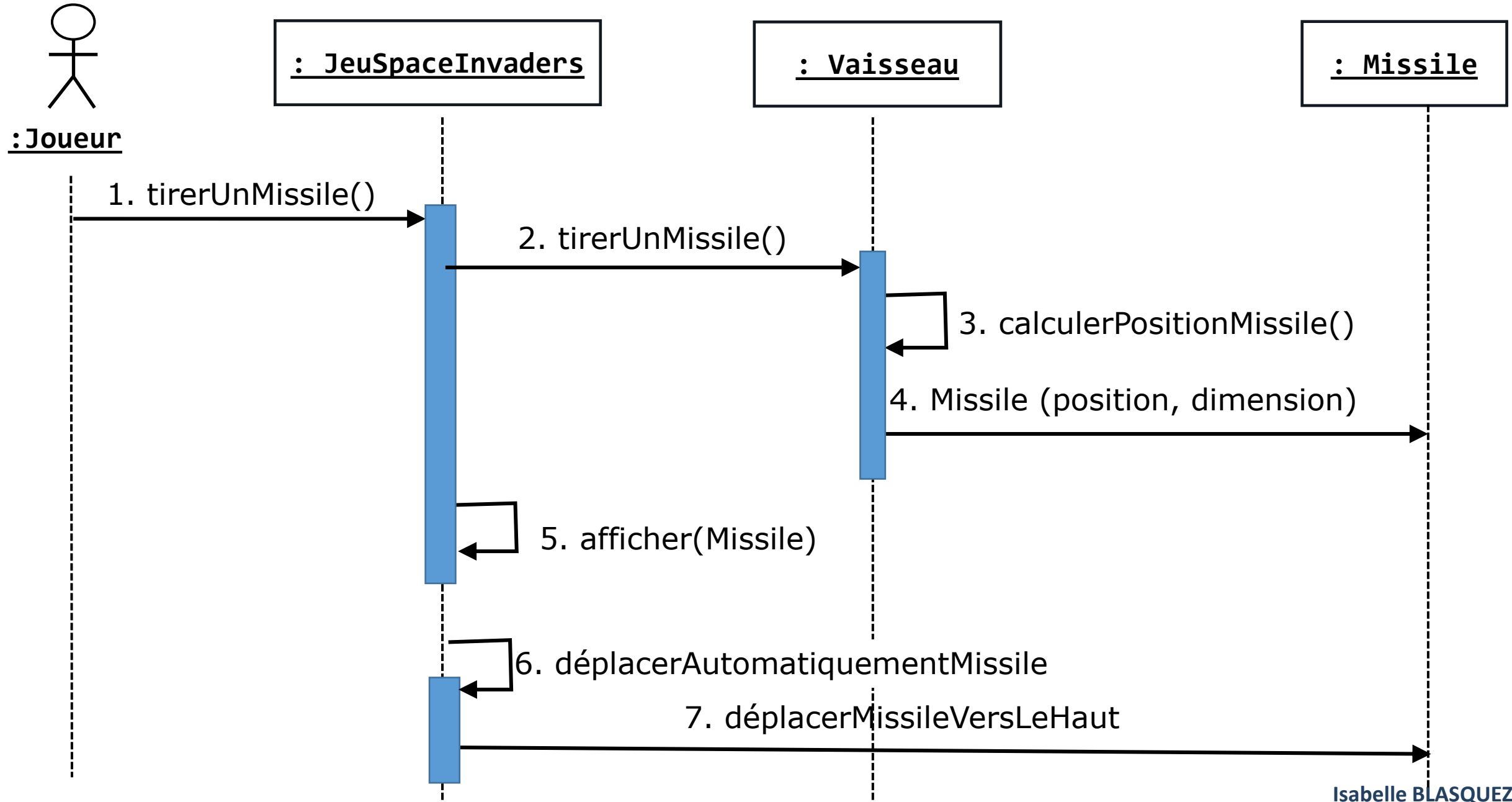
3. Le missile se déplace ensuite jusqu'en haut de l'espace jeu de manière autonome et automatique selon une trajectoire verticale.



Nom-Prénom-
Groupe TP

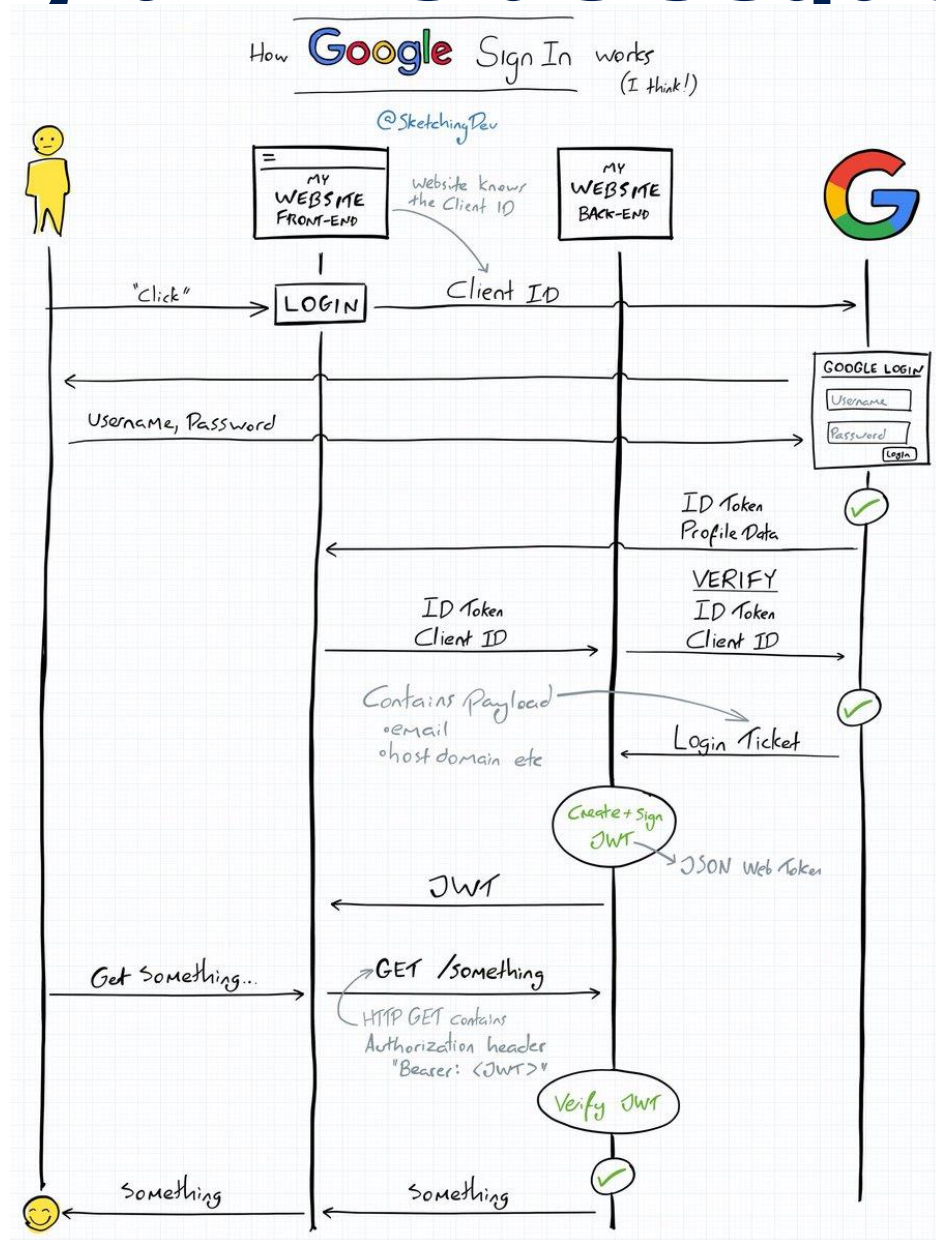


Exemple d'un diagramme de séquence possible ...



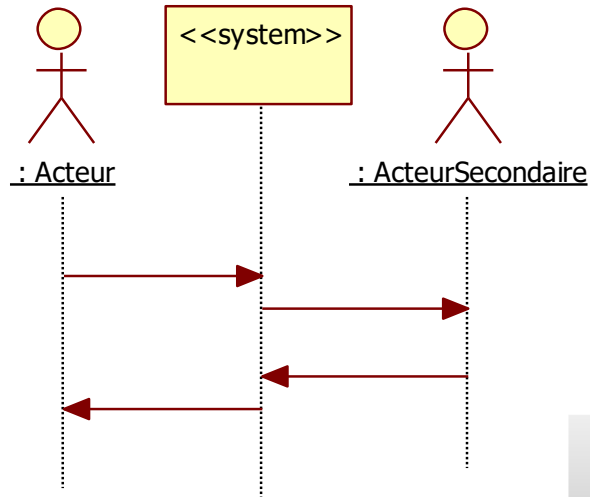
Utilisation des diagrammes de séquence

Diagramme de séquence en tant qu'outil de documentation ...

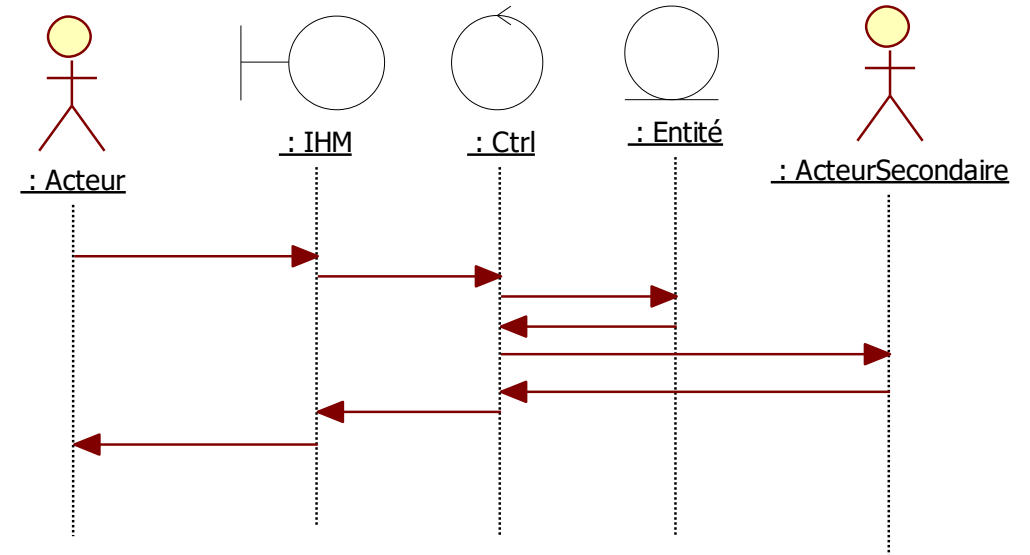


Expliquer
un traitement, un protocole...

Diagramme de séquence en tant qu'**outil de modélisation** pour les phases d'Analyse et de Conception



Ouvrons la boîte !



Analyse
système = boîte noire

Conception préliminaire :
Apparition de composants logiciels

Vue externe du système
(aspect fonctionnel : QUOI)

Vue interne du système
(aspect dynamique : QUAND)

**Analyse
du système
⇒ Cadrage des
exigences**

**Mais avant,
modélisons le périmètre fonctionnel
d'une application à l'aide
d'un diagramme de cas d'utilisation
(use case)**

Besoin du client



Développer un **MVP** du jeu de cartes de la bataille

MVP (Minimum Viable Product)

=> **Simplification** des besoins autour de la Valeur Métier prioritaire pour pouvoir tester rapidement la faisabilité et/ou l'intérêt/l'attrait d'une application

Ce qui est important ici est de pouvoir **Jouer** (Valeur métier prioritaire)

peu importe le multijoueur, les statistiques ... on va à l'essentiel !

=> **1 seul joueur joue contre le système**

Se familiariser avec le contexte métier

❑ **Se renseigner sur les règles métiers :**

<https://www.momes.net/jeux/jeux-interieur/regle-des-jeux-de-cartes/la-bataille-regles-du-jeu-842140>
[https://fr.wikipedia.org/wiki/Bataille_\(jeu\)](https://fr.wikipedia.org/wiki/Bataille_(jeu))

Analyse

❑ **Prendre le temps de bien comprendre les règles métiers et de s'imprégner des différentes étapes/activités du problème à résoudre**



1. Inscrire un nouveau joueur
2. Préparer un jeu de cartes de 52 cartes
3. Mélanger le jeu de 52 cartes
4. Distribuer le jeu de cartes de manière équitable et aléatoire entre les 2 joueurs
5. Tirer une carte du dessus du paquet
6. Jouer une bataille
7. Rechercher le joueur qui a la carte la plus forte
8. Ramasser les cartes
9. Déclarer le joueur vainqueur de la partie

Introduction aux cas d'utilisation (Use Case)

- ❑ Les cas d'utilisation décrivent *les comportements attendus d'un système du point de vue de ses utilisateurs*, sans avoir à préciser la façon dont ces comportements sont réalisés



... uniquement CE QUE le système
doit faire et surtout
pas comment il doit le faire....

Un système à modéliser

- ❑ Les **objectifs** des cas d'utilisations sont de :
 - Décrire le système *du point de vue de l'utilisateur*
 - Mettre en évidence les *services rendus par le système*
 - **Fixer le périmètre fonctionnel** entre le système et son environnement

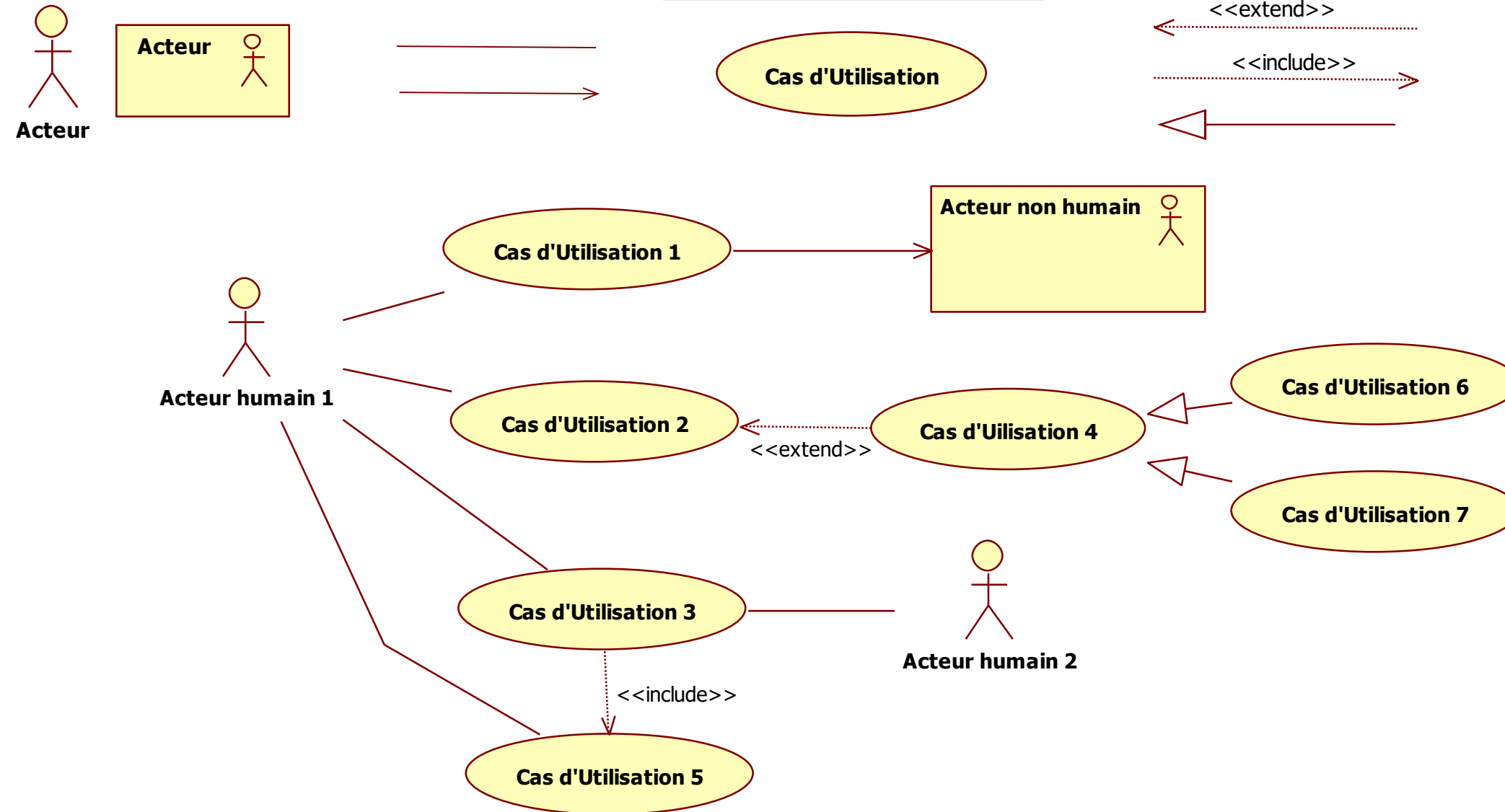
Les éléments du diagramme des cas d'utilisation

des **acteurs**...

reliés à ...

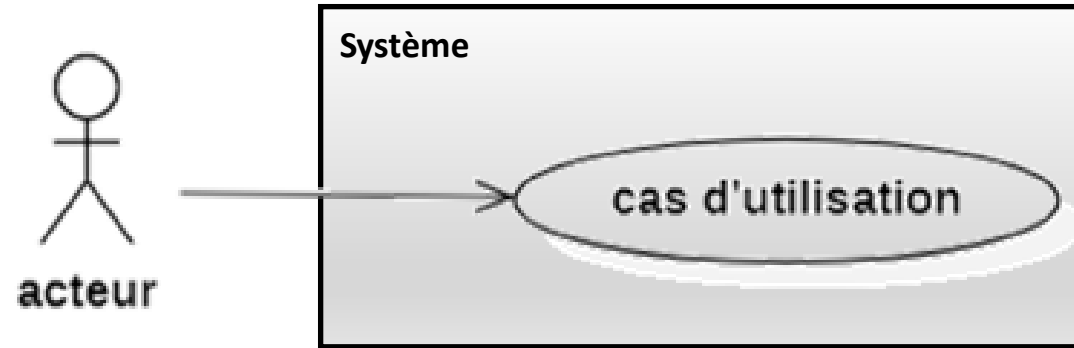
des **Cas d'Utilisation (UC)**

... pouvant être
reliés à d'autres UC



Introduction au diagramme des cas d'utilisation

Le diagramme des cas d'utilisation permet de visualiser et délimiter **le périmètre fonctionnel de l'application à développer**



- ⇒ **Les acteurs** vont permettre de recenser **tout utilisateur humain ou non qui va dialoguer avec le système** (ils seront à l'extérieur du système)
- ⇒ **Les cas d'utilisation** vont permettre de **recenser les besoins fonctionnels** (à l'intérieur du système)
- ⇒ **Un acteur** peut déclencher **plusieurs cas d'utilisations**
- ⇒ Mais les bonnes pratiques de conception préconisent **qu'un seul cas d'utilisation soit déclenché par un seul acteur**

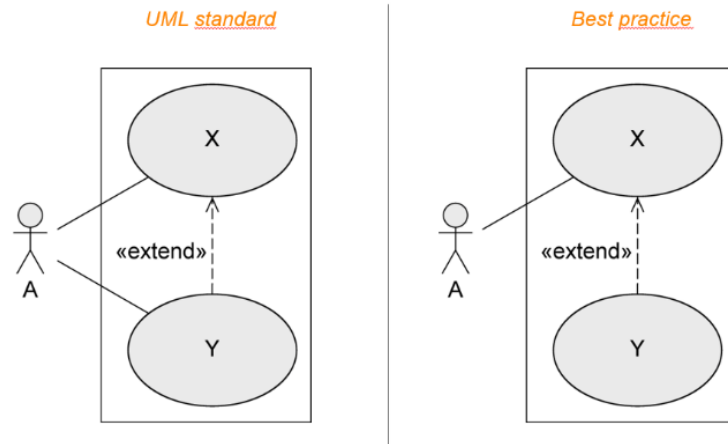
Relations « extends » et « extends » pour une granularité plus fine ...

Comme tout modèle, il est possible d'être **plus ou moins précis en terme de granularité** et de faire apparaître (si nécessaire) entre deux cas d'utilisation des relations supplémentaires :

- Soit une **relation d'extension (<<extends>>)** pour modéliser un caractère **facultatif**
- Soit une **relation d'inclusion (<<include>>)** pour un caractère **obligatoire**
- Soit une **relation d'héritage** que nous ne détaillerons pas ici.

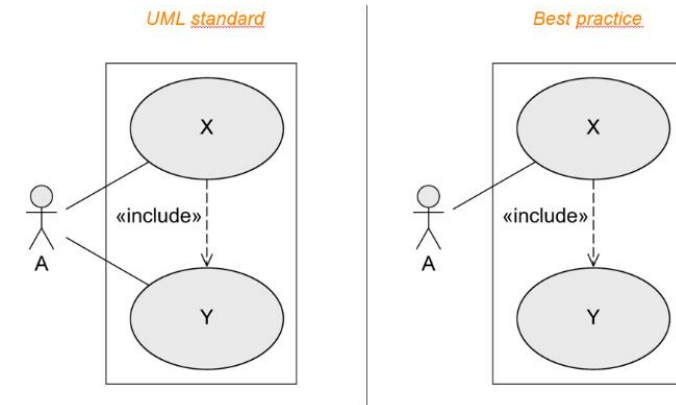
Best Practices

«extend»



Best Practices

«include»



Dans le cas d'une **extension**, le cas d'utilisation X décide si le cas d'utilisation Y sera exécuté (ou pas) **selon les besoins de l'acteur**

Dans le cas d'une **inclusion**, le cas d'utilisation X a **absolument** besoin du cas d'utilisation Y pour être exécuté.

De l'immersion métier au périmètre fonctionnel

L'expérience utilisateur vécue via l'immersion a permis de lister pas à pas toutes les « étapes » à développer dans notre logiciel pour mener à bien une partie du jeu de la bataille.

1. Inscrire un nouveau joueur
2. Préparer un jeu de cartes de 52 cartes
3. Mélanger le jeu de 52 cartes
4. Distribuer le jeu de cartes de manière équitable et aléatoire entre les 2 joueurs
5. Tirer une carte du dessus du paquet
6. Jouer une bataille
7. Rechercher le joueur qui a la carte la plus forte
8. Ramasser les cartes
9. Déclarer le joueur vainqueur de la partie

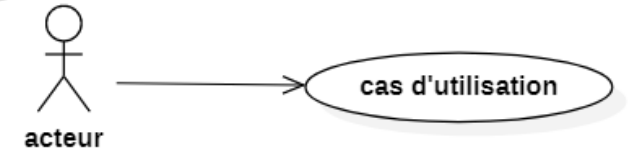
Analyse

Au travers de ces « étapes », vous venez d'identifier un **certain nombre de fonctionnalités** à développer (ou **cas d'utilisation** ou de **cas d'usage** ou **Use Case**)

Un premier cas d'utilisation ..

❑ Pour chaque étape identifiée précédemment, modélisez le couple acteur/cas d'utilisation en adoptant la norme suivante

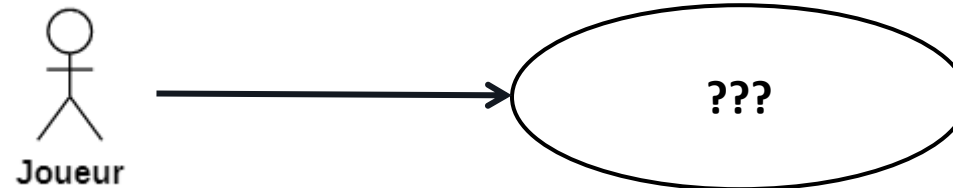
Analyse



⇒ À noter que parfois, le système peut aussi être un acteur

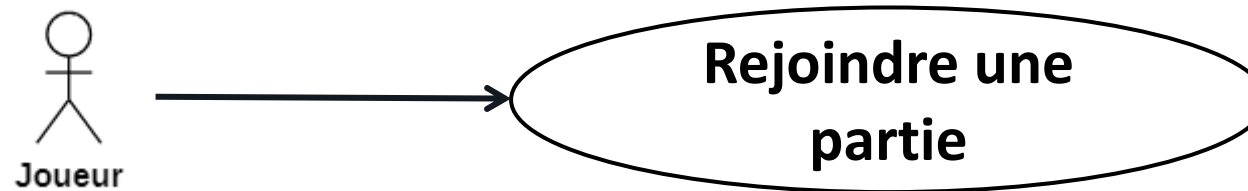
⇒ L'acteur est celui qui « déclenche » la fonctionnalité (UC) de notre application

1. Inscrire un nouveau joueur



⇒ **Par convention**, un cas d'utilisation se nomme avec un **verbe suivi d'un complément** qui doit refléter **le point de vue de l'utilisateur** (en principe au *singulier*)

(c-a-d une fonctionnalité vue de l'extérieur du système qui apporte un **nouveau comportement**)



⇒ Utilisez bien sûr la **terminologie métier** : c'est le moment de commencer un **glossaire** !!!

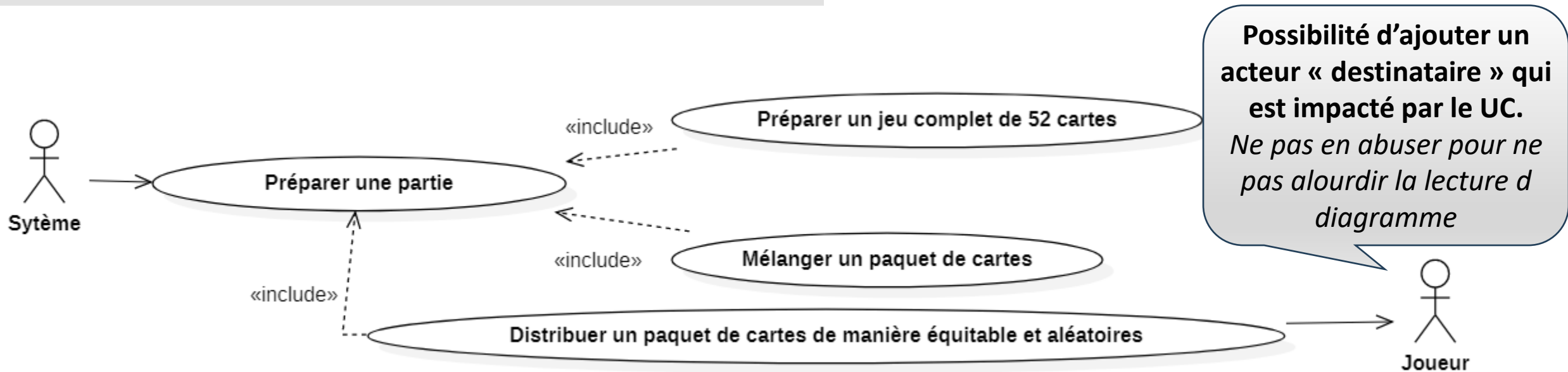
Plus on itère (travail) sur l'analyse ⇒ plus on affine le modèle

en terme de **granularité** et de **termes métier**

Analyse

1. ...
2. Préparer un jeu de cartes de 52 cartes
3. Mélanger le jeu de 52 cartes
4. Distribuer le jeu de cartes de manière équitable et aléatoire entre les 2 joueurs
5.

3 sous-étapes qui permettent
de préparer une partie



Itérer sur l'analyse (re-travailler sur plusieurs séances) pour mieux appréhender le problème

Analyse

....

5. Tirer une carte du dessus du paquet
6. Jouer une bataille
7. Rechercher le joueur qui a la carte la plus forte
8. Ramasser les cartes
9. Déclarer le joueur vainqueur de la partie

Approfondir connaissances sur le besoin métier

⇒ Identifier et définir **des termes métiers**

⇒ Meilleure identification/détail des étapes (**workflow**)
(meilleure définition/compréhension du **parcours utilisateur**)

....

5. Jouer une **main**

⇒ ~~Tirer une carte du dessus du paquet~~ **piocher** une carte sur le dessus du **tas** (paquet de cartes)

⇒ ~~Jouer une bataille~~ éventuellement organiser une **bataille**

⇒ ~~Rechercher le joueur qui a la carte la plus forte~~ Désigner le joueur qui **peut lever la main** (Rechercher le joueur qui a la carte la plus forte) => finalement ça c'est une action de **jouer une main** 😊

~~6. Ramasser les cartes~~

6. S'emparer de la **main** (effectuer une **levée**)

7. **Plier la levée** (Ranger la levée au-dessous du tas)

~~. Déclarer le joueur vainqueur de la partie~~ Terminer la partie

⇒ soit quitter volontairement de la partie

⇒ soit déclarer un vainqueur de la partie

Après quelques itérations ...

En pratique plus facile/lisible de faire les itérations directement en visuel en modifiant/incrémentant un diagramme de UC plutôt que du texte 😊
(diag UC. à retrouver en annexe)

Exemple d'utilisation de la relation «extends»

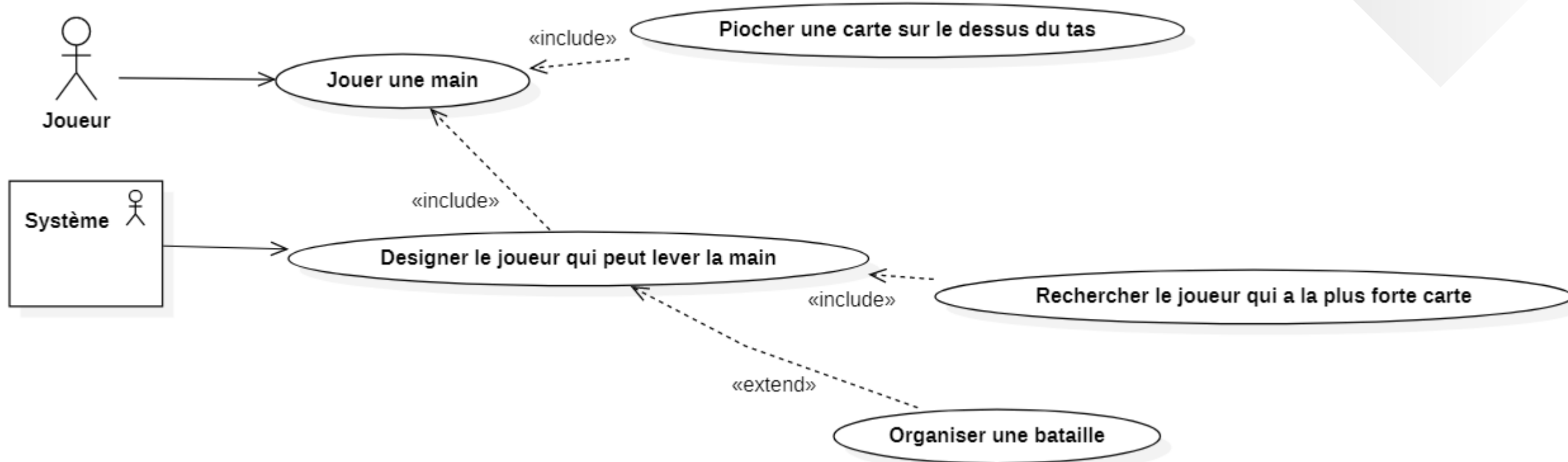
...

5. Jouer une main

- piocher une carte sur le dessus du tas (paquet de cartes)
- désigner le joueur qui peut lever la main
- éventuellement organiser une bataille

...

Un exemple possible de modélisation à l'aide du diagramme UC



Analyse

**Puis analysons
chaque fonctionnalité
de manière détaillée**

Analyse détaillée de la première fonctionnalité



❑ Décrire étape par étape le scénario idéal du UC

1. Le système demande le nom du joueur
2. Le joueur saisit son nom
3. Le système enregistre un nouveau joueur dans la partie avec le nom saisi
4. Le système affiche un message de bienvenue avec le nom du joueur

❑ Réaliser des Maquette(s) À faire valider par votre client

**Analyse détaillée
d'une fonctionnalité (UC)**

❑ Prendre le temps d'identifier les règles métiers particulières

pour que votre fonctionnalité se déroule selon les besoins du client et sans bug 😊

- Doit-on limiter le nombre de caractère pour la saisie d'un nom ?
- Permet-on que le nom saisi soit uniquement composé d'espaces ?
- Formate-t-on tous les noms de la même manière : première lettre en majuscule et le reste en minuscule ?
- Permet-on uniquement des lettres ? les chiffres sont-ils acceptés dans le nom ?
- ... etc

**Après la phase d'analyse,
vient la phase de conception !
(où nous allons retrouver
les **diagrammes de séquences**
et **diagrammes de classes**)**

Contraintes de **conception** :



Développer le jeu de cartes de la bataille
avec un **pàttern MVC**

Le **pattern MVC** (**M**odèle **V**ue **C**ontrôleur) est le plus populaire et le plus couramment utilisé dans le développement d'applications proposant une interface graphique.

Il est important lors de la mise en place d'une architecture de ne pas « réinventer la roue », mais de s'appuyer sur des **patterns** (*modèles de programmation*) robustes et simples qui ont fait leur preuve.

 **Contrainte *bonnes pratiques d'implémentation***
pour ce projet 

**Utilisation d'un pattern MVC pour
favoriser le découplage de l'application**

Utilisation
des stéréotypes de Jacobson
pour la modélisation
d'un scénario via le pattern MVC
(Modèle Vue Contrôleur)

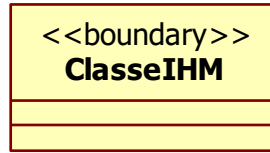
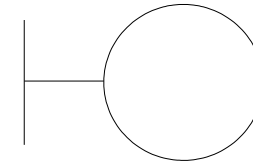
Stéréotypes de Jacobson

Dans le processus de conception

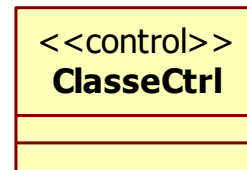
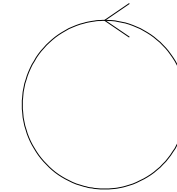
(diagramme de séquence)

Typologie des stéréotypes de Jacobson

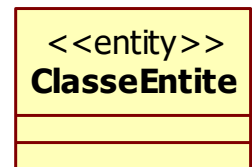
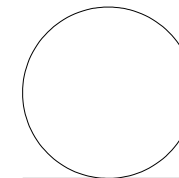
Les dialogues (<<boundary>>) sont les classes qui permettent l'interaction entre l'application et ses utilisateurs. Il s'agit des écrans proposés à l'utilisateur (**V**ue)



Les contrôles (<<control>> ou contrôleur) sont les classes qui contiennent la dynamique de l'application. (**C**ontroleur)

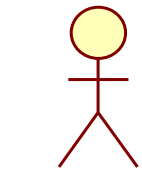


Les **entités** (<<entity>>) sont les classes qui représentent les concepts métiers. Elles sont souvent persistantes. (**M**étier/Modèle)



Mise en place du diagramme de séquence (pattern MVC)

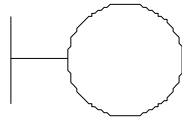
L'**acteur**
principal
du scénario



: User

L'acteur enverra
des messages
mais n'en recevra
jamais

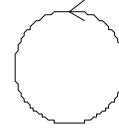
Vue permet à
l'acteur d'interagir
avec le système
via une IHM



: GameView

<<boundary>>
Par convention :
nom du UC
suivi de View

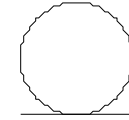
Contrôleur :
contrôle et
orchestre
l'application



: GameController

<<control>>
Par convention :
nom du UC
suivi de Controller

Métier : classes métiers que
le diagramme de séquence
va permettre de découvrir ...

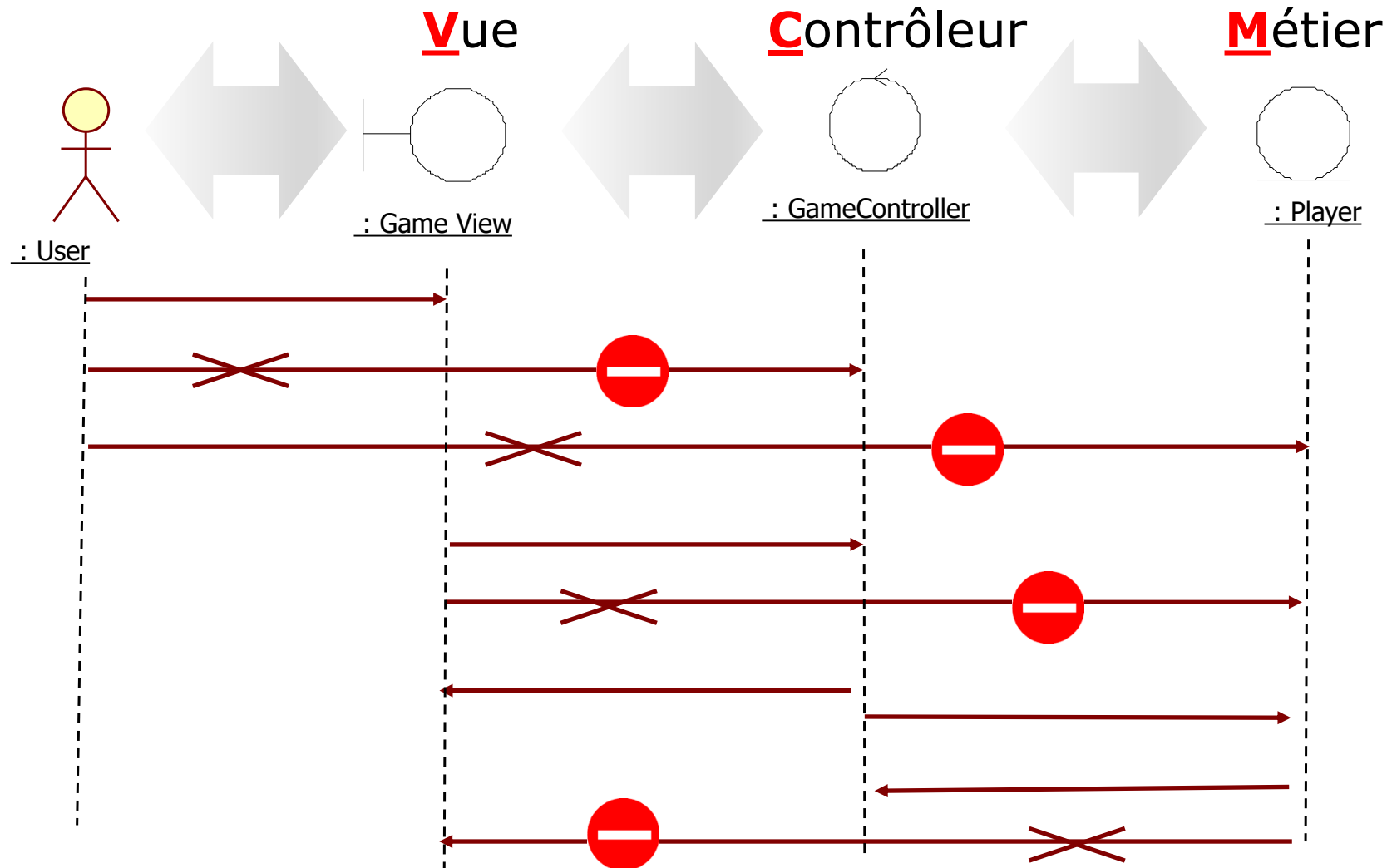


: Player

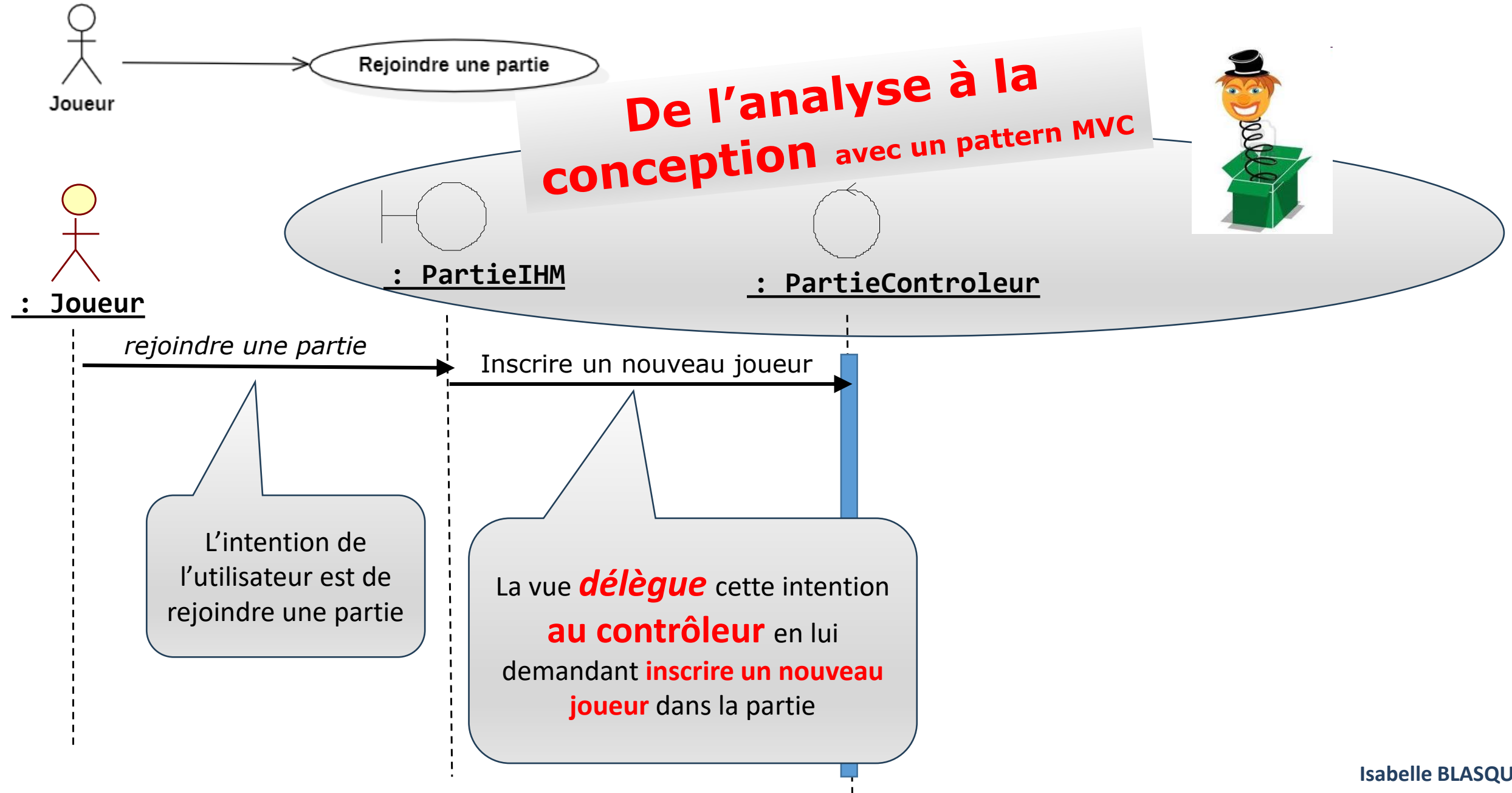
<<entity>>
(... mais aussi des
<<interface>>
et autres <<control>>)

Rappel des règles de construction d'un pattern d'analyse MVC

A retenir : Les messages ne devront pas "sauter" de couche !!!



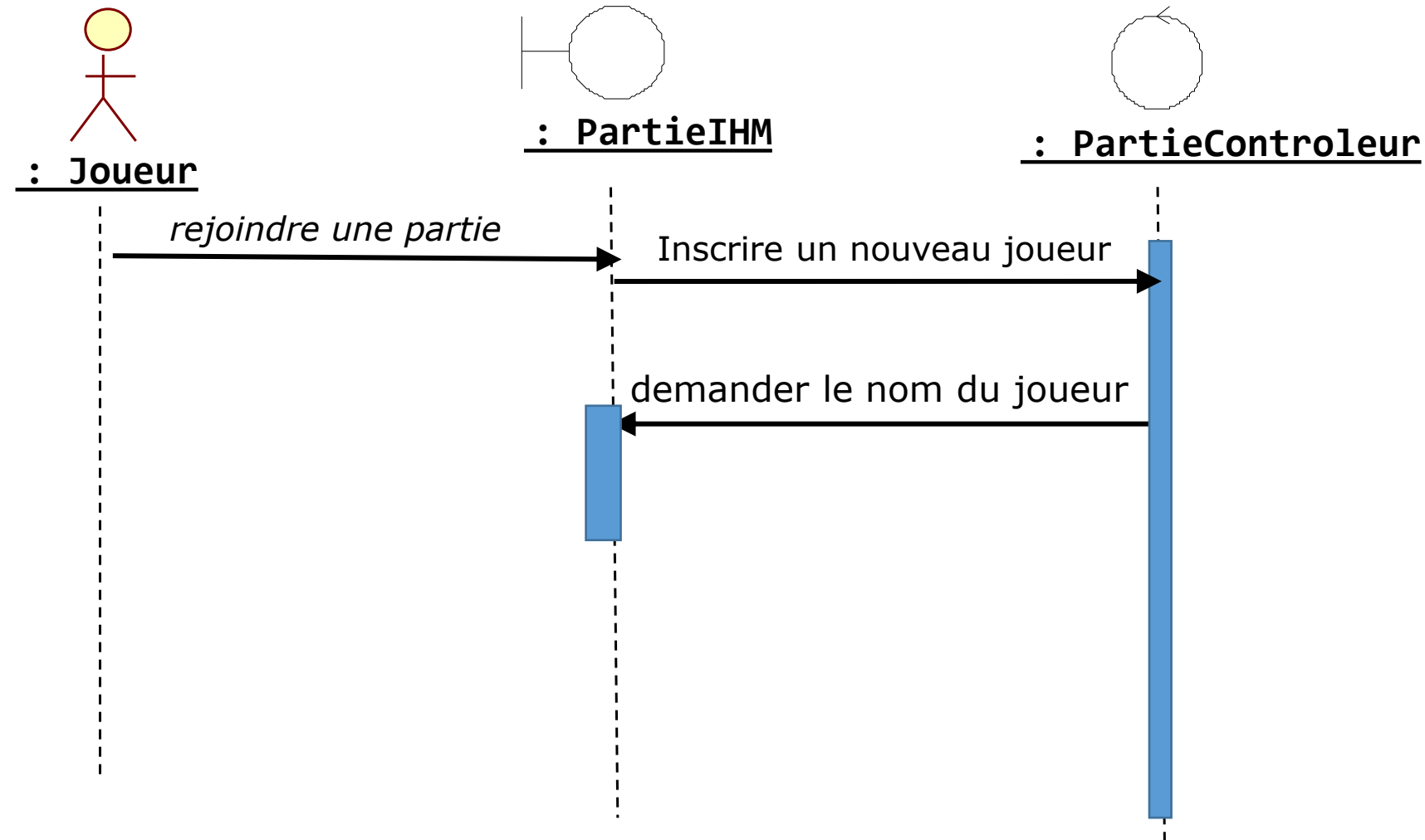
Zoom sur la fonctionnalité rejoindre une partie (1/5)



Zoom sur la fonctionnalité rejoindre une partie (2/5)

❑ Scénario à modéliser :

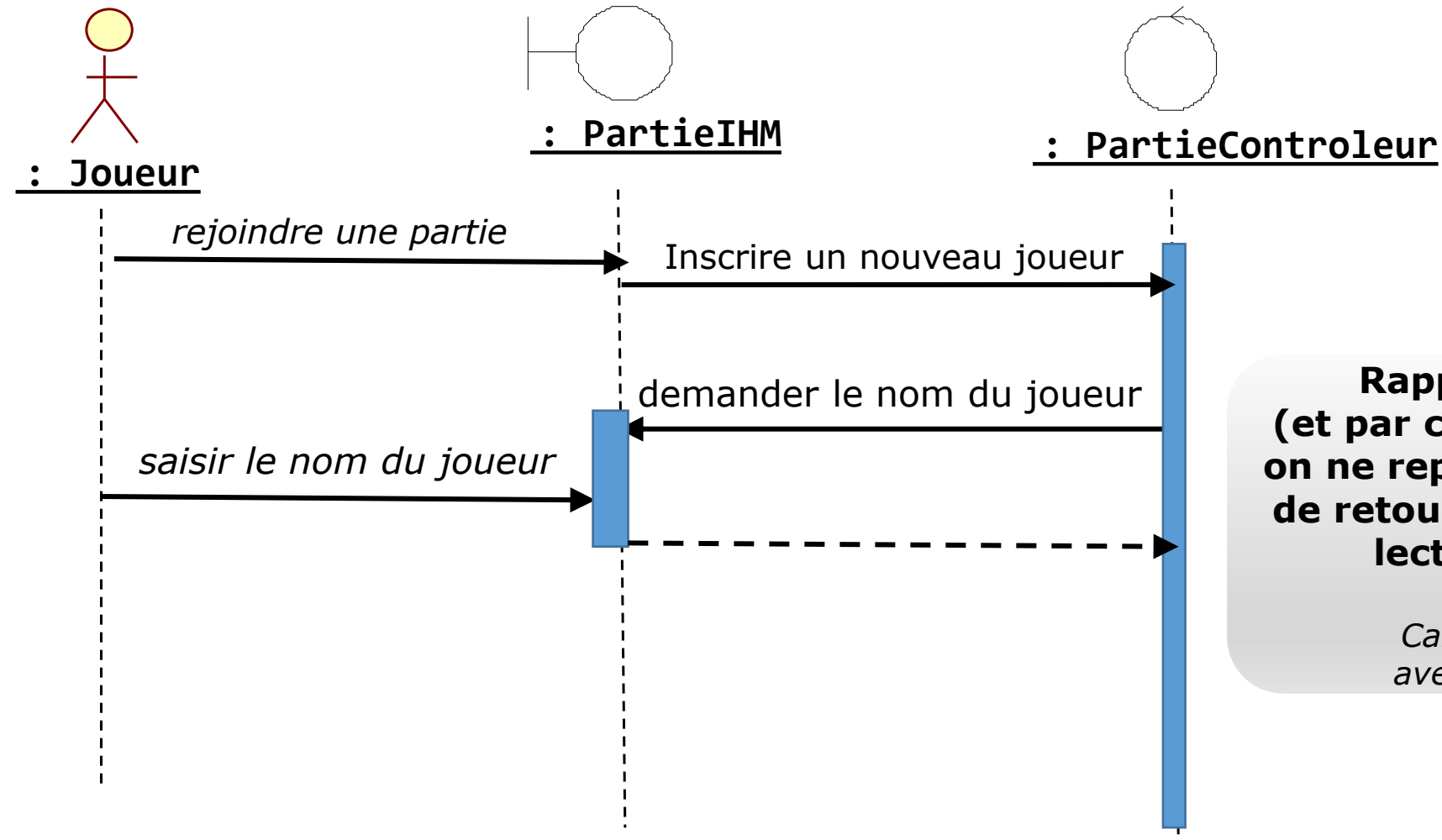
1. Le système demande le nom du joueur
2. Le joueur saisit son nom
3. Le système enregistre un nouveau joueur dans la partie avec le nom saisi
4. Le système affiche un message de bienvenue avec le nom du joueur



Zoom sur la fonctionnalité rejoindre une partie (3/5)

❑ Scénario à modéliser :

1. Le système demande le nom du joueur
2. Le joueur saisit son nom
3. Le système enregistre un nouveau joueur dans la partie avec le nom saisi
4. Le système affiche un message de bienvenue avec le nom du joueur



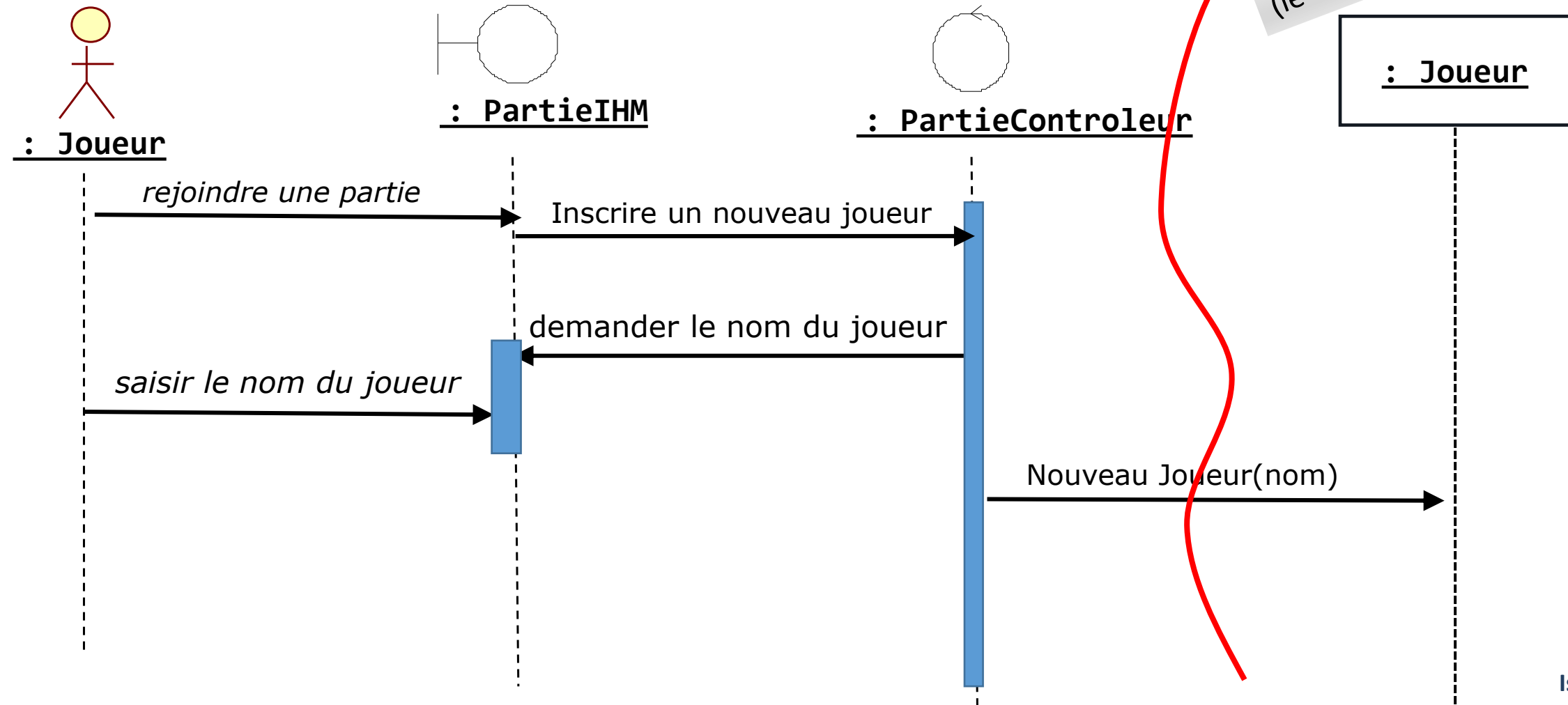
**Rappel : Habituellement
(et par convention pour la suite)
on ne représente pas de message
de retour pour ne pas alourdir la
lecture du diagramme.**

*Car le retour est implicite
avec un appel synchrone !*

Zoom sur la fonctionnalité rejoindre une partie (4/5)

❑ Scénario à modéliser :

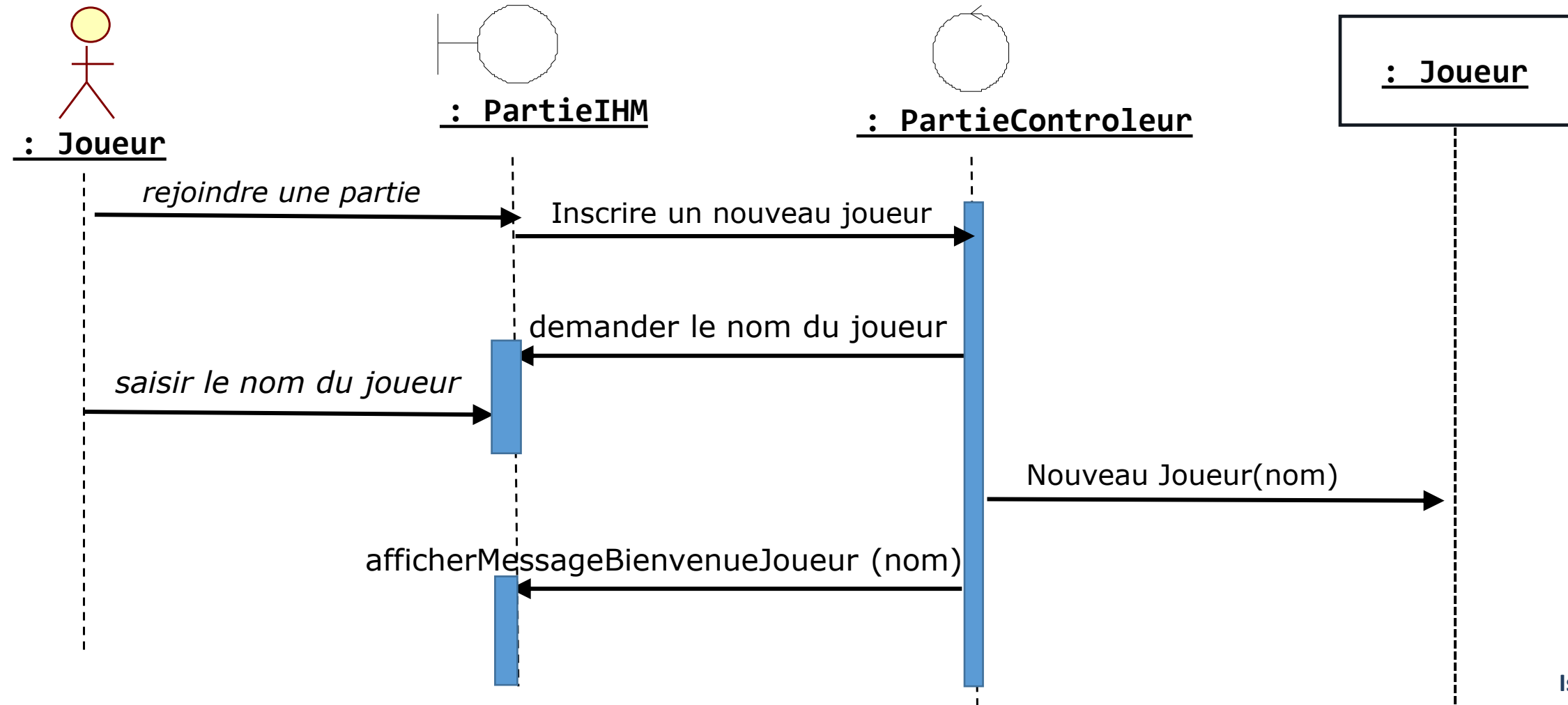
1. Le système demande le nom du joueur
2. Le joueur saisit son nom
3. Le système enregistre un nouveau joueur dans la partie avec le nom saisi
4. Le système affiche un message de bienvenue avec le nom du joueur



Zoom sur la fonctionnalité rejoindre une partie (5/5)

❑ Scénario à modéliser :

1. Le système demande le nom du joueur
2. Le joueur saisit son nom
3. Le système enregistre un nouveau joueur dans la partie avec le nom saisi
4. Le système affiche un message de bienvenue avec le nom du joueur



Intérêt de construire un diagramme de séquence pendant la phase de conception

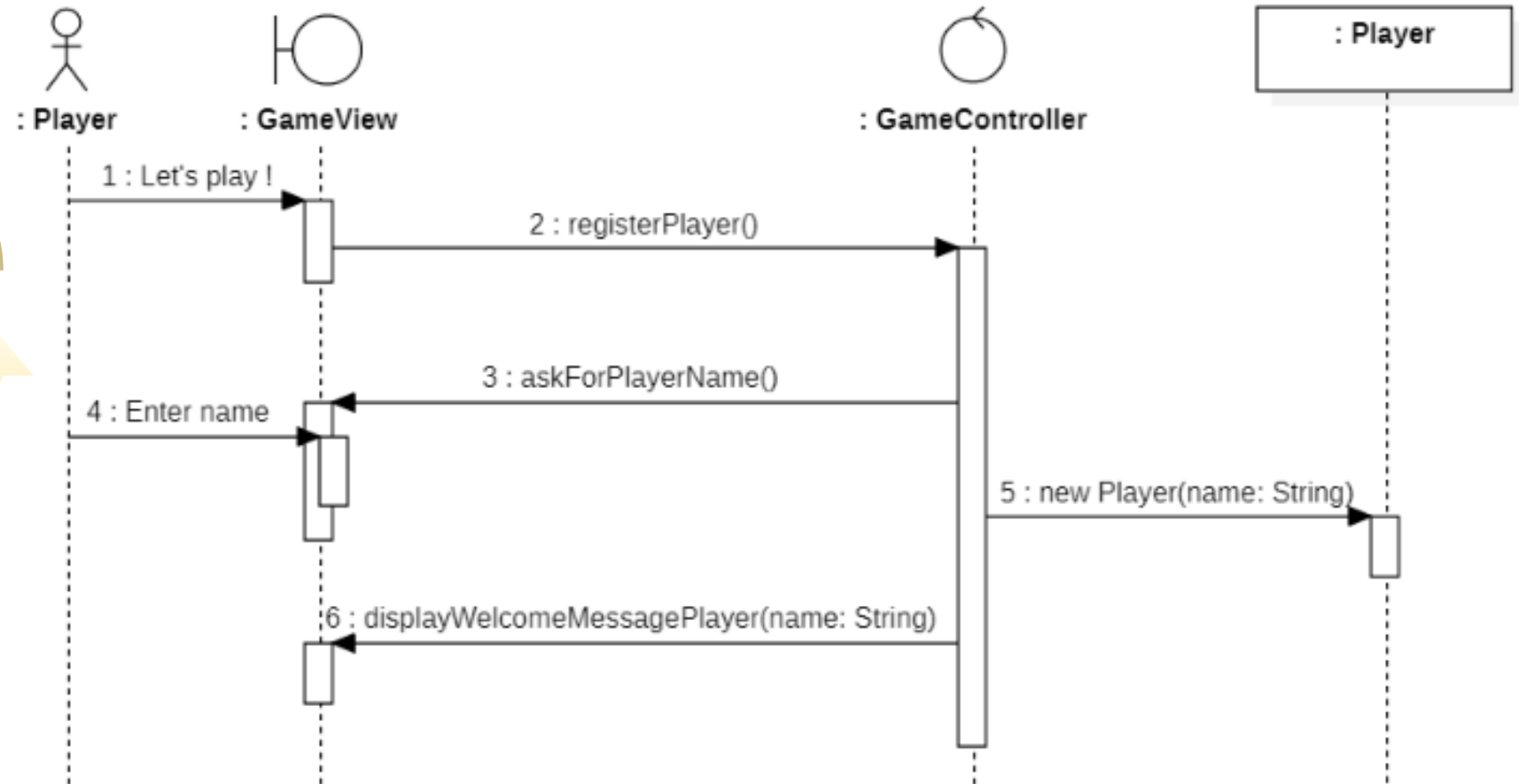
Du point de vue de la **conception objet**,
lancer un message synchrone à un objet revient à
provoquer **l'exécution d'une opération**
définie dans la classe de cet objet.

Le diagramme de séquence permet
d'enrichir le diagramme de classes
en identifiant de nouvelles **opérations**
et de nouvelles **classes**

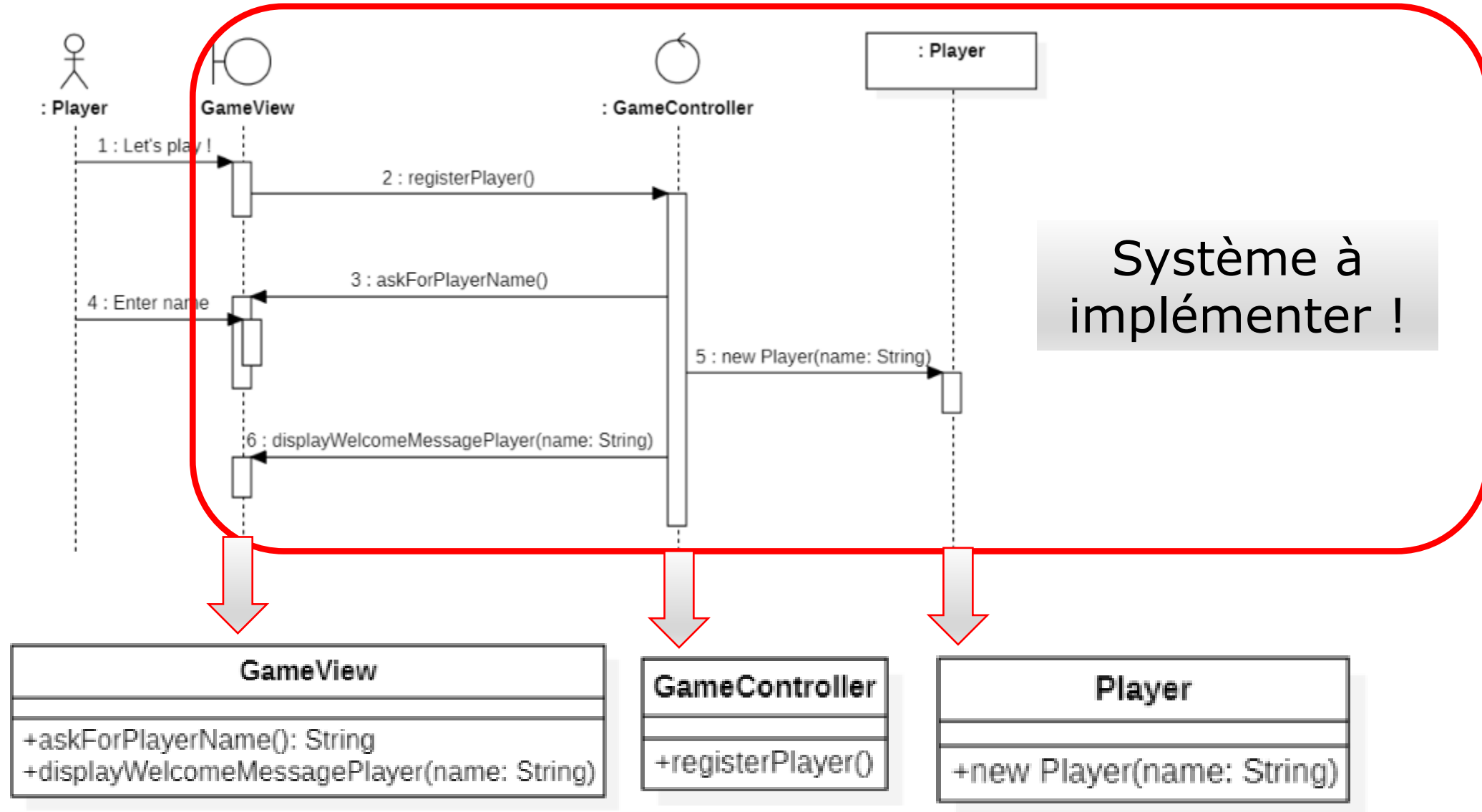
Diagramme de séquence rejoindre une partie

(plus proche du code c-a-d avec des opérations et non plus des messages)

Après quelques itérations, on pourrait obtenir le diagramme suivant (en anglais)

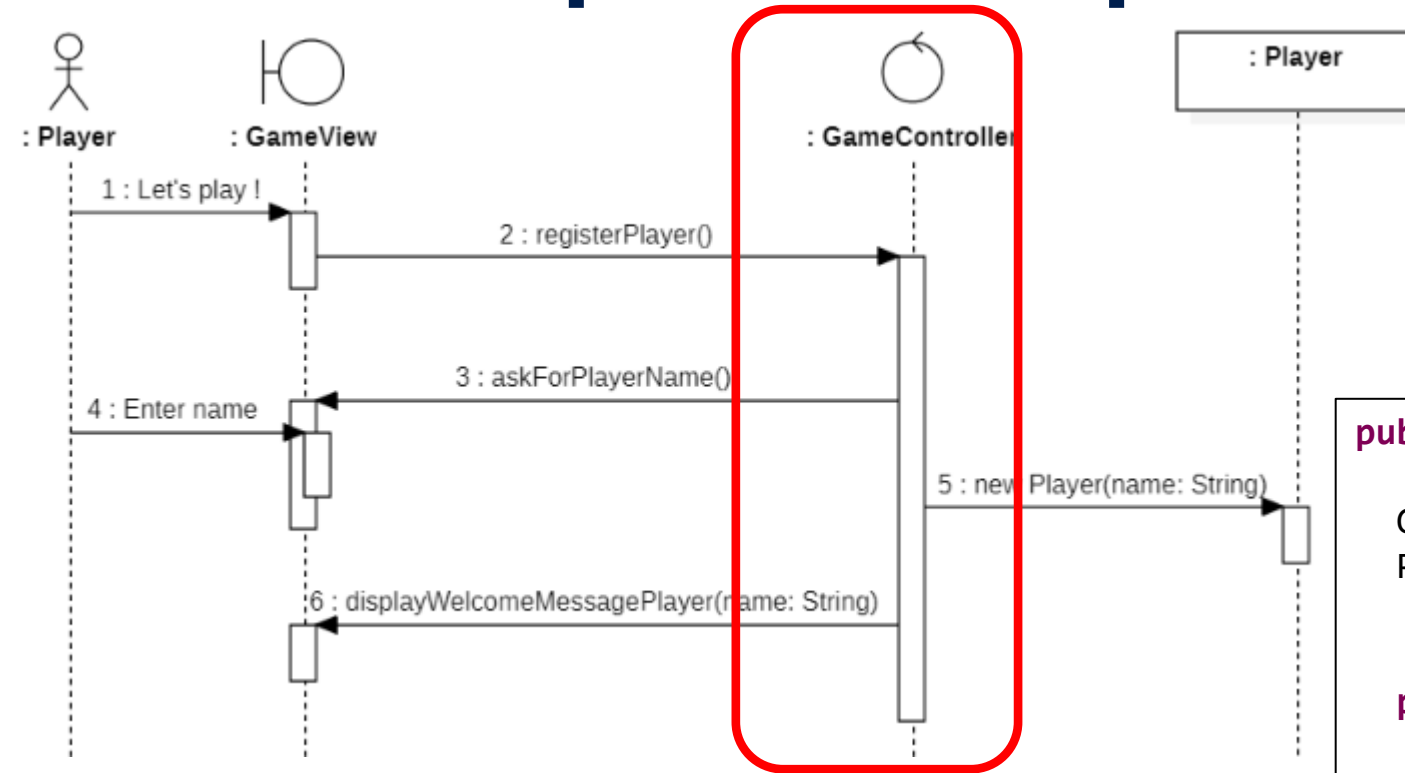


Le diagramme de séquence permet d'enrichir les classes avec les opérations !



**Après la phase de conception,
vient la phase d'implémentation !**
(Exemple d'une implémentation simplifiée
du pattern MVC)

De la conception à l'implémentation : GameController



Le ctrl **dialogue avec la vue** (envoi de messages) : la vue est donc un attribut de la classe

Le ctrl **dialogue avec la couche métier** : on peut supposer que le **joueur** sera réutilisé dans plusieurs méthodes 😊

Seule **la vue** existe au lancement du contrôleur, les objets de la couche métier apparaissent par la suite 😊

```
public class GameController {
    GameView view;
    Player player;

    public GameController(GameView view) {
        this.view = view;
    }

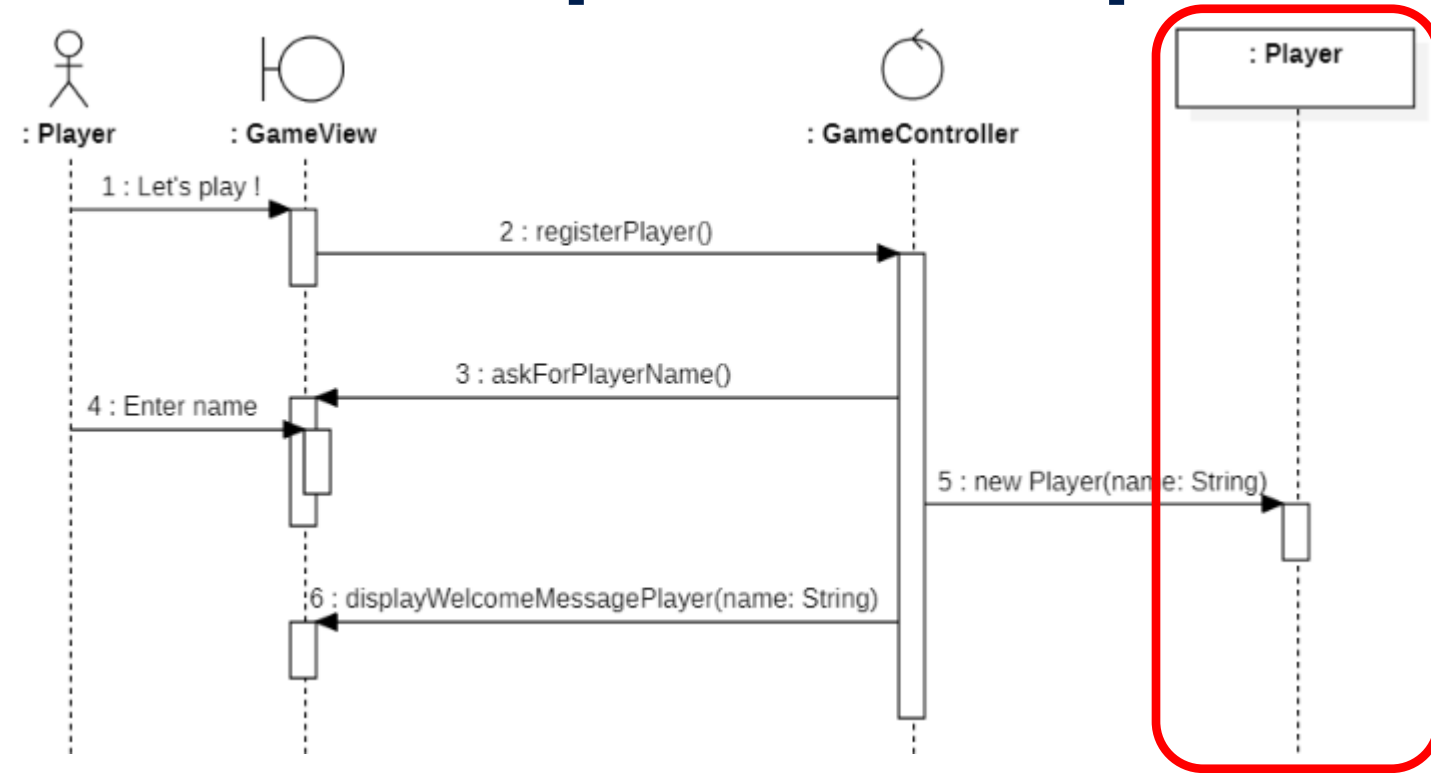
    public void registerPlayer() {
        String name = view.askForPlayerName();
        this.player = new Player(name);
        view.displayWelcomeMessagePlayer(player.name());
    }
}
```

Une méthode registerPlayer qui **orchestre** l'enchaînement des activités (**appel de méthodes**) pour satisfaire le **comportement attendu** pour le UC rejoindre une partie



Conformément au diagramme : envoi des **messages 3, 5 et 6** en utilisant les opérations adéquates des classes destinataires

De la conception à l'implémentation : Player



```
public class Player {  
  
    private final String name;  
  
    public Player(String name) {  
        this.name = name;  
    }  
  
    public String name() {  
        return this.name;  
    }  
  
    //hashCode & equals  
}
```

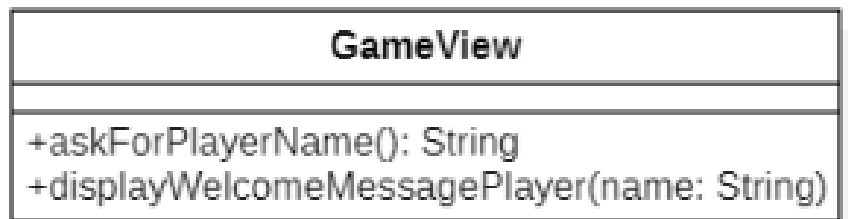
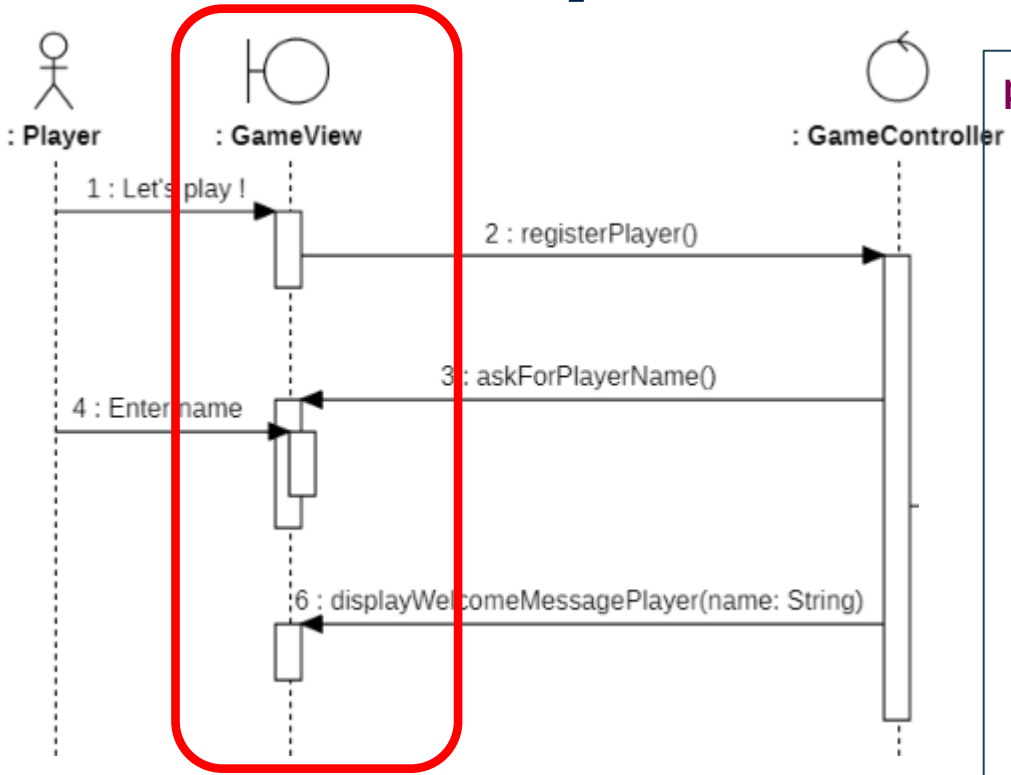
Classe métier
« classique »

Pouvant également à ce stade être
implémentée sous forme de record
pour des versions > Java 16



```
record Player(String name) { }
```

De la conception à l'implémentation : GameView



```
public class GameView {
```

```
    GameController controller;
```

```
    Scanner keyboard = new Scanner(System.in);
```

```
    public void setController(GameController gameController) {
        this.controller = gameController;
    }
```

```
    public String askForPlayerName() {
        System.out.println("Enter Player Name.");
        String name = keyboard.nextLine();
        return name;
    }
```

```
    public void displayWelcomeMessagePlayer(String name) {
        System.out.println("Welcome ! Let's start !" + name);
    }
```

```
}
```

La vue **dialogue avec le Ctrl** (envoi de messages)
: le ctrl est donc un attribut de la classe

Cette vue sera en mode console

Le ctrl est créé après la vue, donc
il faut permettre de mettre à jour
le ctrl dans la vue si le ctrl envoie
des messages à la vue (ce qui n'est
pas encore le cas ici) 😊

Lancement de l'application : GameApplication

```
public class GameApplication {  
  
    public static void main(String[] args) {  
  
        GameView gameView = new GameView();  
  
        GameController gameController = new GameController(gameView);  
        gameView.setController(gameController);  
  
        gameController.registerPlayer();  
  
    }  
}
```

Création de la **V**ue

Création du **C**ontroleur
sensé dialoguer
avec la vue précédente

Maintenant que le contrôleur est créé,
on peut enfin également établir le dialogue entre la Vue
et le contrôleur créé précédemment
depuis la classe de la Vue

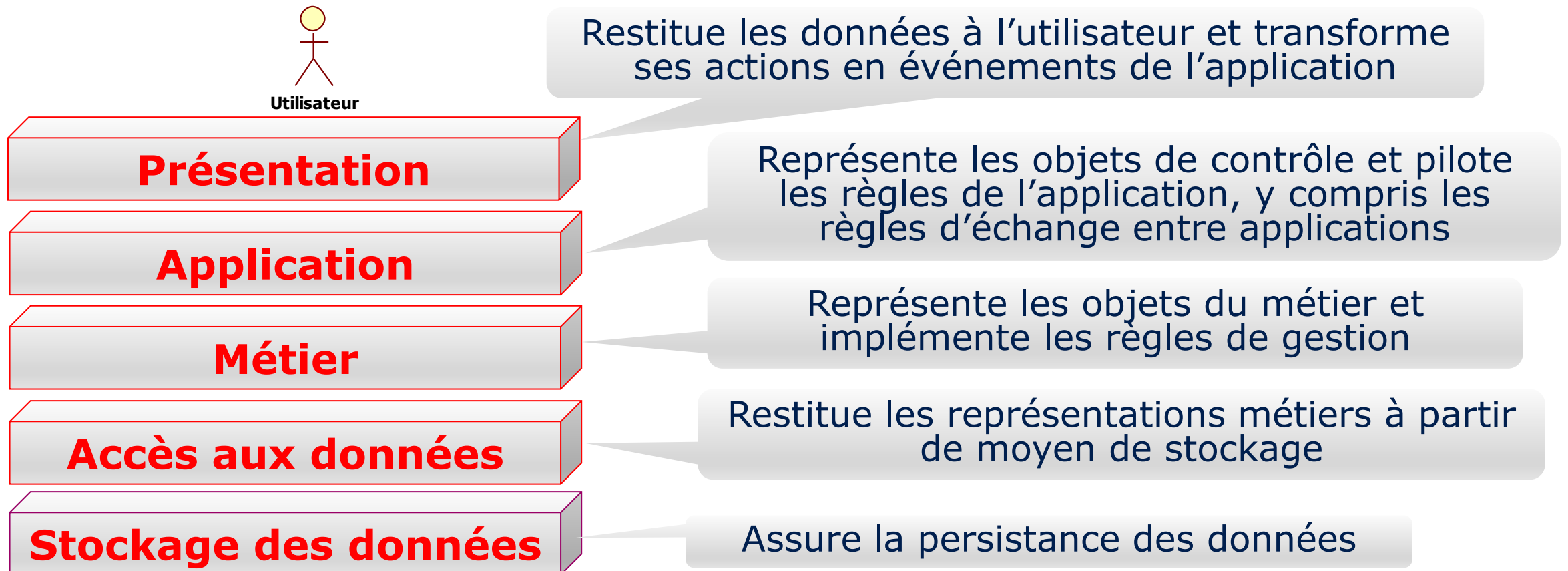
Appel du **s**ervice offert par le **c**ontrôleur qui permet
de lancer la **f**onctionnalité (UC) **s**ouhaitée

Notion d'architecture logicielle

Notion de couches logicielles

Une **couche logicielle** représente un ensemble de spécifications ou de réalisations qui expriment ou mettent en œuvre des **responsabilités techniques et homogènes** pour un système logiciel

Architecture en 5 couches : 1 responsabilité / couche



Les couches logicielles dans un modèle UML

Le **package** (élément UML permettant de regrouper d'autres éléments UML) associé au stéréotype **<<layer>>** permet de modéliser une couche

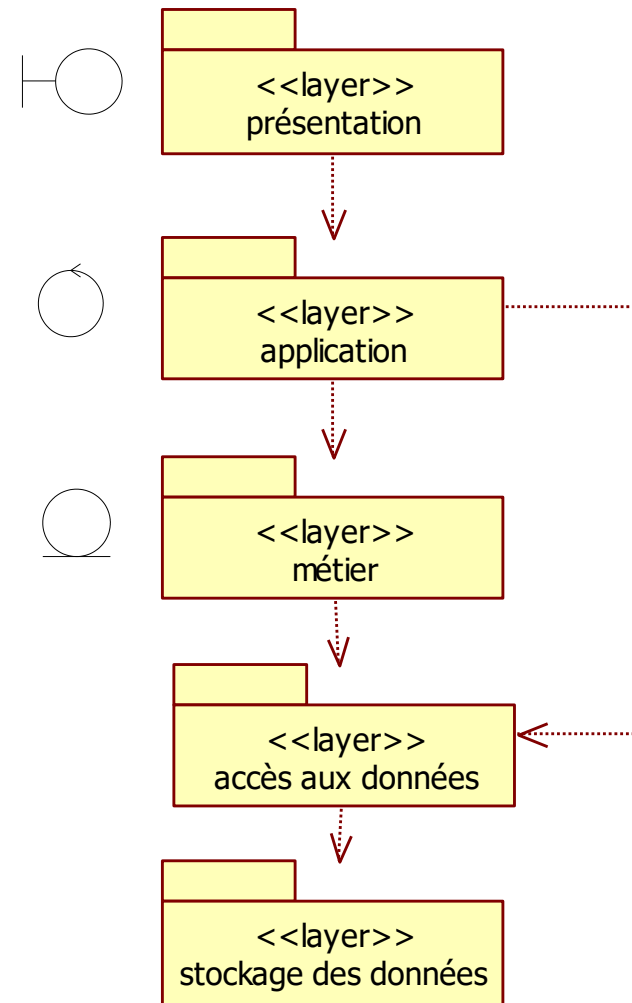
<<layer>>
une couche logicielle

Un **diagramme de packages** permet de montrer les **couches** et leurs **dépendances**.

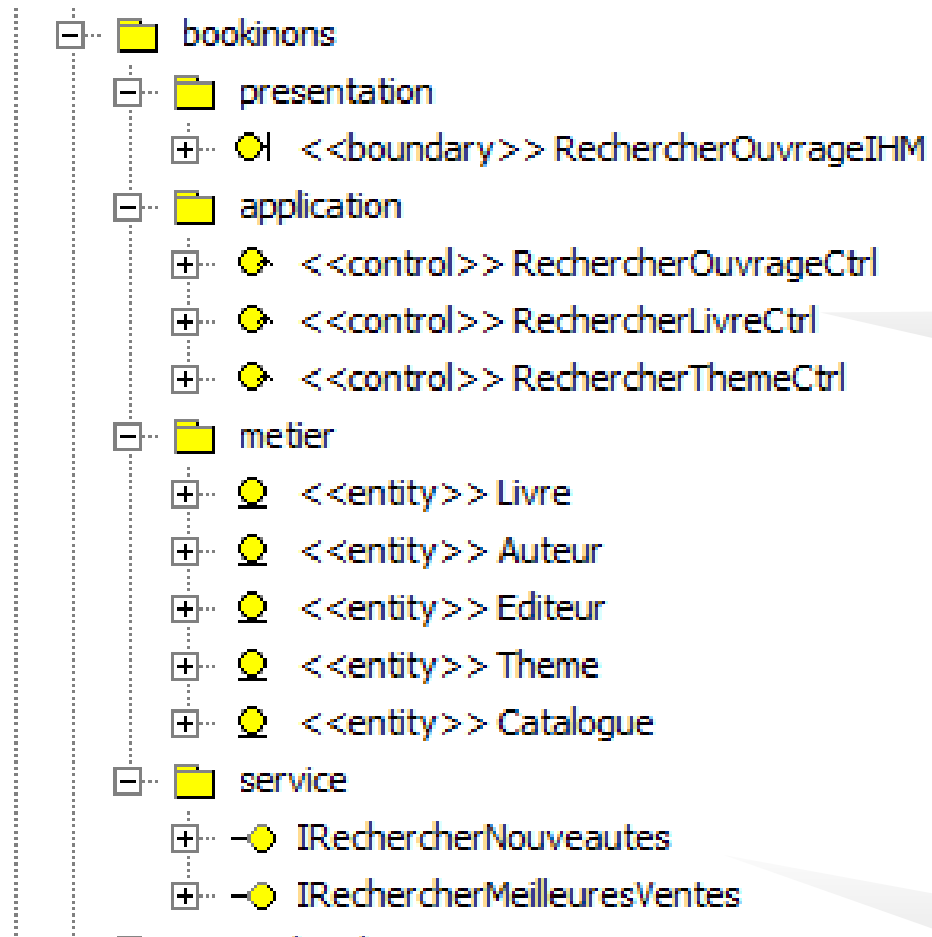
Les **dépendances** indiquent les **interactions** (*messages*) qui peuvent être envoyées d'une couche à l'autre

L'intérêt d'une architecture en couche est de pouvoir répondre à un **critère d'évolutivité**

⇒ les dépendances entre couches doivent être minimisées afin de **favoriser un faible couplage**



Les couches logicielles dans Bookinons



Habituellement, il y a plusieurs contrôleurs par application ...

Les objets stéréotypés **<<interface>>** ont été regroupés dans une couche appelée **service**.

Annexes

Opérateurs pour un fragment combiné d'interaction

Choix et boucles

- Alternatif (**alt**) : plusieurs fragments possibles. Seul celui dont la condition est vraie s'exécute
- Optionnel (**opt**) : ne s'exécute que si la condition est vraie
- Exception (**break**) : la fin de ce fragment interrompt la séquence entière
- Itération (**loop**) : le fragment peut s'exécuter plusieurs fois selon les conditions de la garde

Parallélisation

- Parallèle (**par**) : chaque fragment est exécuté en parallèle
- Critique (**critical**) : le fragment ne peut avoir qu'un thread qui s'exécute à la fois

Contrôle de l'envoi de messages

- Insignifiant (**ignore**) : les messages du fragment sont considérés comme insignifiants
- Signifiant (**consider**) : seuls les messages du fragment sont considérés comme signifiants
- Assertion (**assert**) : seul l'interaction du fragment est considérée comme valide
- Invalide (**negative**) : le fragment représente une interaction invalide

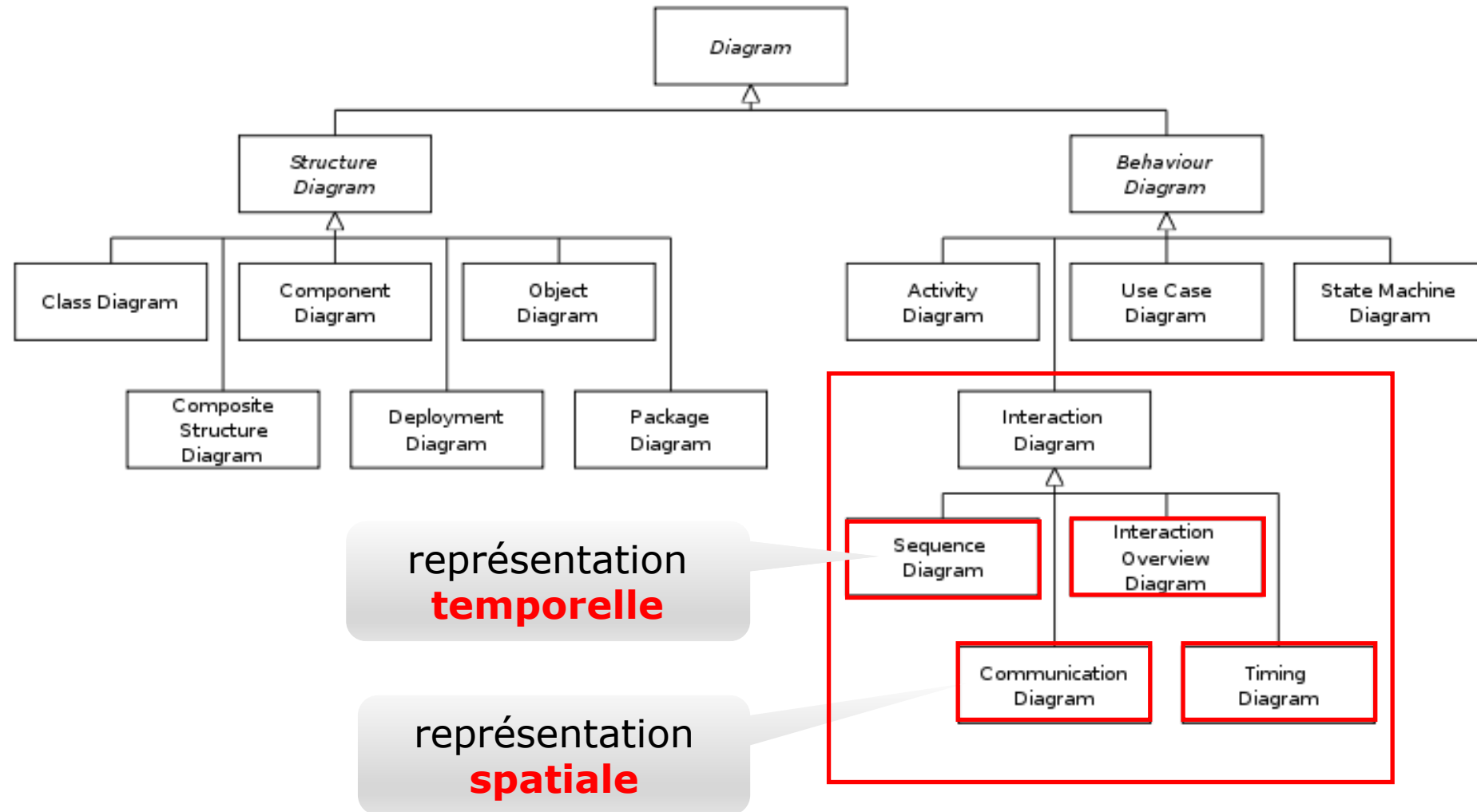
Fixe l'ordre d'envoi des messages

- Séquencement faible (**seq**) : les sous-fragment s'exécutent dans un ordre quelconque
- Séquencement fort (**strict**) : les sous-fragments s'exécutent selon l'ordre d'apparition.

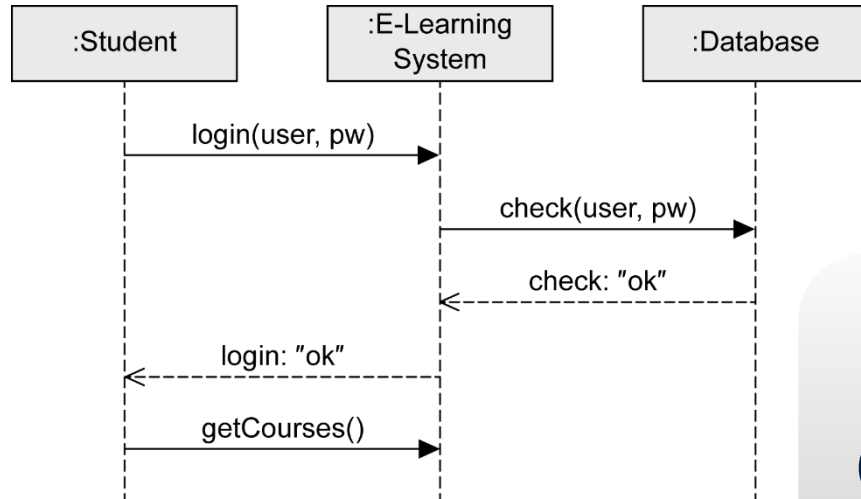
Référence

- Référence (**ref**) : référencement d'une interaction
- Diagramme de séquence (**sd**) : référencement d'un diagramme de séquence

Les 4 diagrammes d'interaction

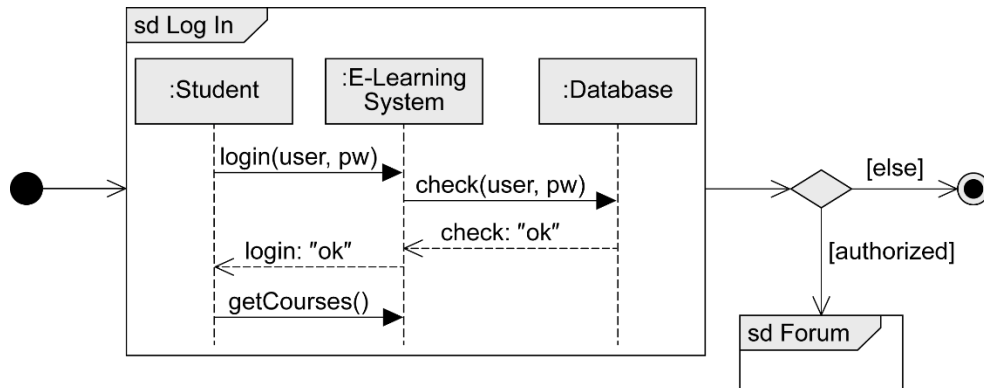


Diagrammes de séquence



Les 4 diagrammes d'interaction

Diagramme global d'interactions



Diagrammes de communication

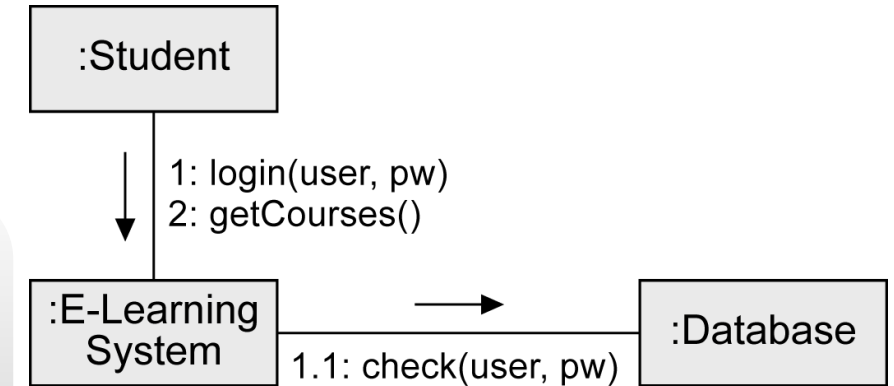


Diagramme de temps

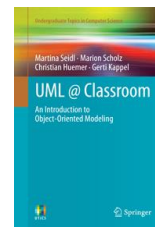
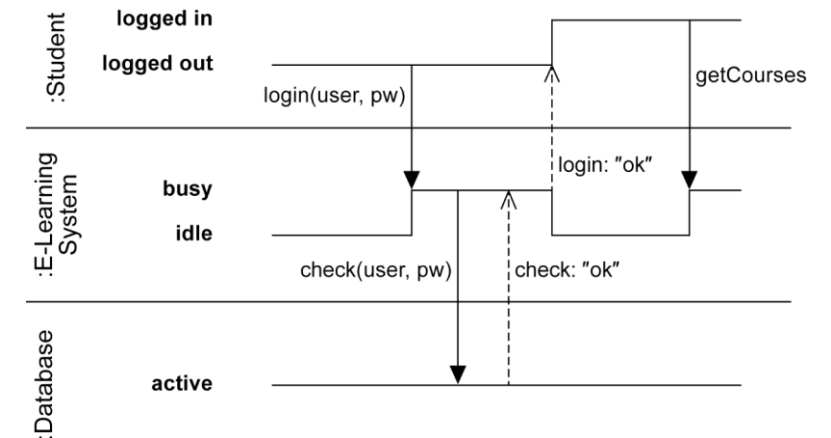
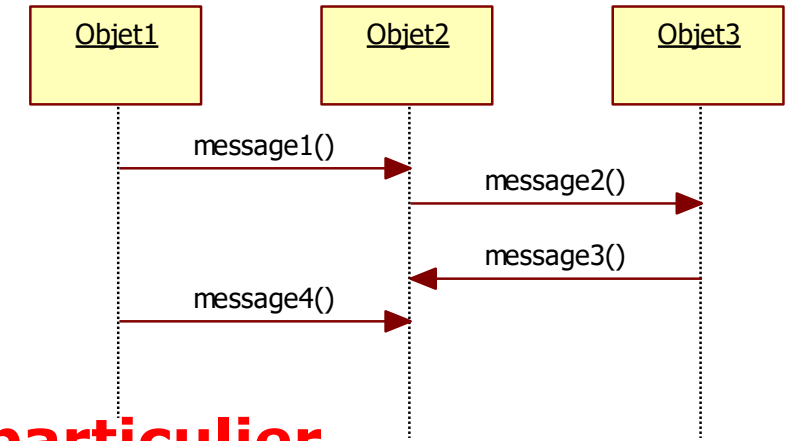


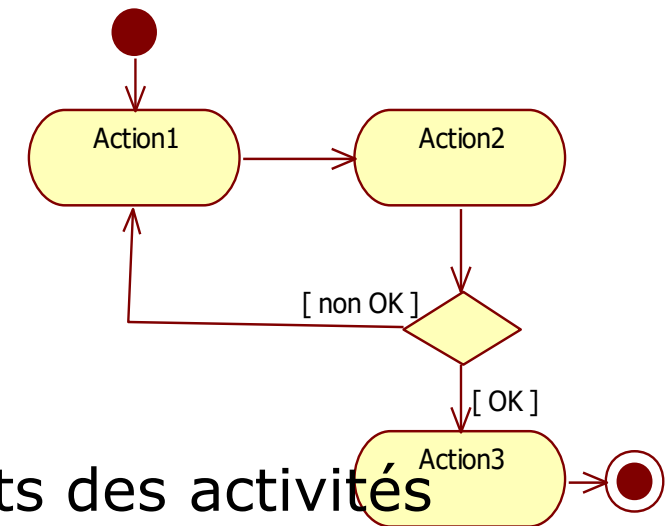
Diagramme de séquence vs Diagramme d'activité

Le **Diagramme de séquence** propose une **représentation temporelle (séquentielle)** du déroulement des traitements et des interactions entre les éléments du système et/ou de ses acteurs



⇒ Il permet d'illustrer graphiquement **un scénario en particulier**.

Le **Diagramme d'activité** représente les **règles d'enchaînements** des actions et des décisions au sein d'une activité : c'est un graphe orienté d'actions et de transitions



⇒ Il permet de documenter graphiquement les enchaînements des activités au sein d'un cas d'utilisation puisqu'il est possible **d'identifier d'un seul coup d'œil la famille de tous les scénarios du cas d'utilisation** et d'envisager ainsi toutes les possibilités d'exécution offertes par ce Use Case.

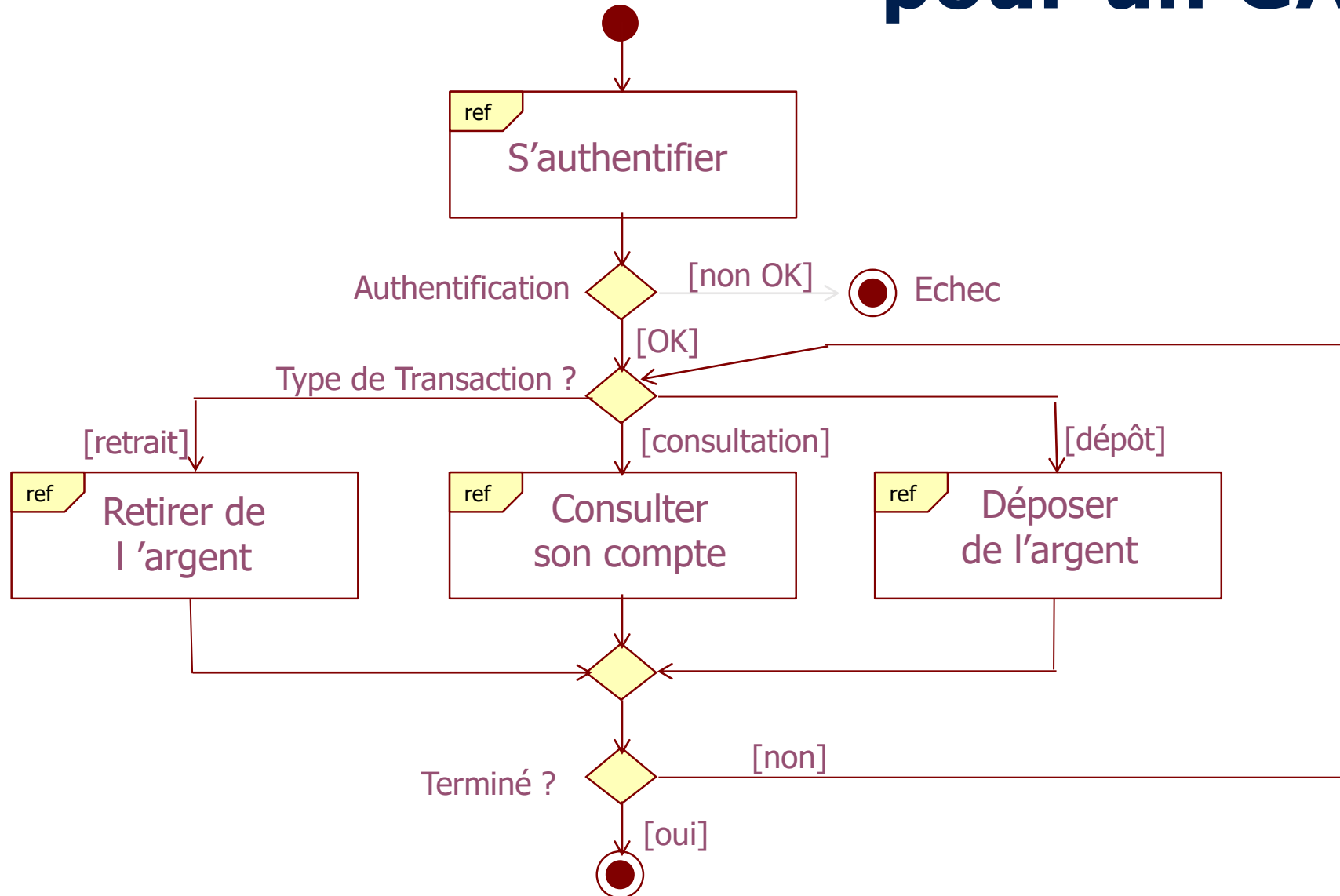
Interaction Overview Diagram

UML 2 a proposé un nouveau type de diagramme, issu de la *fusion* des notations du diagramme d'**activité** et du diagramme de **séquence**, qui est appelé : **Interaction Overview Diagram** (ou diagramme global d'interactions)

L'**Interaction Overview Diagram** permet d'**organiser des interactions** (représentées par exemple par des diagrammes de séquence) **au moyen de noeuds de contrôle** (présents habituellement dans un diagramme d'activités).

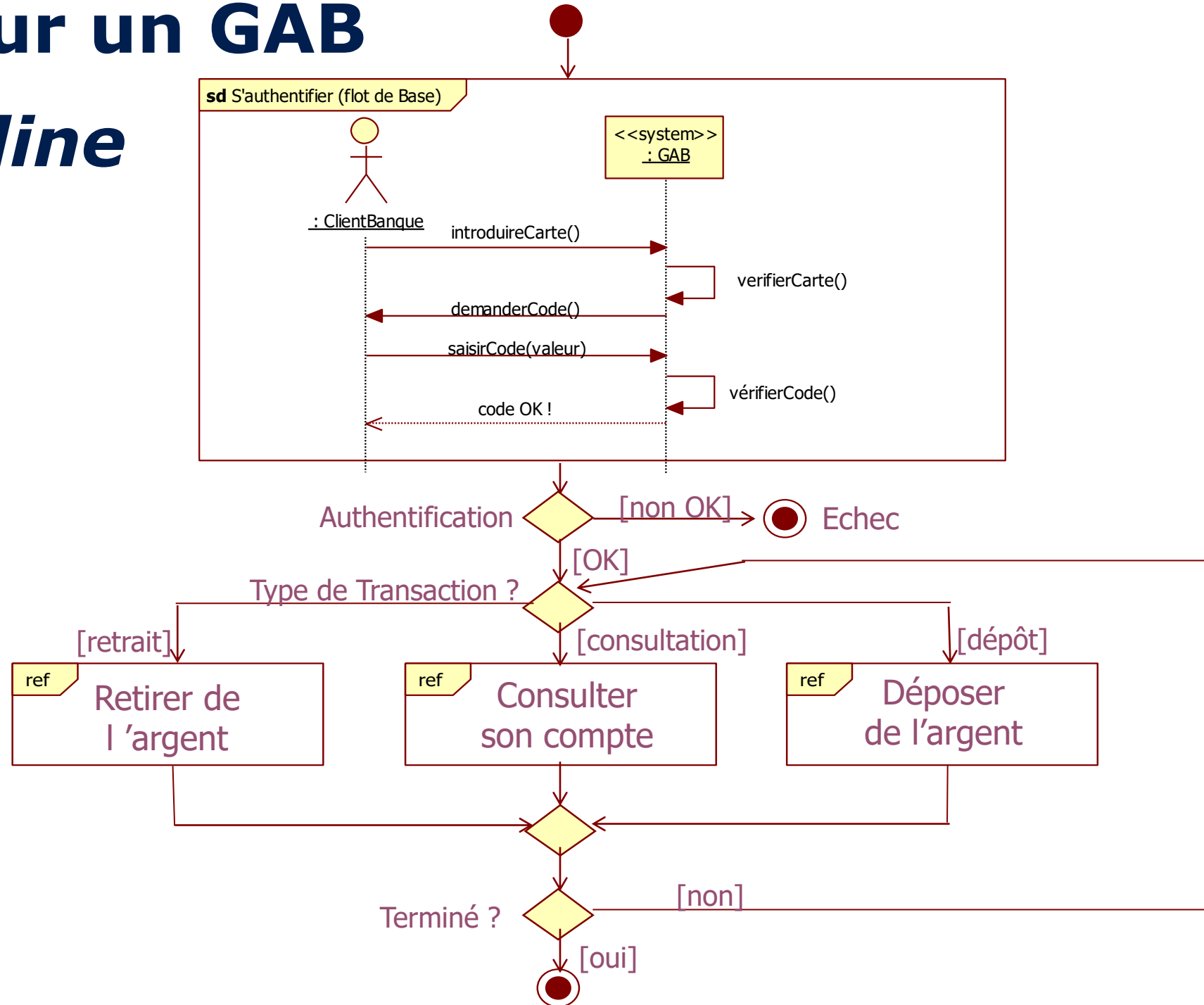
L'**Interaction Overview Diagram** est une sorte de diagramme d'activités où les actions sont remplacées par des interactions.

Interaction Overview Diagram pour un GAB

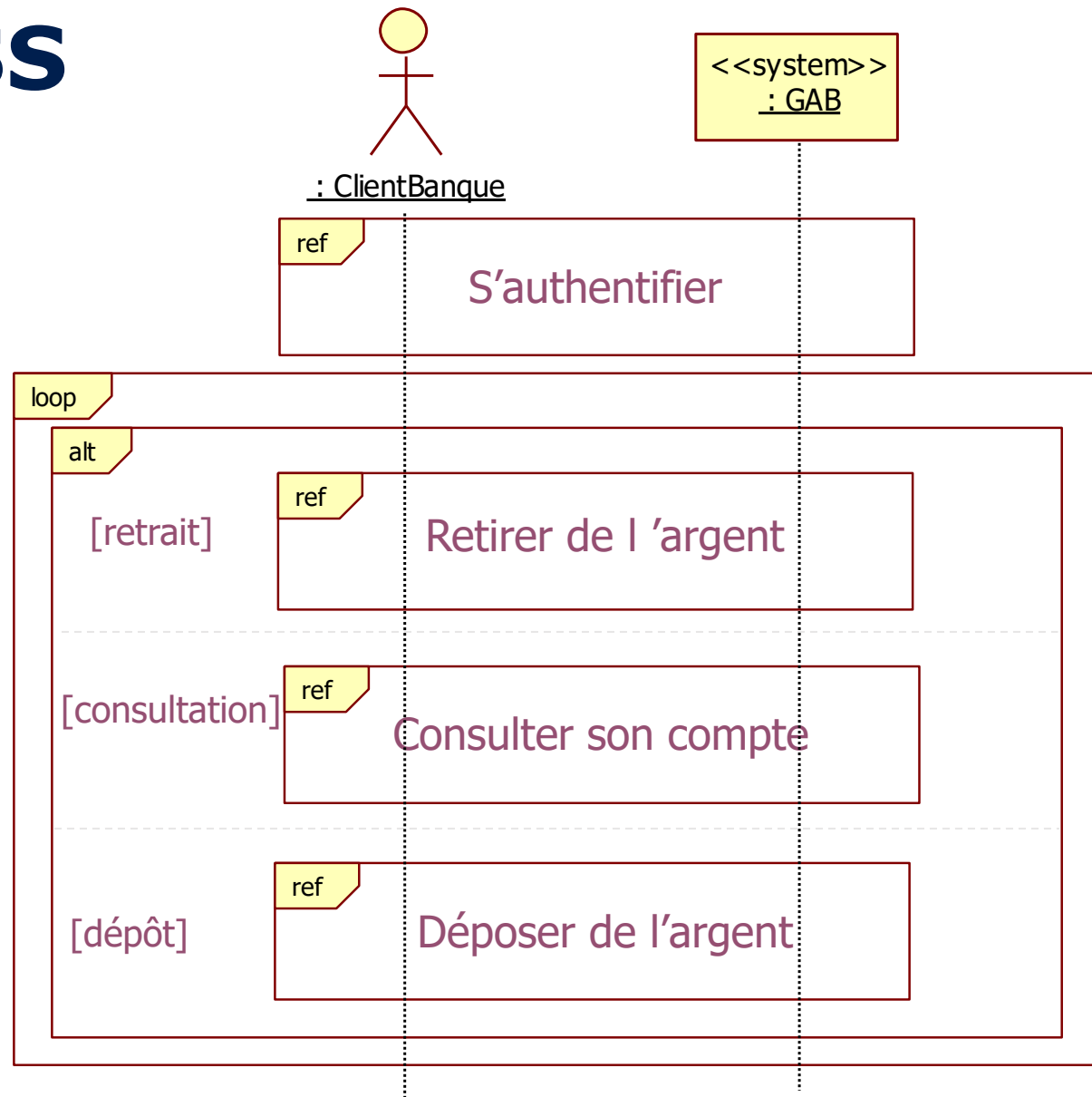


Remarque : il est également possible de remplacer chaque référence (**ref**) par un diagramme de séquence *in line*

IOD pour un GAB avec DSS *in line*



IOD vs DSS



La valeur ajoutée de l'Interaction Overview Diagram par rapport au Diagramme de Séquence n'est pas vraiment évidente sur cet exemple. IOD pourra par exemple être utilisé pour la description d'une méthode complexe

Autre Exemple de modélisation d'un diagramme de séquences













Un cas d'usage qui consiste à Rechercher un Ouvrage

Livres

Les meilleures ventes

Nos produits les plus populaires selon les ventes. Mises à jour chaque heure.

Les meilleures ventes en Livres

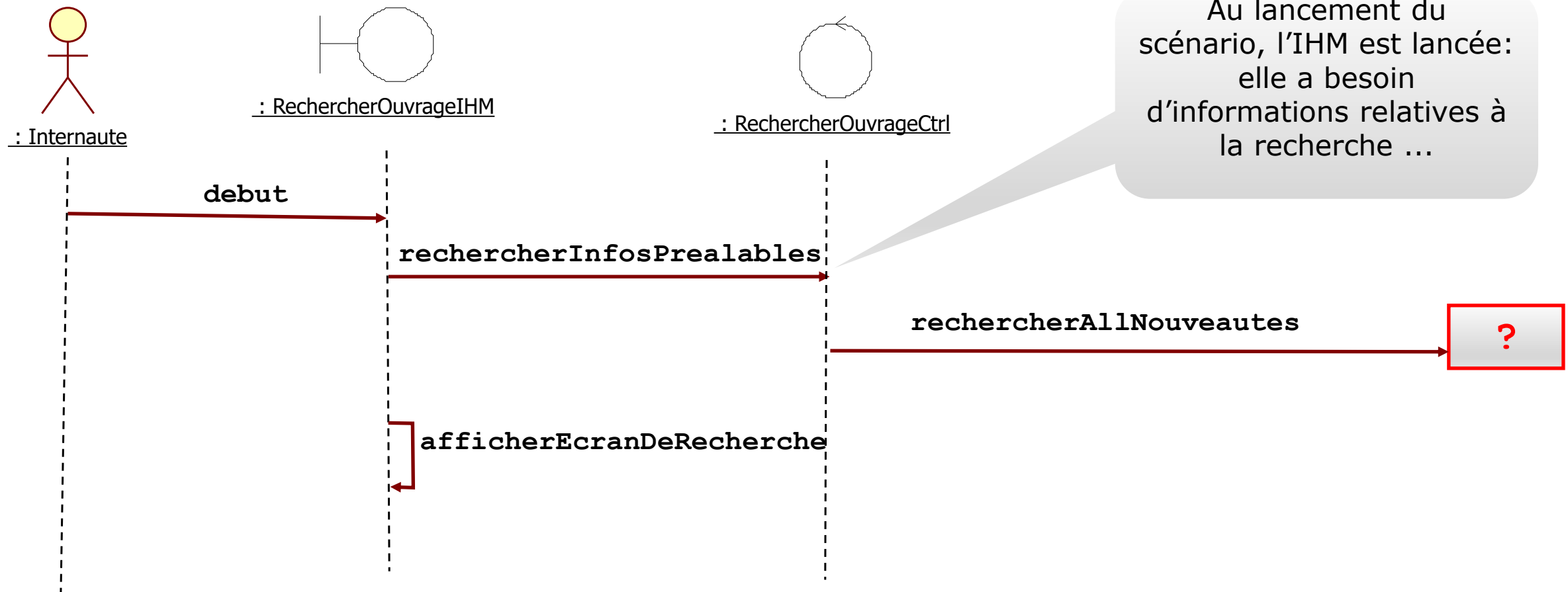
<div>#1</div> <div></div> <div>Être mère, c'est que du bonheur... ou pas ! Daniela Martins Broché 14,90 €</div>	<div>#2</div> <div></div> <div>Les fossoyeurs: Révélation sur le système qu... Victor Castanet ★★★★☆ 222 Broché 22,90 €</div>	<div>#3</div> <div></div> <div>L'Attaque des Titans T33 Edition limitée > Hajime Isayama ★★★★★ 805 Broché 11,50 €</div>	<div>#4</div> <div></div> <div>Programme officiel LFI L'avenir en commun... > Jean-Luc Mélenchon ★★★★★ 256 Broché 3,00 €</div>	<div>#5</div> <div></div> <div>Solo Leveling T05 Chugong ★★★★★ 4 Broché 14,95 €</div>	<div>#6</div> <div></div> <div>Climat, crises: Le plan de transformation de... The Shift Project ★★★★☆ 15 Broché 11,90 €</div>
<div>#7</div> <div></div> <div>Spy x Family - T7 (7) > Tatsuya Endo ★★★★★ 2 Broché 6,90 €</div>	<div>#8</div> <div></div> <div>Parce que tu es un garçon formidable: Des... Sofia Beauvais ★★★★★ 511 Broché 11,95 €</div>	<div>#9</div> <div></div> <div>Dieu - La science Les preuves > Michel-Yves Bolloré ★★★★★ 986 Broché 24,00 €</div>	<div>#10</div> <div></div> <div>La Clé de votre énergie: 22 protocoles pour... Natacha Caletremre ★★★★★ 3 245 Broché 19,90 €</div>	<div>#11</div> <div></div> <div>Connemara > Nicolas Mathieu ★★★★★ 35 Broché 22,00 €</div>	<div>#12</div> <div></div> <div>Le Grand Monde > Pierre Lemaître ★★★★★ 182 Broché 22,90 €</div>

Scénario à modéliser (flot de base)

Rechercher un Ouvrage à partir de mots clés

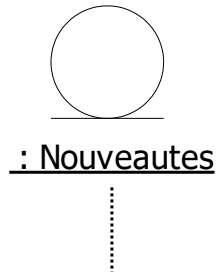
1. **Le système** recherche les informatives relatives à la recherche (Meilleures ventes, Nouveautés...) et affiche l'écran de recherche ...
2. **L'Internaute** saisit un ou plusieurs mots-clés (un thème, un titre, un auteur, un nom d'auteur) et valide.
3. **Le système** recherche dans le catalogue les ouvrages pouvant correspondre à la demande de l'utilisateur.
4. **Le système** trie les ouvrages dans l'ordre souhaité.
5. **Le système** affiche dans une page de résultat un résumé des ouvrages trouvés.
6. **L'Internaute** sélectionne un ouvrage.
7. **Le système** recherche le détail de l'ouvrage.
8. **Le système** affiche une fiche détaillée de l'ouvrage qui contient :
 - une image de l'ouvrage, le titre, l'auteur
 - l'éditeur, l'isbn, la langue, la date de parution
 - le prix et la disponibilité.
9. **L'Internaute** choisit de quitter.
10. **Le système** ferme le Use Case.

1. Le système recherche les informatives relatives à la recherche (Nouveautés, Meilleures ventes,...) et affiche l'écran de recherche



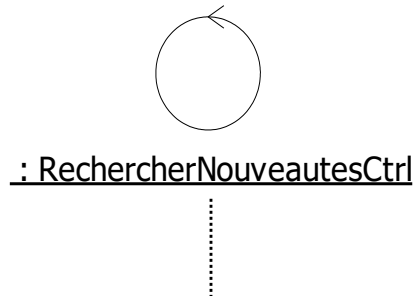
Comment récupérer les nouveautés ?

Lorsqu'un nouvel objet est identifié, 3 possibilités sont envisageables



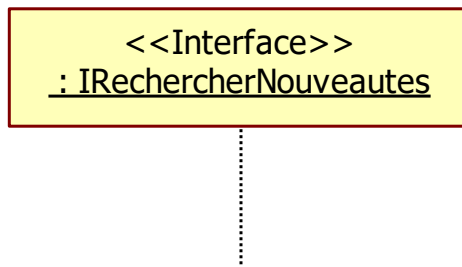
Une **classe métier** (<<entity>>)

- classe de « notre responsabilité »
- appel à un service à implémenter au sein du UC



Un **contrôleur** « secondaire » (<<control>>)

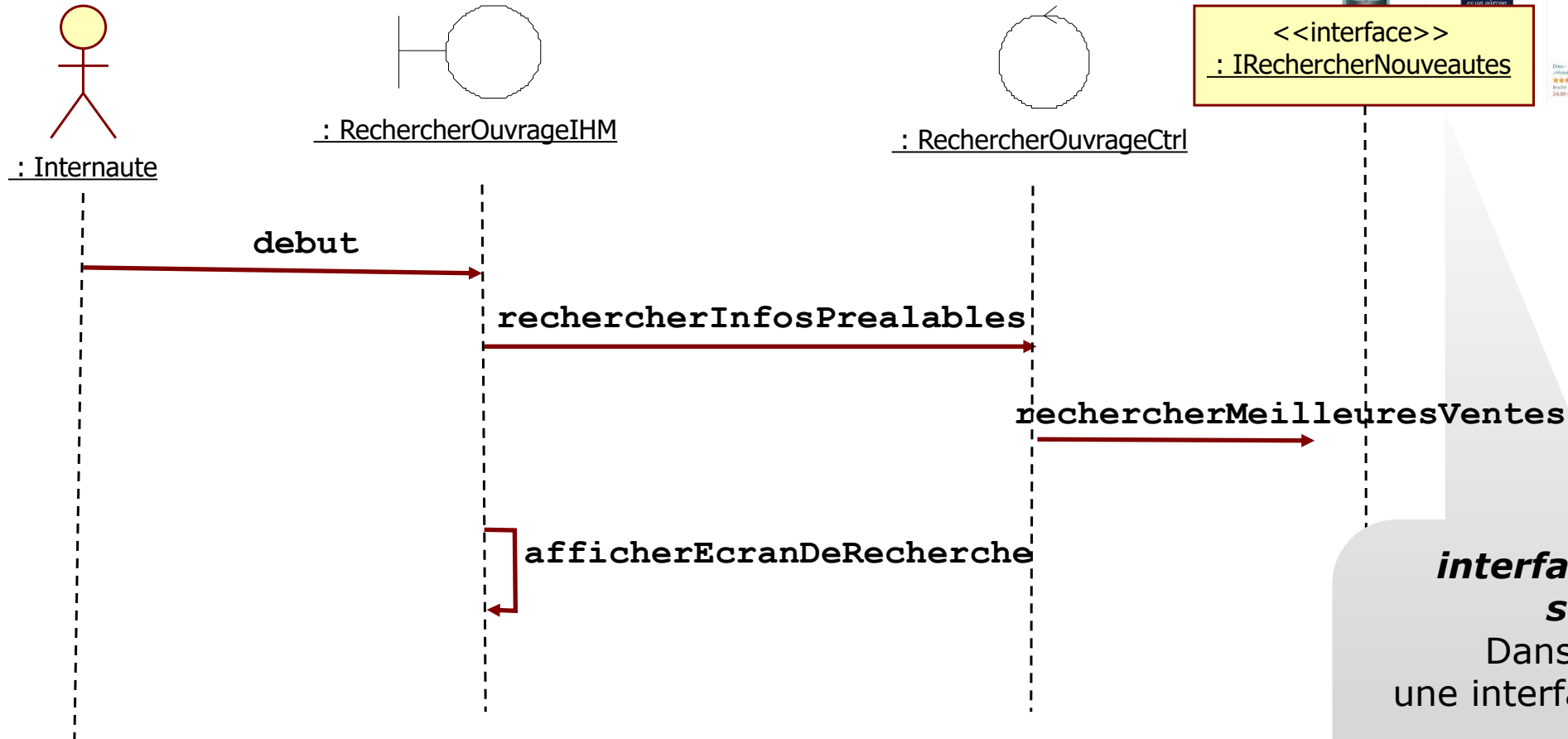
- référentiel **INTERNE**
- appel à un **service interne** réalisée par un autre UC de l'application.



Une **interface** (<<interface>>)

- référentiel **EXTERNE**
- appel à un **service externe** (hors périmètre)

1. Le système recherche les informatives relatives à la recherche (Meilleures ventes) et affiche l'écran de recherche



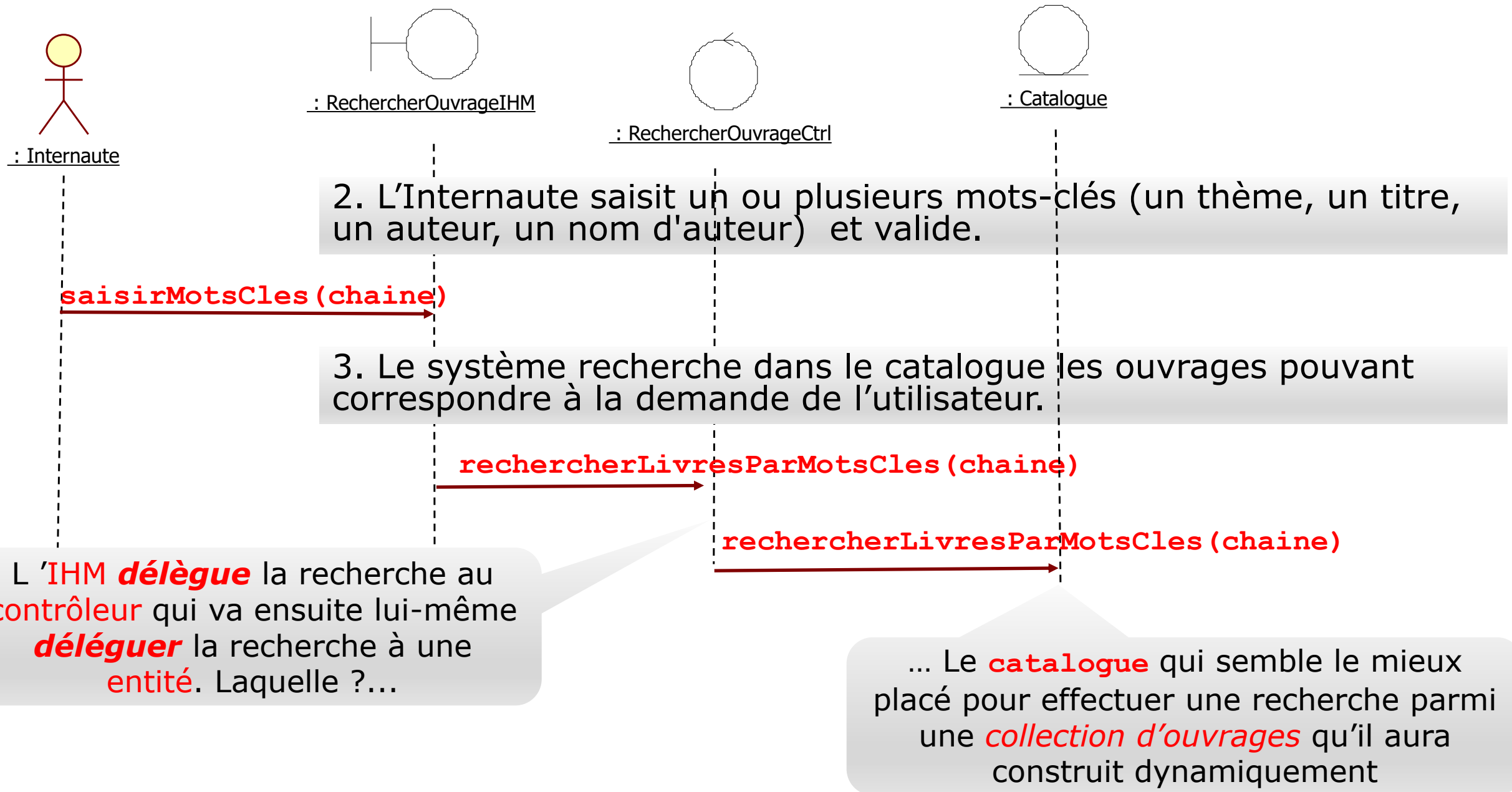
<<interface>>
: IRechercherNouveautés



interface ou contrôleur secondaire ?

Dans le doute choisir une interface qui pourra ensuite être affinée en contrôleur secondaire (si dans le périmètre fonctionnel de l'application)

Pas de message de retour pour ne pas alourdir le diagramme : Rappel: Le retour est implicite avec un appel synchrone !



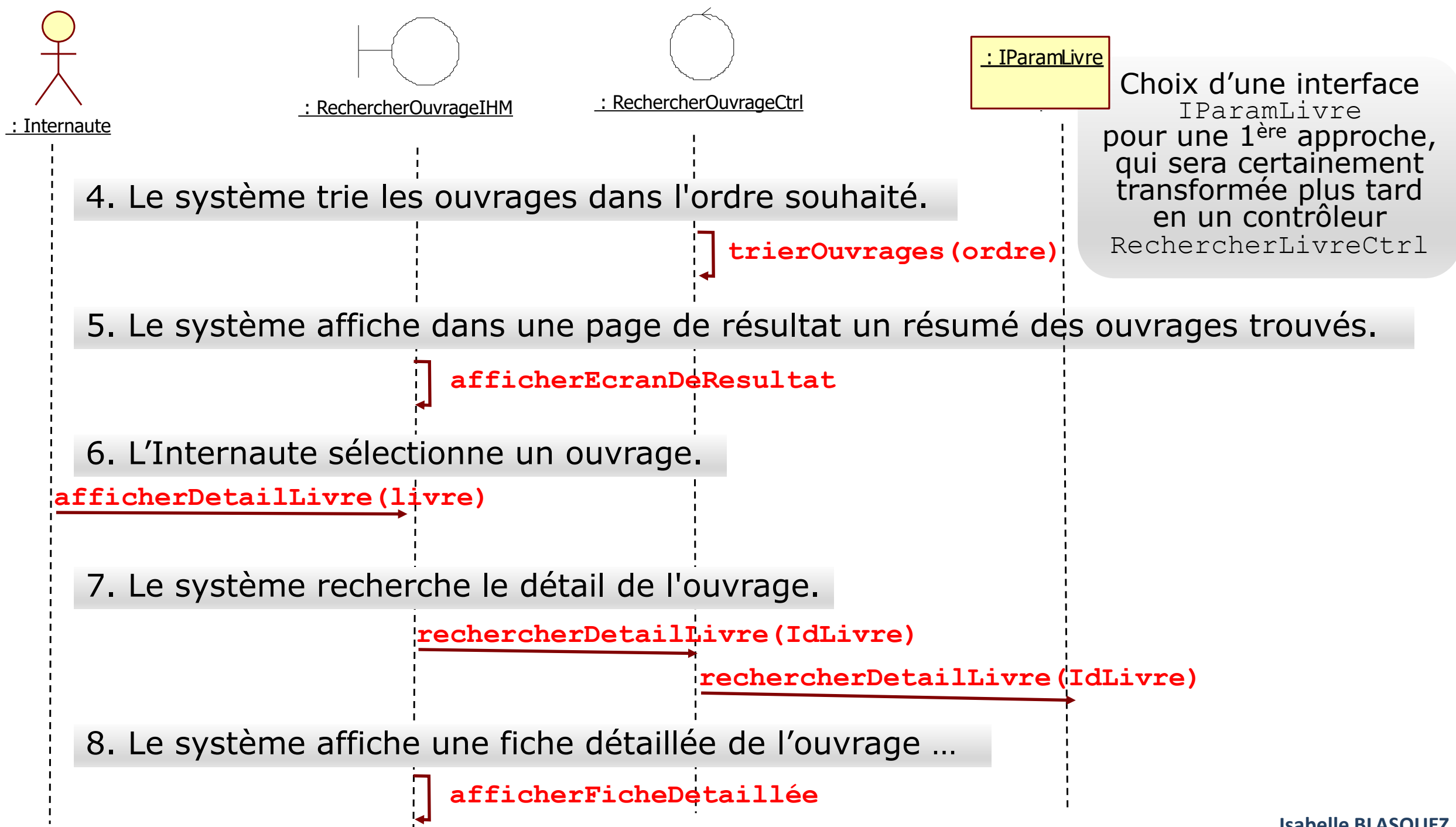
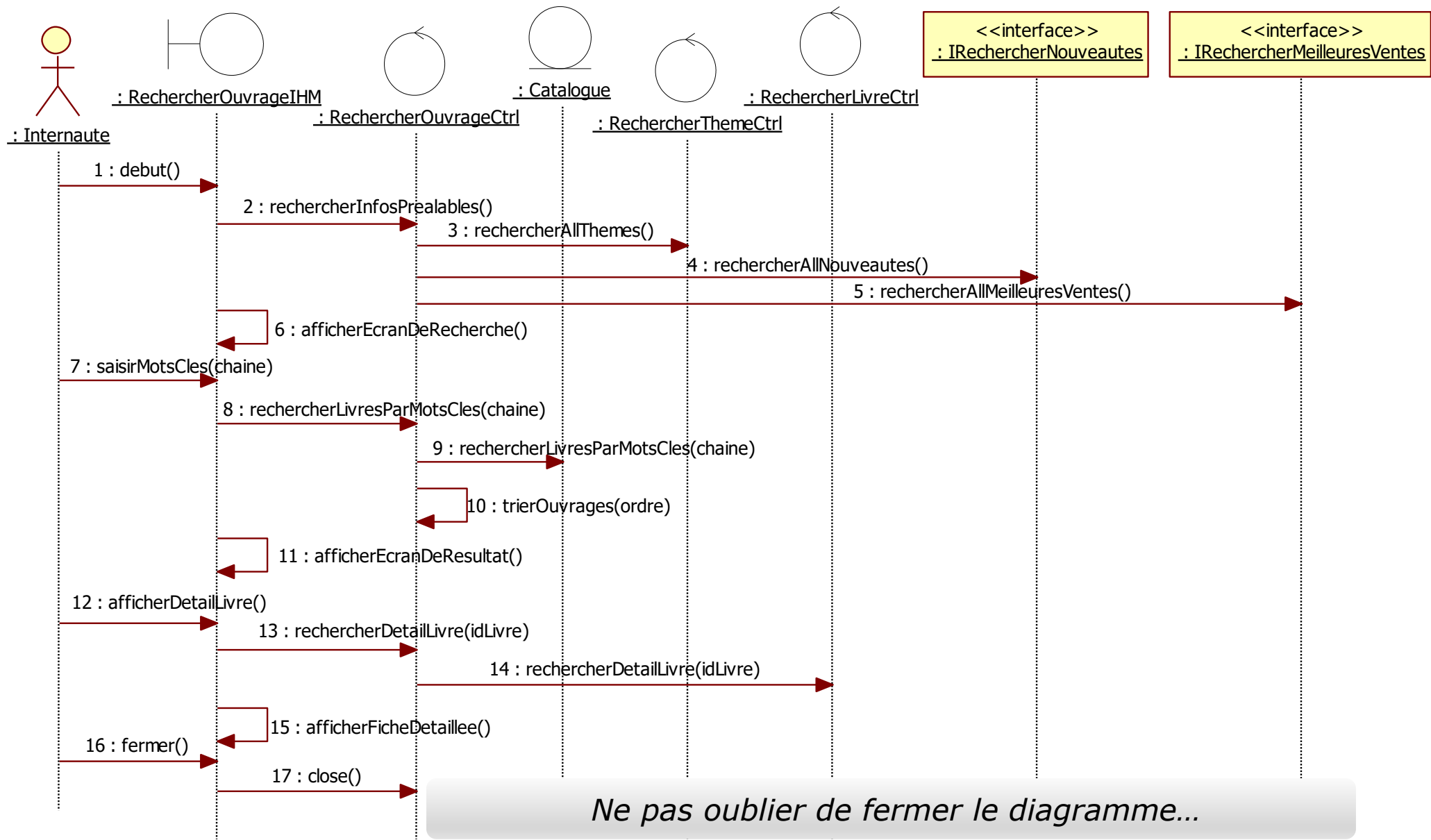


Diagramme de séquence du Scénario (flot de base)

Rechercher un Ouvrage à partir de mots clés



DS en tant qu'outil de documentation : Exemple

L'idée : déclencher un paiement par la voix :
What if a merchant could **accept Apple Pay transactions using only their voice?**



voir la video [ici](#)

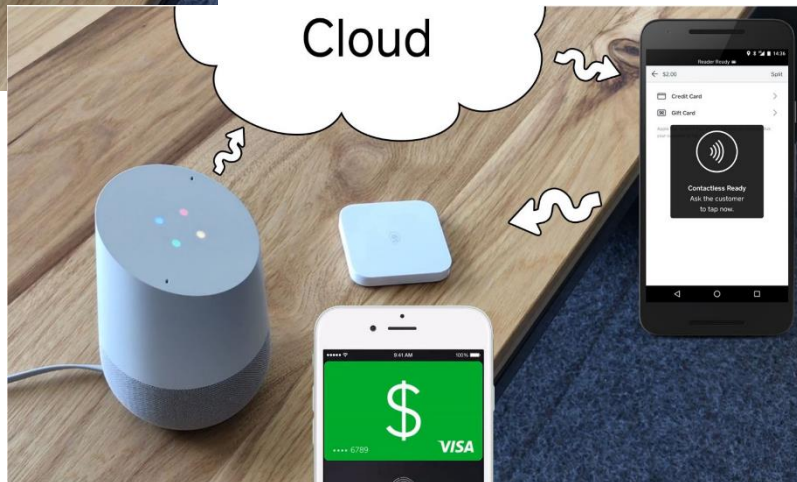
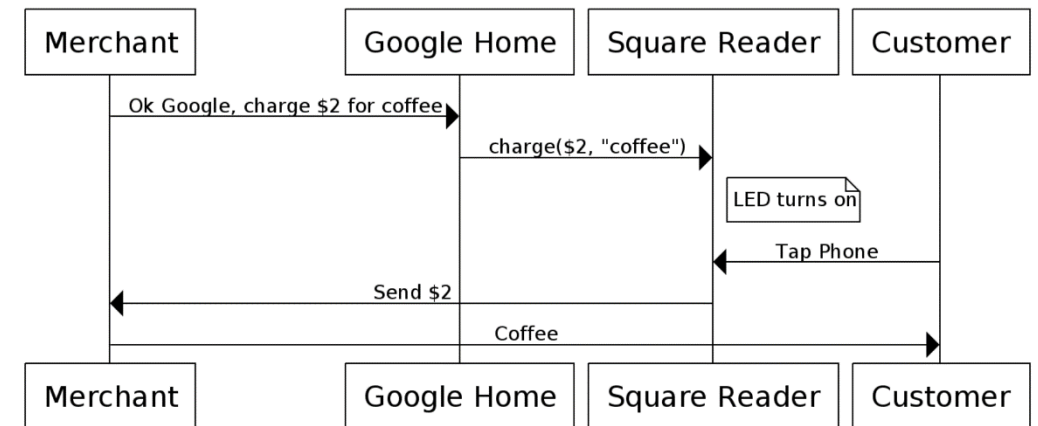


Diagramme de séquence système pour expliquer le principe de fonctionnement (grossièrement « gros grain »)

Turning voice into coffee

Here's what seems to be happening:



Google Home to activate the Square Contactless Reader and take a real Apple Pay transaction backed by a Square Cash virtual card (only public APIs)

DS en tant qu'outil de documentation : Exemple

**Diagramme de séquence
pour détailler le traitement**
(plus finement « en ouvrant la boîte »)

The Full Picture

Now that we know the steps needed to turn voice into coffee, we can update our initial sequence diagram to include all the interactions taking place:

