



Modélisation UML

3

Ouassila Labbani-Narsis
ouassila.narsis@u-bourgogne.fr

4. Diagramme d'états-transitions

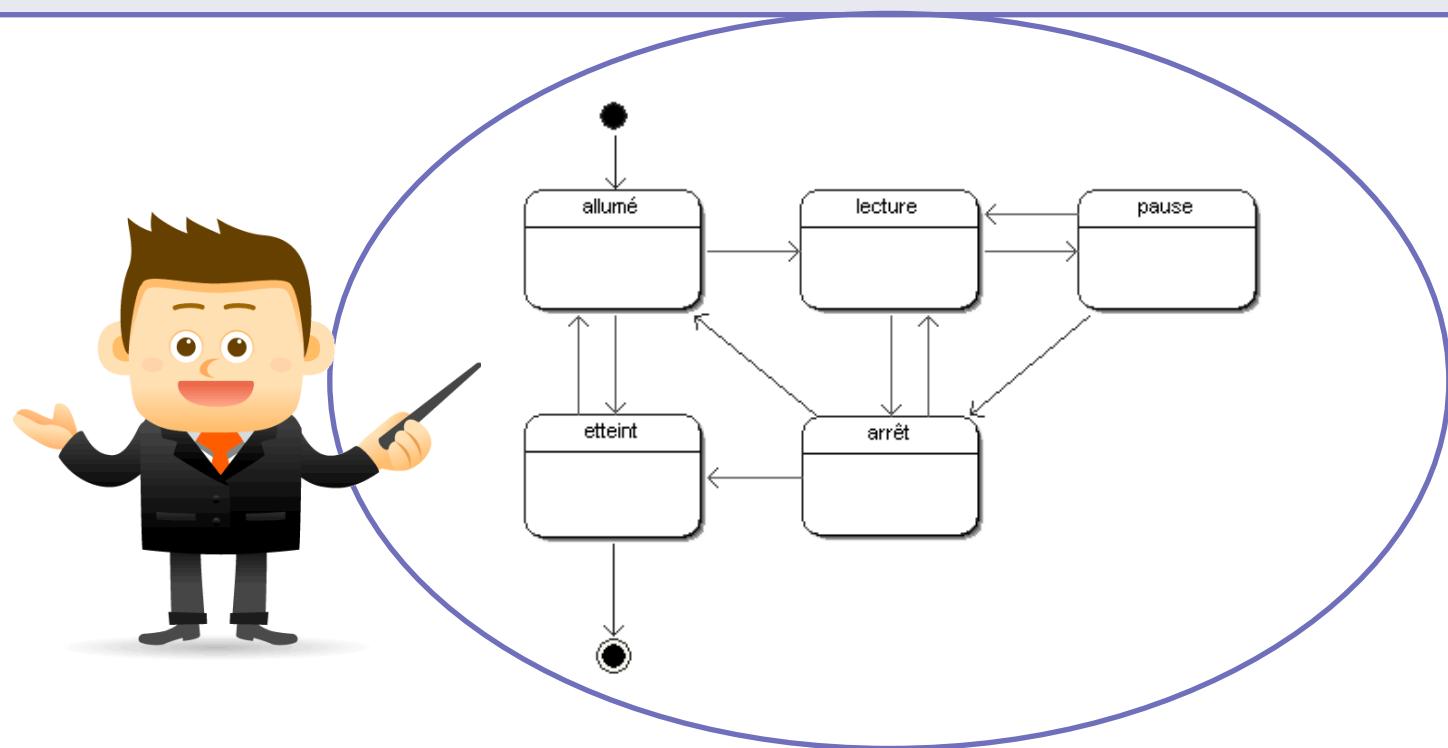


Diagramme d'états-transitions

- Le diagramme états-transitions (**State Machine Diagram**) permet de décrire le fonctionnement d'un objet ayant un comportement dynamique et séquentiel
- Il représente les différents **états** dans lesquels peut se trouver l'objet ainsi que la façon dont cet objet passe d'un état à l'autre en réponse à des **événements**
- Un diagramme d'états-transition est toujours associé à une et une seule classe et permettent de représenter les états d'un objet ainsi que les transitions entre ces états

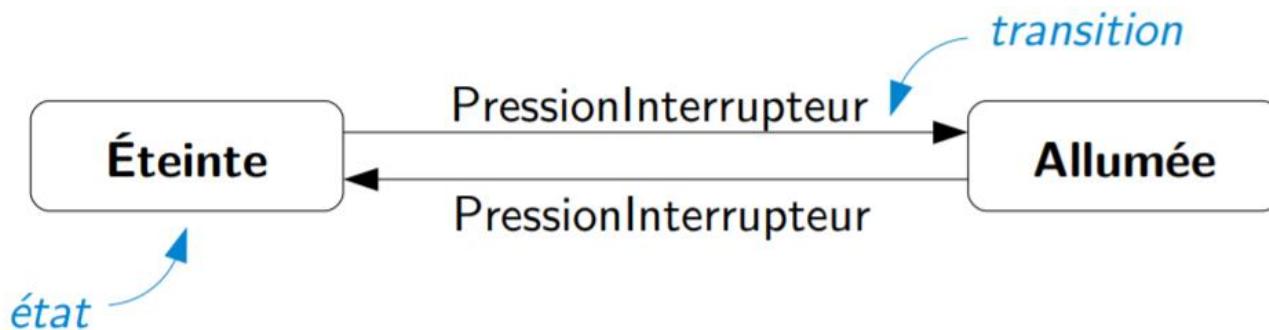


Diagramme E/T - Concepts

□ Etat :

- a tout instant un état représente l'ensemble des valeurs des attributs et des liens avec les autres objets
- un état est caractérisé par une **durée / stabilité**

□ Transition :

- connexion **unidirectionnelle** reliant 2 états
- assure le passage d'un état à l'autre
- est supposée **instantanée**

□ Événement :

- occurrence de quelque chose qui arrive à un instant inconnu ou connu (interruption, envoi de message, ...)
- peut être chargé d'information
- doit être prévu par la conception
- l'événement est le déclencheur d'une transition

Diagramme E/T - Etats

- Un objet peut passer par une **série d'états** pendant sa durée de vie
- L'état de l'objet correspond donc à un moment de son cycle de vie
- Un état représente une période dans la vie d'un objet pendant laquelle ce dernier attend un événement ou accomplit une activité
- Un état se caractérise par sa **durée** et sa **stabilité**, il représente l'ensemble des valeurs des attributs d'un objet
- Un objet ne peut pas être dans un état inconnu ou non défini

Diagramme E/T - Etats

- Pendant qu'il se trouve dans un état, un objet peut se contenter d'attendre un signal provenant d'autres objets. Il est alors **inactif**
- Il peut également être **actif** et réaliser une activité
- Une **activité** est l'exécution d'une série de méthodes et d'interactions avec d'autres objets
- L'ensemble des états du cycle de vie d'un objet contient un **état initial**. Celui-ci correspond à l'état de l'objet juste après sa création
- Il peut également contenir **un ou plusieurs états finaux**. Ceux-ci correspondent à une phase de destruction de l'objet
- Il arrive également qu'il n'y ait pas d'état final car un objet peut ne jamais être détruit (dans le cas des services notamment)

Diagramme E/T - Etats

- Un **état** est représenté par un rectangle arrondi contenant son nom



- L'état initial est représenté par un point noir
- L'état final se représente par un point entouré d'un cercle
- Un état comporte un certain nombre de champs qui permettent de décrire les actions dans un état :
 - **entry / action** : action exécutée à l'entrée de l'état
 - **exit / action** : action exécutée à la sortie de l'état
 - **on événement / action** : action exécutée à chaque fois que l'événement cité survient
 - **do / action** : action récurrente ou significative, exécutée dans l'état

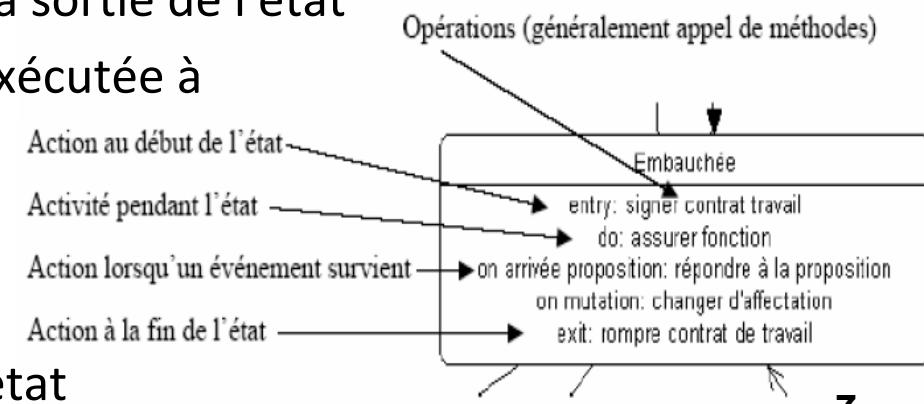


Diagramme E/T - Evénements

- Un **événement** est un fait qui déclenche le changement d'état, et fait passer un objet d'un état à un autre état
- Un événement se produit à un instant précis et est dépourvu de durée. Quand un événement est reçu, une transition peut être déclenchée et faire basculer l'objet dans un nouvel état
- L'objet recevant un événement peut réagir en changeant son état (en exécutant une opération) et/ou en émettant un autre événement
- Un objet placé dans un état donné attend l'occurrence d'un événement pour changer d'état
- Un événement n'a **pas de durée** contrairement à un état
- Un événement peut être émis par un objet du système ou provenir de l'extérieur du système (clic de souris par exemple)

Diagramme E/T - Evénements

Il existe quatre types d'événements :

1. Événement de type signal (signal event) :

Correspond à la réception d'un signal asynchrone émit par un autre objet ou par un acteur

Un événement de type signal peut être porteur de paramètres et s'écrit comme suit :

nom_événement

ou

nom_événement(paramètre1 ;paramètre2...)

ou

nom_événement(paramètre1 :type-paramètre1 ;paramètre2 :type_paramètre2...)

Différents niveaux de précision

Les signaux sont déclarés dans le diagramme de classes de la même manière qu'une classe mais en ajoutant le stéréotype « **signal** » au dessus du nom



Un signal peut aussi être déclaré de façon textuelle (sur la transition)

Diagramme E/T - Evénements

Il existe quatre types d'événements :

2. Événement appel d'opération (call event) :

Appel d'une méthode de l'objet courant par un autre objet ou par un acteur. Il possède la même syntaxe que l'événement de type signal. La méthode est aussi déclarée dans le diagramme de classe, mais à l'intérieur de la classe

3. Événement de changement (change event) :

Un événement de changement d'état est généré par la satisfaction (vrai ou faux) d'une expression booléenne sur des valeurs d'attributs. Il s'agit d'une manière déclarative d'attendre qu'une condition particulière soit satisfaite. **L'expression est testée en permanence**

La syntaxe est la suivante :

when(expression booléenne)

Diagramme E/T - Evénements

Il existe quatre types d'événements :

4. Événement temporel (time event) :

Les événements temporels sont générés par l'écoulement du temps. Ils sont spécifiés soit de manière **absolue** (date précise), soit de manière **relative** (temps écoulé). Par défaut, le temps commence à s'écouler dès l'entrée dans l'état courant

- de manière absolue (déclenchement à une date précise)**

Syntaxe : **when(date= « expression précise d'une date »)**

Exemple : when(date=17/12/2010)

- de manière relative (déclenchement après une certaine durée passée dans l'état actuel)**

Syntaxe : **after(« expression d'une durée »)**

Exemple : after(10secondes)

Diagramme E/T - Transitions

- Une **transition** définit la réponse d'un objet à l'arrivée d'un événement
- Elle représente le passage **instantané** d'un état vers un autre
- Elle indique qu'un objet qui se trouve dans un état peut «transiter» vers un autre état en exécutant éventuellement certaines activités, si un **événement déclencheur** se produit et qu'une **condition de garde** est vérifiée

Syntaxe d'une transition

- Une transition entre deux états est représentée par un trait droit fléché allant de l'état source vers l'état cible
- L'événement qui détermine le franchissement de la transition est indiqué sous forme de texte
- Si la transition est automatique, aucun événement n'est spécifié



Diagramme E/T - Transitions

- La structure de l'événement correspondant aux détails de la transition sont précisés au moyen de la syntaxe suivante :

déclencheur [garde] / effet

(chacun de ces éléments est facultatif)

déclencheur : nom de l'événement qui provoque le franchissement de la transition

garde : **condition de garde [entre crochets]**. Il est possible d'associer une condition à une transition. Pour que la transition soit franchie, il faut que la condition soit remplie en plus de la réception de l'événement associé (si celui-ci existe). C'est une expression booléenne (sur les attributs de l'objet ou les paramètres de l'événement déclencheur) qui doit être vraie pour que la transition soit déclenchée. La condition de garde est évaluée uniquement lorsque l'événement déclencheur se produit

effet : spécifie une activité à effectuer lors du déclenchement (nom de l'activité précédés de **/**). Une activité est une série d'actions. Une action consiste à affecter une valeur à un attribut, créer ou détruire un objet, effectuer une opération (lancer une méthode), envoyer un signal à un autre objet ou à soi-même, ...

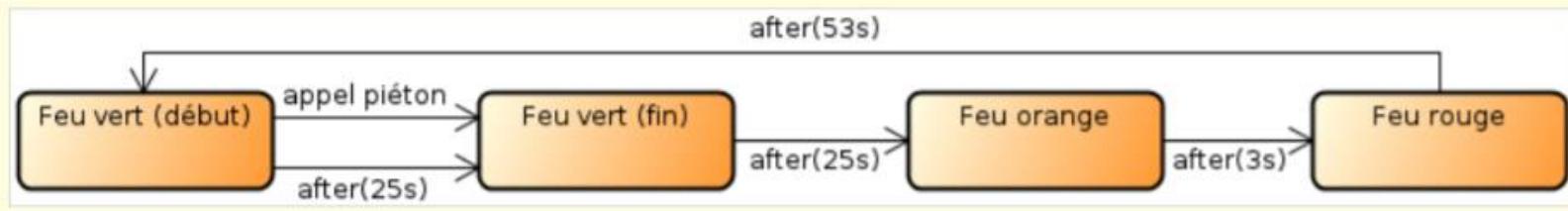
Diagramme E/T - Transitions

Remarque : Une transition sans nom d'événement est appelée transition **automatique**, elle est déclenchée lorsque l'activité de l'état source est terminée

Exemples:



Exemple 1 : Le feu tricolore (avec un bouton appel piéton).



Exemple 2 : le tourniquet (pour compter le nombre de personnes qui rentrent dans un téléphérique).

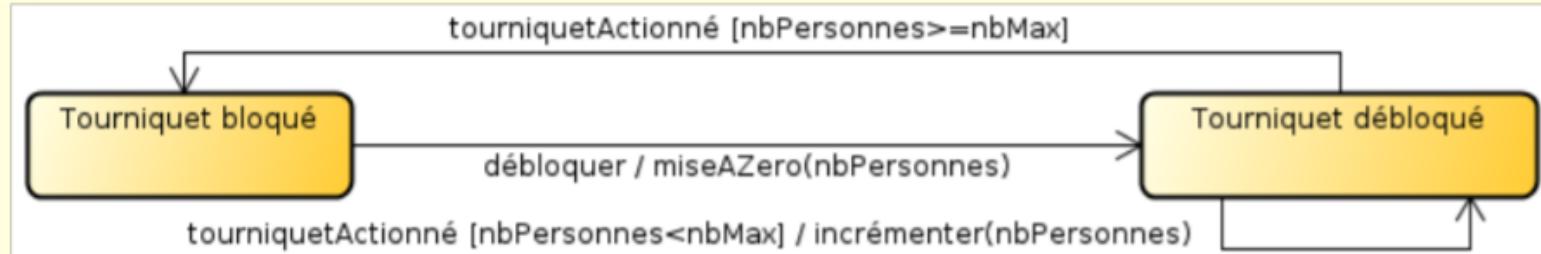


Diagramme E/T – Point de choix

1. Point de jonction :

Les points de jonction sont représentés par un cercle plein : ●

Ils permettent à plusieurs transitions d'avoir une partie commune en partageant des segments de transition

L'utilisation de points de jonction a pour but de rendre la notation des transitions alternatives plus lisible

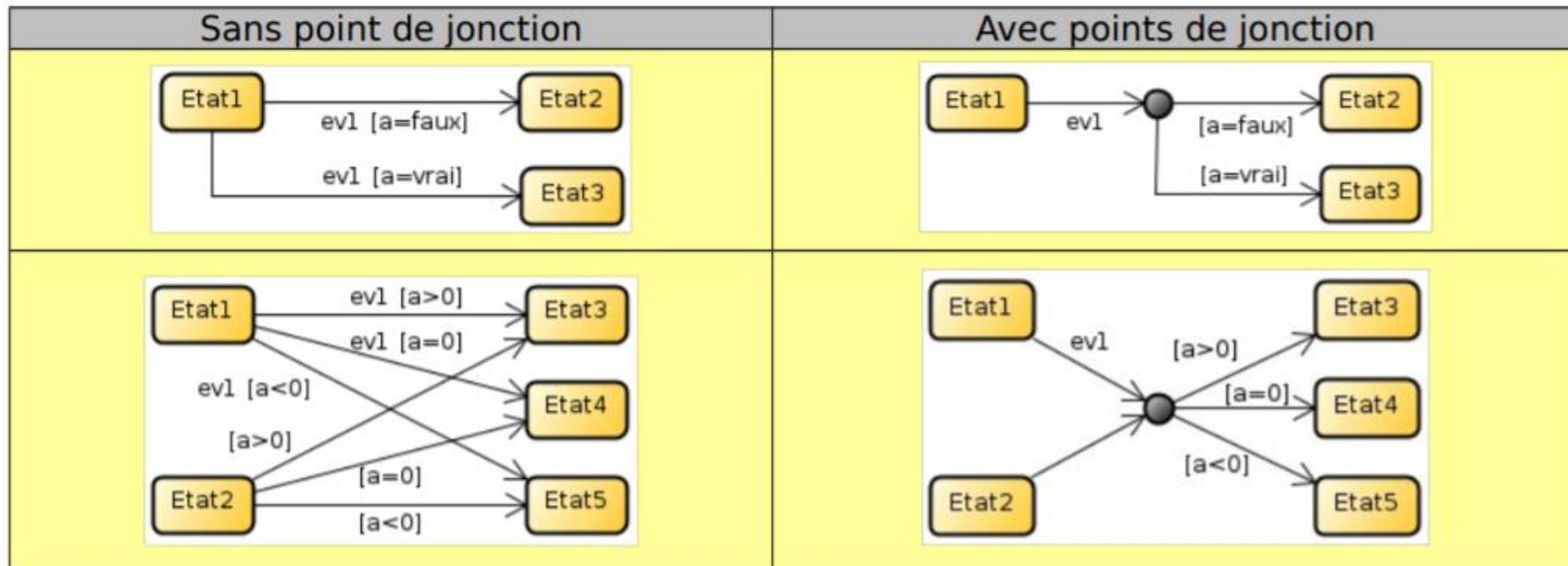


Diagramme E/T – Point de choix

2. Point de décision :

Les points de décision sont représentés par des losanges :



Le choix du segment à franchir derrière le point de décision se fait au moment de l'arrivée sur le point de décision

Lorsque nous arrivons sur un point de décision, il faut qu'un des segments de transition qui suit, soit franchissable (une transition n'ayant pas de durée, nous ne pouvons pas rester à attendre quelque chose au niveau du point de décision). Pour éviter ce problème nous pouvons ajouter un segment avec la garde [else] qui est automatiquement franchie si aucune des gardes des autres segments n'est vraie

Un segment de transition derrière un point de décision ne peut pas comporter d'événement

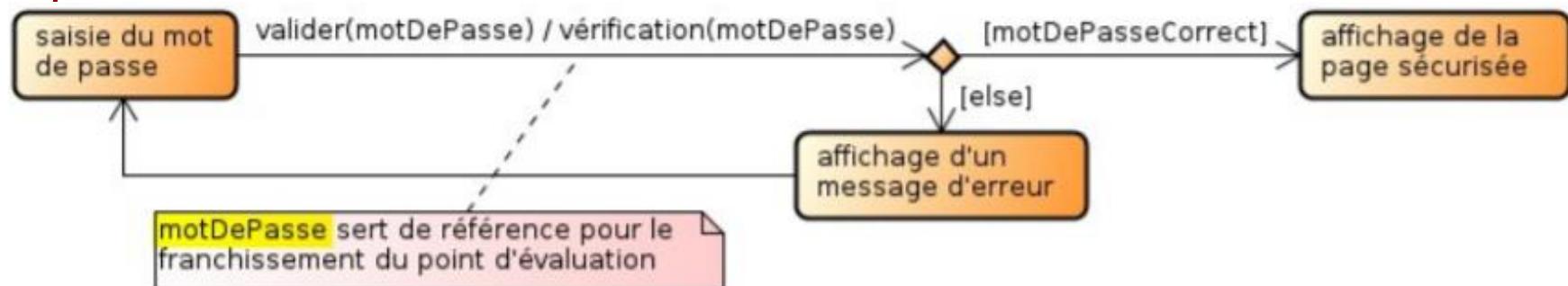
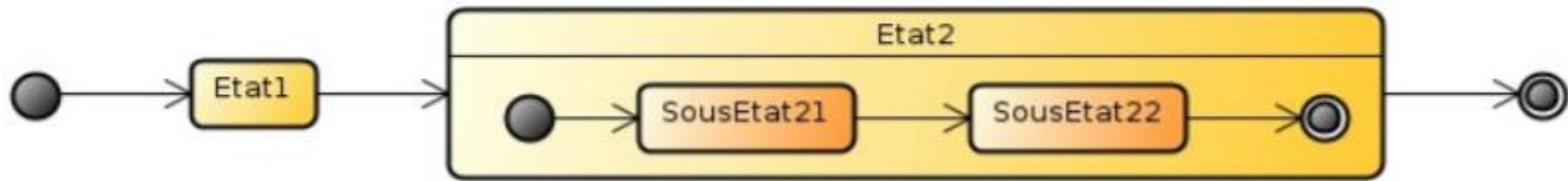


Diagramme E/T – Etat composite

Certains états sont complexes et correspondent à la réalisation de plusieurs activités (séquentielles ou simultanées) qui ne pourront pas être définis par des transitions internes. Il peut alors être intéressant de le décomposer en **sous-états**. On parle alors d'**état composite**.



- Dès qu'un état composite est actif, il active son sous état initial
- Si un état composite est raccordé à une transition automatique, elle est franchie lorsque nous atteignons le sous état final de l'état composite
- Si une des transitions externes attenantes à l'état composite est franchissable alors elle est franchie et tous les sous-états deviennent inactifs

Diagramme E/T – Etat composite



Exemple : le distributeur automatique de boisson.

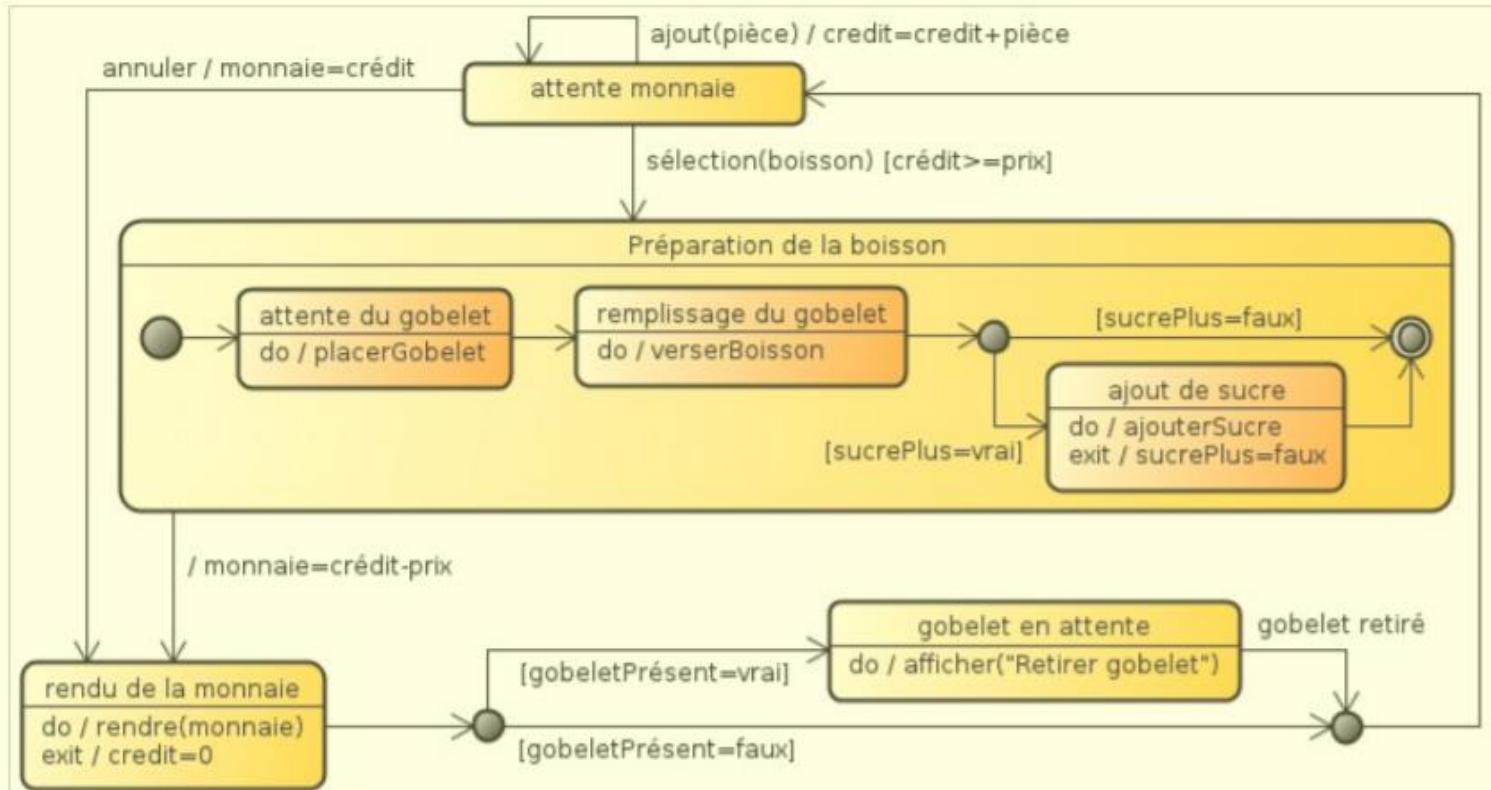


Diagramme E/T – Etat composite

Afin d'éviter de surcharger le diagramme d'états-transition, il est possible de placer le symbole (o-o) à l'intérieur de l'état composite et de spécifier ensuite son contenu ailleurs dans une autre page

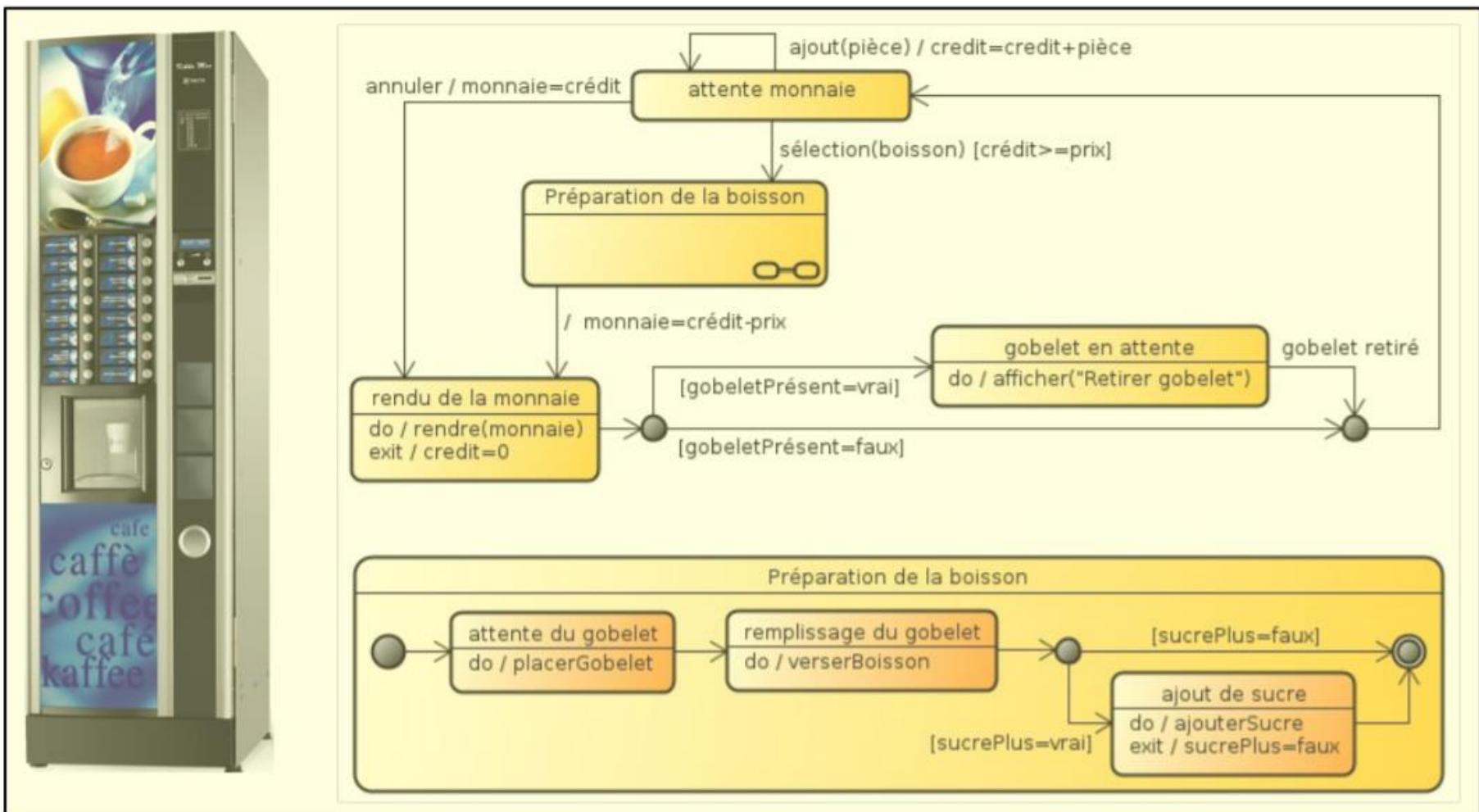


Diagramme E/T – Etat historique

Lorsqu'une transition raccordée à un état composite est franchissable alors elle est franchie même si les activités en cours (sous-états) ne sont pas terminées

Si nous désirons qu'un état composite reprenne son activité à l'endroit où il s'était interrompu lors de sa précédente activation, il faut lui définir un nouveau sous état d'entrée que nous appelons **état historique** et que nous désignons par H ou H^* :

- H pour reprendre au début du sous-état du plus haut niveau dans lequel nous nous étions arrêté
- H^* pour reprendre au début du sous état dans lequel nous nous étions arrêté, quelque soit son niveau d'imbrication (historique profond)

Diagramme E/T – Etat historique

Exemple:

Exemple : un système de lavage automatique de voiture.

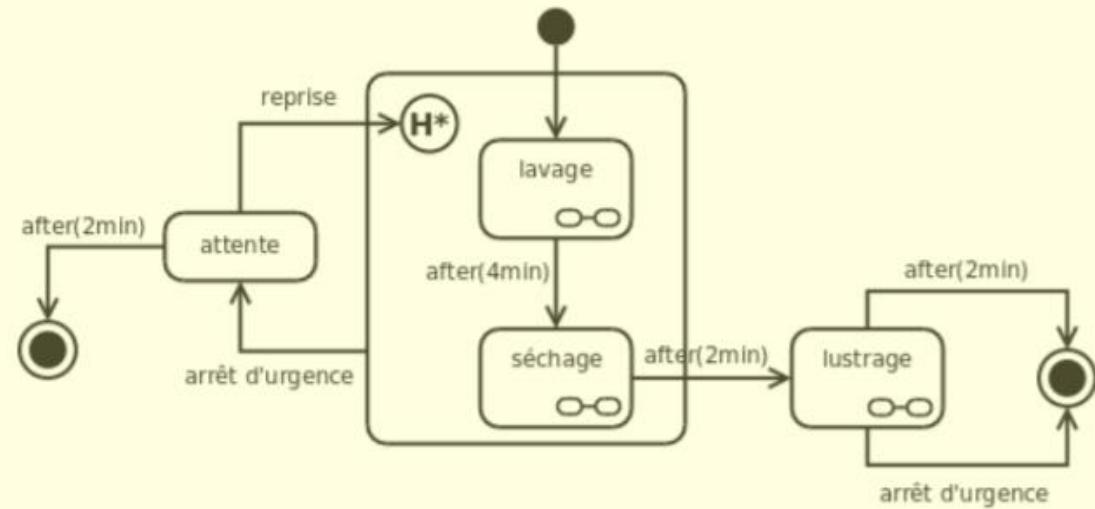
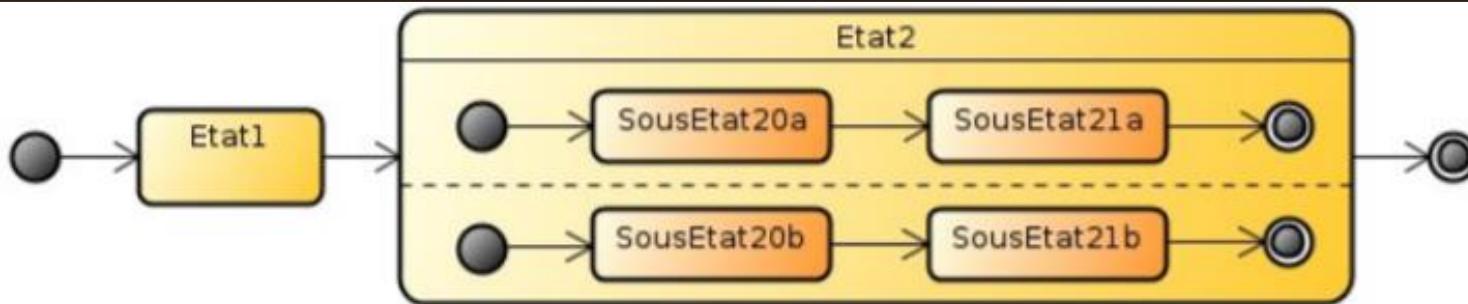


Diagramme E/T – Etat orthogonal

Les diagrammes états-transitions permettent de décrire efficacement les mécanismes concurrents grâce à l'utilisation d'**états orthogonaux**

- Un état orthogonal est un état qui possède plusieurs régions (séparées par des pointillés horizontaux) qui évoluent simultanément et en parallèles
- Chaque région représente un flot d'exécution, elle peut posséder un état initial et un état final

Une transition qui atteint la bordure d'un état composite orthogonal est équivalente à une transition qui atteint les états initiaux de toutes ses régions concurrentes. Toutes les régions concurrentes d'un état composite orthogonal doivent attendre leur état final pour que l'état composite soit considéré comme terminé.



- Lorsque l'état2 est activé, l'activation des états initiaux de chaque région est exécutée
- Si la transition externe qui permet de sortir de l'état2 est une transition automatique, elle est franchie uniquement lorsque toutes les régions ont leur état final actif

Diagramme E/T – Etat orthogonal



Exemple : Dans l'exemple du distributeur automatique de boisson que nous avons vu précédemment, si nous considérons que nous pouvons rendre la monnaie en même que se prépare la boisson, nous obtenons le diagramme d'états-transitions suivant :

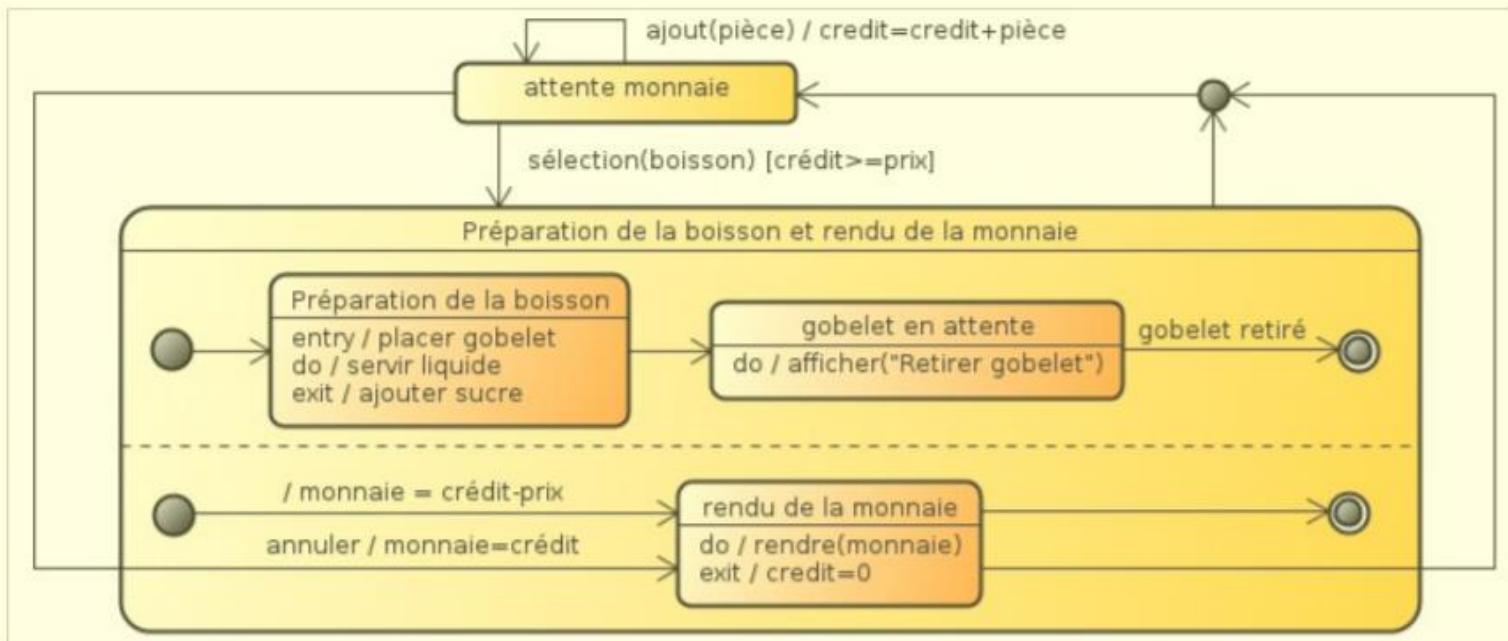
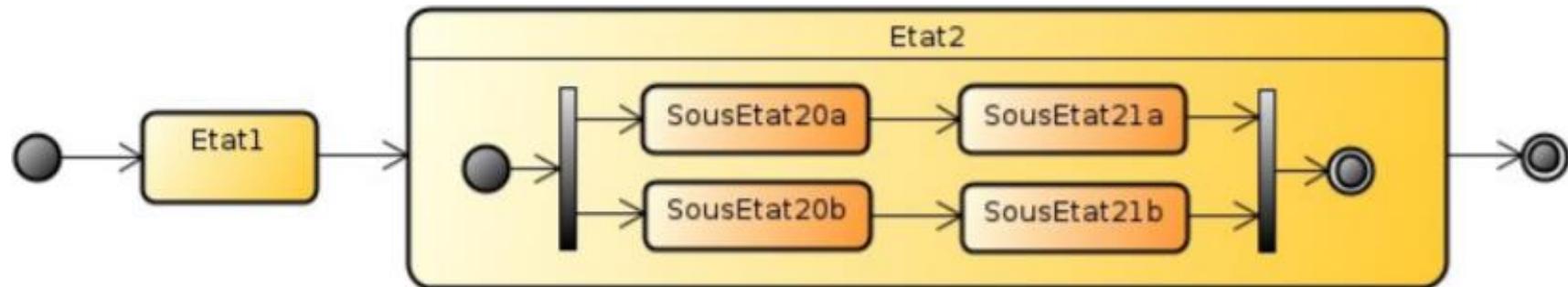
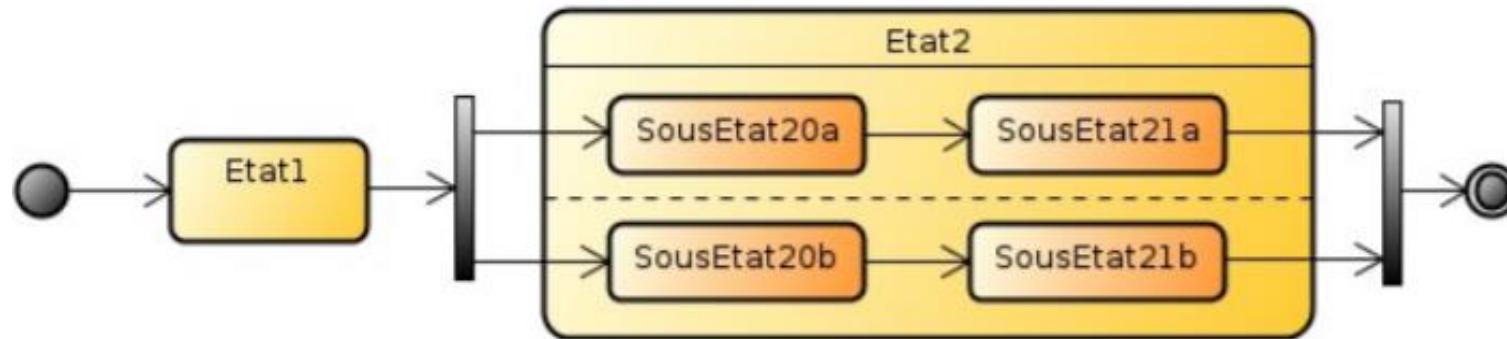


Diagramme E/T – Etat orthogonal

Le tracé précédent est très clair, mais il est aussi possible de représenter des flots d'exécutions parallèles en utilisant la notation des transitions concurrentes (**barres épaisses**) :



Comment construire un diagramme E/T?

- Choisir une classe sur le diagramme des classes
- Identifier tous les messages qui arrivent sur un objet correspondant dans le diagramme de séquence afin de trouver les transitions
- Dressez une liste d'états possibles en pensant à la vie de l'objet (comment est-il créé, comment et à quel moment est-il supprimé, l'objet possède-t-il des états actifs, inactifs, y a-t-il des états où l'objet attend ?)
- Construisez des fragments du diagramme d'états avec des chemins possibles entre états
- Révisez tous les chemins en identifiant les chemins concurrents; jumelez différents états et demandez-vous si l'objet peut être dans ces deux états en même temps, ou si l'un suit l'autre naturellement
- Développez chaque transition et chaque état
- Réfléchissez à la possibilité q'un état puisse inclure des sous-états
- Révisez votre diagramme

5. Diagramme de séquence

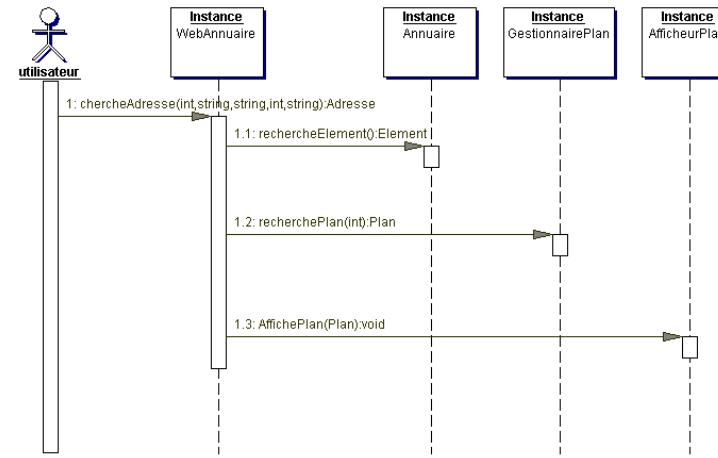


Diagramme de séquence

- Les diagrammes de séquences permettent de représenter des interactions entre instances (objets)
- Ils permettent de décrire **COMMENT** les éléments du système interagissent entre eux et avec les acteurs
- Comme il représente la dynamique du système, le diagramme de séquence fait entrer en action les instances de classes intervenant dans la réalisation d'un cas d'utilisation particulier

*A moins que le système à modéliser soit extrêmement simple, nous ne pouvons pas modéliser la dynamique globale du système dans un seul diagramme. Nous ferons donc appel à un ensemble de diagrammes de séquences chacun correspondant à une sous fonction du système (**généralement utilisé pour illustrer un cas d'utilisation**)*

Diagramme de séquence

Représentation du diagramme de séquence :

1. L'objet (instance) :

Dans un diagramme de séquence, l'objet à la même représentation que dans le diagramme des objets. C'est-à-dire un rectangle dans lequel figure le nom de l'objet. Le nom de l'objet peut prendre l'une des formes suivantes :

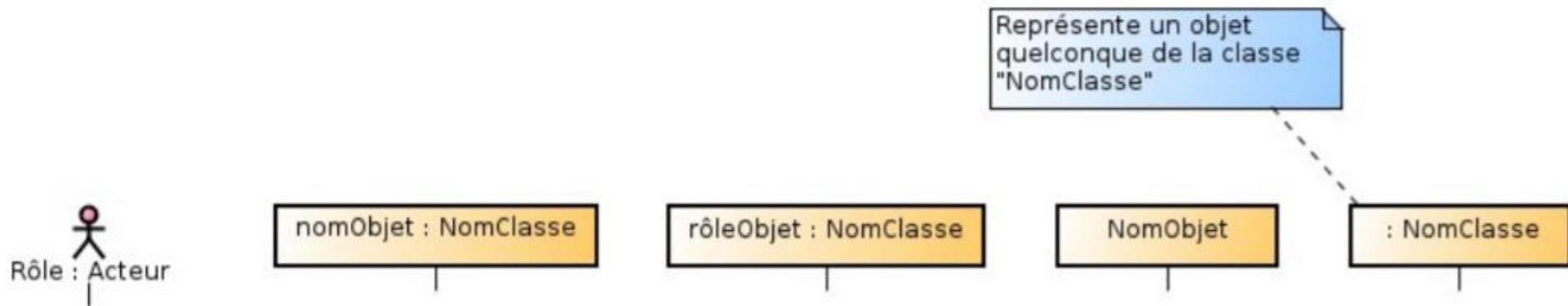


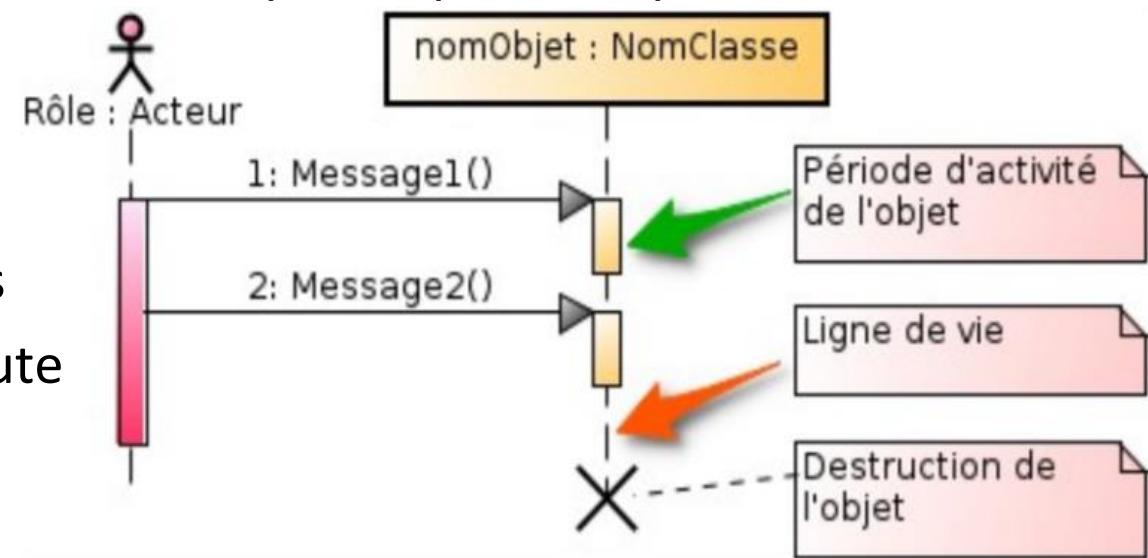
Diagramme de séquence

Représentation du diagramme de séquence :

2. La ligne de vie :

A chaque objet est associé une **ligne de vie** (en trait pointillés à la verticale de l'objet) qui peut être considéré comme un axe temporel (le temps s'écoule du haut vers le bas). Dans ce genre de diagramme, la quantification du temps n'a pas d'importance

- La ligne de vie indique les périodes d'activité de l'objet (généralement, les moments où l'objet exécute une de ces méthodes)



- Lorsque l'objet est détruit, la ligne de vie s'achève par une croix

Diagramme de séquence

Représentation du diagramme de séquence :

3. Les messages

- Les principales informations contenues dans un diagramme de séquence sont les **messages** échangés entre les lignes de vie :
 - Ils sont représentés par des flèches
 - Ils sont présentés du haut vers le bas le long des lignes de vie, dans un ordre chronologique
- La réception des messages provoque une période d'activité (rectangle vertical sur la ligne de vie) marquant le traitement du message (spécification d'exécution dans le cas d'un appel de méthode)

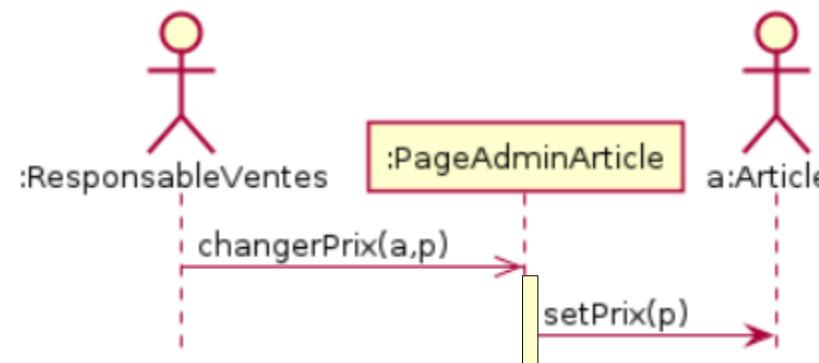


Diagramme de séquence

Représentation du diagramme de séquence :

3. Les messages

- Un message définit une **communication particulière** entre des lignes de vie. 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)
- Plusieurs types de messages existent, les plus communs sont :
 - **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)
 - **La création ou la destruction** d'une instance de classe

Diagramme de séquence

Représentation du diagramme de séquence :

3.1. Les messages synchrone

- La réception d'un **message synchrone** doit provoquer chez le destinataire le lancement d'une de ses méthodes (qui souvent porte le même nom que le message)
- Un message **synchrone** bloque l'expéditeur jusqu'à la réponse du destinataire : l'expéditeur du message reste bloqué pendant toute l'exécution de la méthode et attend donc la fin de celle-ci avant de pouvoir lancer un nouveau message. C'est le message le plus fréquemment utilisé

Si un objet A envoie un message synchrone à un objet B, A reste bloqué tant que B n'a pas terminé

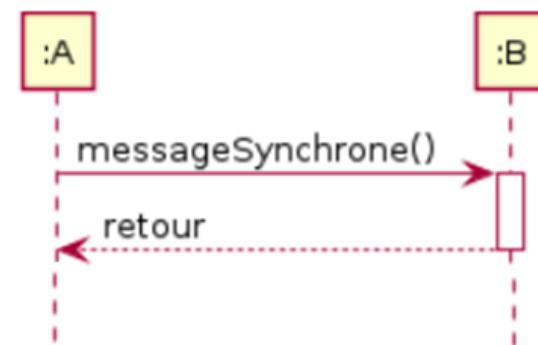


Diagramme de séquence

Représentation du diagramme de séquence :

3.2. Les messages asynchrones

- Dans le cas d'un **message asynchrone**, l'expéditeur n'attend pas la fin de l'activation de la méthode invoquée chez le destinataire
- Un message **asynchrone** peut être :
 - Un appel de méthode : Fréquent dans un système multi-threads (multi-tâche). Ainsi, l'objet expéditeur n'étant pas bloqué pendant l'exécution de la méthode, il peut continuer à envoyer d'autres messages
 - Un signal (cas le plus fréquent) : L'objet expéditeur transmet juste une information à l'objet destinataire. Souvent, ce sont les acteurs ou les périphériques qui envoient des signaux

Si un objet A envoie un message asynchrone à un objet B, A n'est pas bloqué et peut continuer sans attendre un retour de B

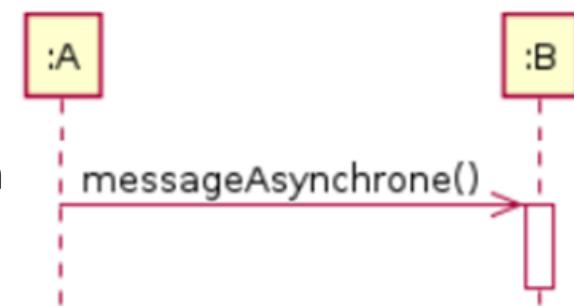


Diagramme de séquence

Représentation du diagramme de séquence :

Remarques sur les messages

- Si une méthode qui a été activée (par un message) doit retourner des valeurs à la fin de son activation, cela se fait par un message retour
- Le message de retour n'est donc pas un appel de méthode (il ne provoque donc pas l'activation d'un objet)
- Le message retour porte souvent le nom de l'élément retourné

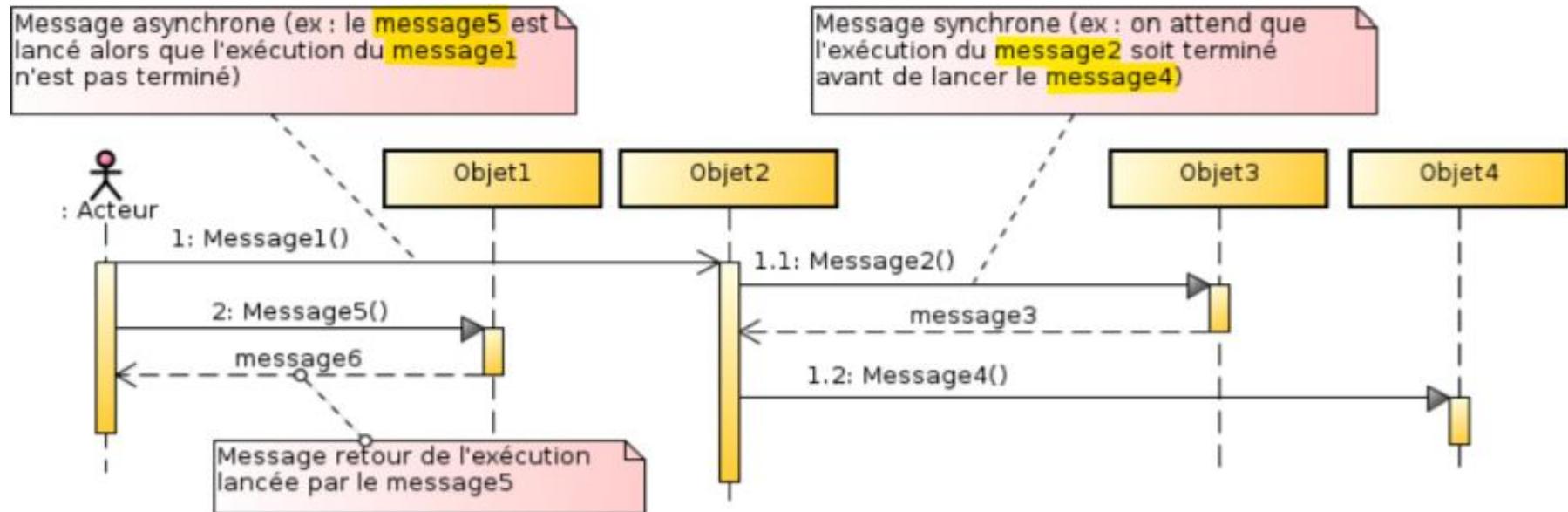


Diagramme de séquence

Représentation du diagramme de séquence :

4. Crédit et destruction d'objets : Une séquence peut aussi contenir la création ou la destruction d'un objet

- La création d'un objet est matérialisée par un message spécifique, appel d'un constructeur, généralement accompagné du stéréotype «*create*» qui pointe sur le début (le sommet) 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. Souvent l'objet est détruit suite à la réception d'un message mais ce n'est pas obligatoire. Dans ce cas là, il porte le stéréotype «*destroy*»

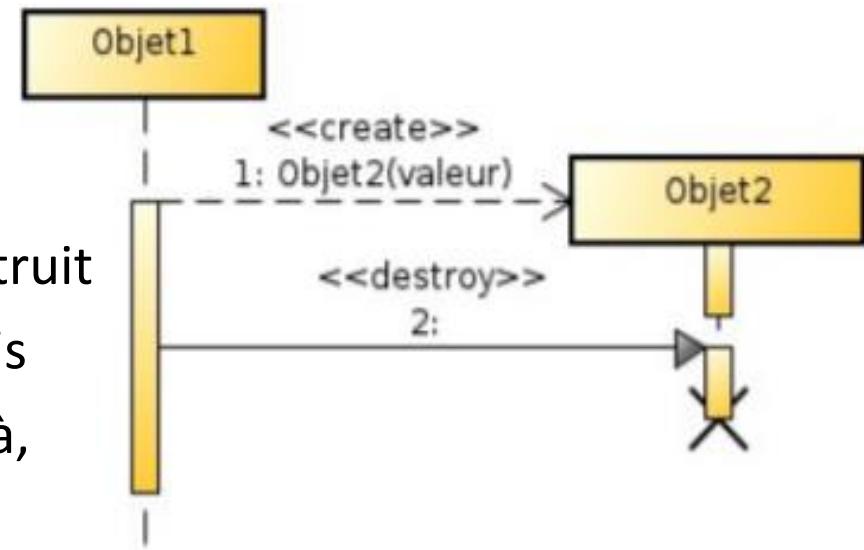


Diagramme de séquence

Représentation du diagramme de séquence :

4. Syntaxe des messages synchrones et asynchrones

Dans la plupart des cas, la réception d'un message est suivie de l'exécution de la méthode de la classe cible. Cette méthode peut recevoir des arguments et la syntaxe des messages permet de transmettre ces arguments

Dans un diagramme de séquence, nous pouvons nous contenter de définir un message par :

- **Son nom** (qui est le nom de la méthode appelée ou du signal envoyé)
- Nous pouvons lui adjoindre facultativement :
 - **Une numérotation** devant le nom du message (séparée 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)

Diagramme de séquence

Représentation du diagramme de séquence :

4. Syntaxe des messages synchrones et asynchrones

Exemple : installation communication téléphonique

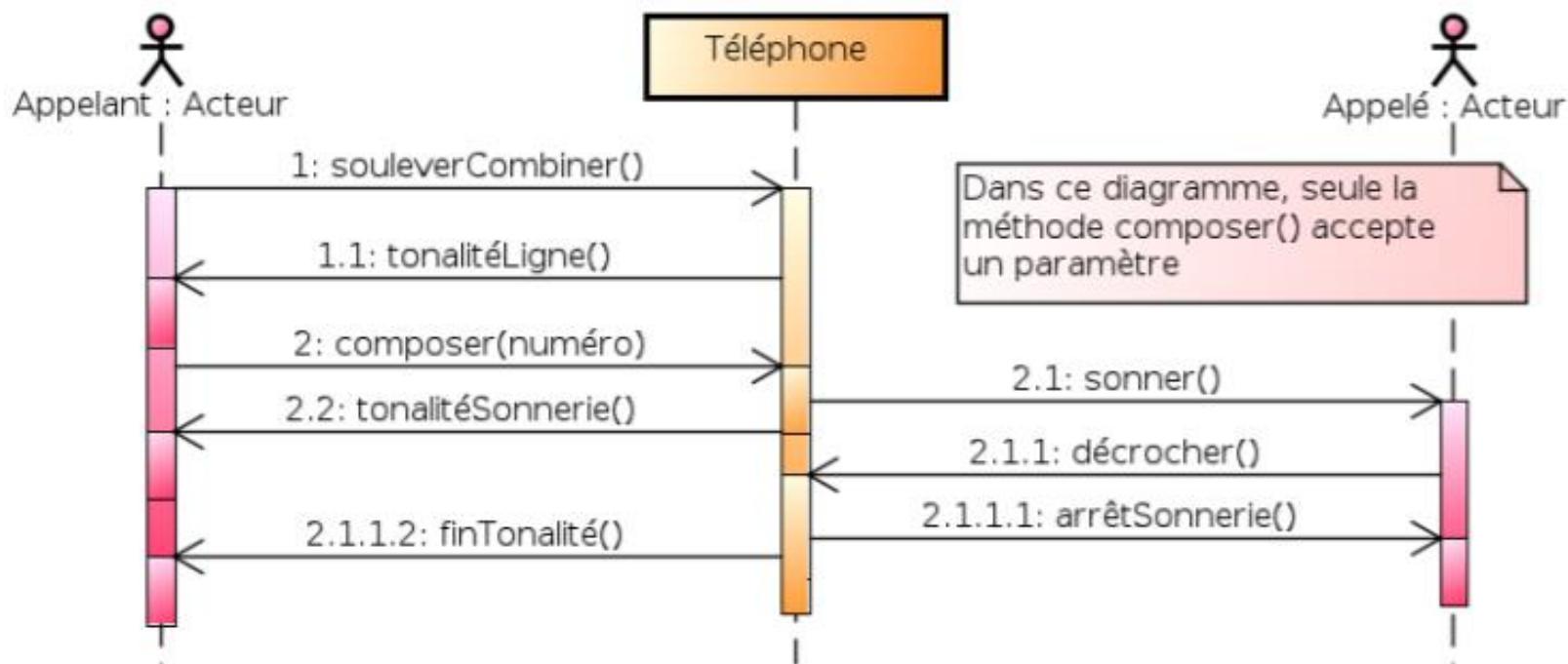


Diagramme de séquence

Représentation du diagramme de séquence :

5. Syntaxe des réponses (messages retour) : Comme pour les messages synchrones ou asynchrones, nous pouvons nous contenter de donner au message retour un simple **nom**, mais nous pouvons aussi les caractériser plus précisément en utilisant la syntaxe suivante :

numéro : attribut = nomMessage (paramètres) : valeurDeRetour

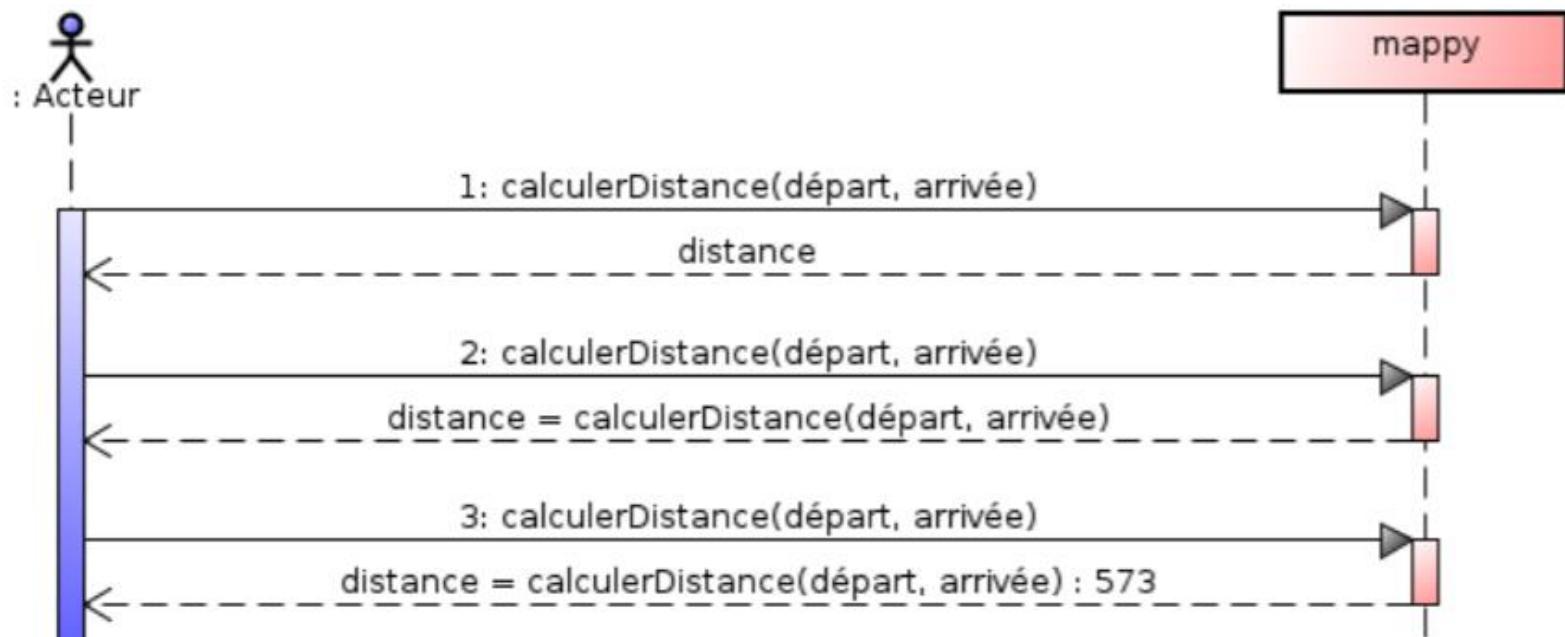


Diagramme de séquence

Représentation du diagramme de séquence :

6. Messages retours implicites et explicites : Le retour d'un message synchrone peut ne pas être représenté, le retour est alors **implicite**. Par contre, dans le cas d'un message asynchrone, il est impératif de faire apparaître le message de retour. Le retour est **explicite**

- Si l'exécution de la méthode lancée par le message **asynchrone** ne doit rien retourner, nous ne devons pas représenter le message de retour
- Le retour dans un message **synchrone** peut ne pas être implicite et ne sera pas modélisé. Les 2 diagrammes suivants sont équivalents :

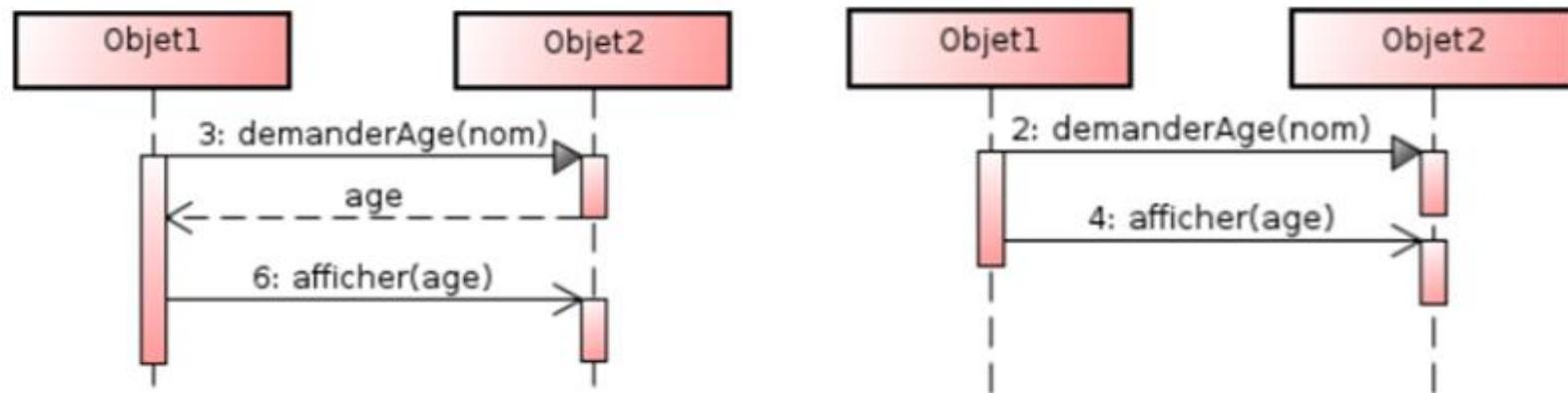


Diagramme de séquence

Représentation du diagramme de séquence :

7. Recouvrement des bandes d'activations : Lorsqu'un objet est déjà activé il peut quand même recevoir d'autres messages (appel d'une autre de ses méthodes), cela se représente par un dédoublement de la bande d'activation

8. Messages récursifs : Un objet peut s'envoyer un message à lui-même (utilisation d'une autre méthode du même objet). Cela se représente là aussi par un dédoublement de la bande d'activation

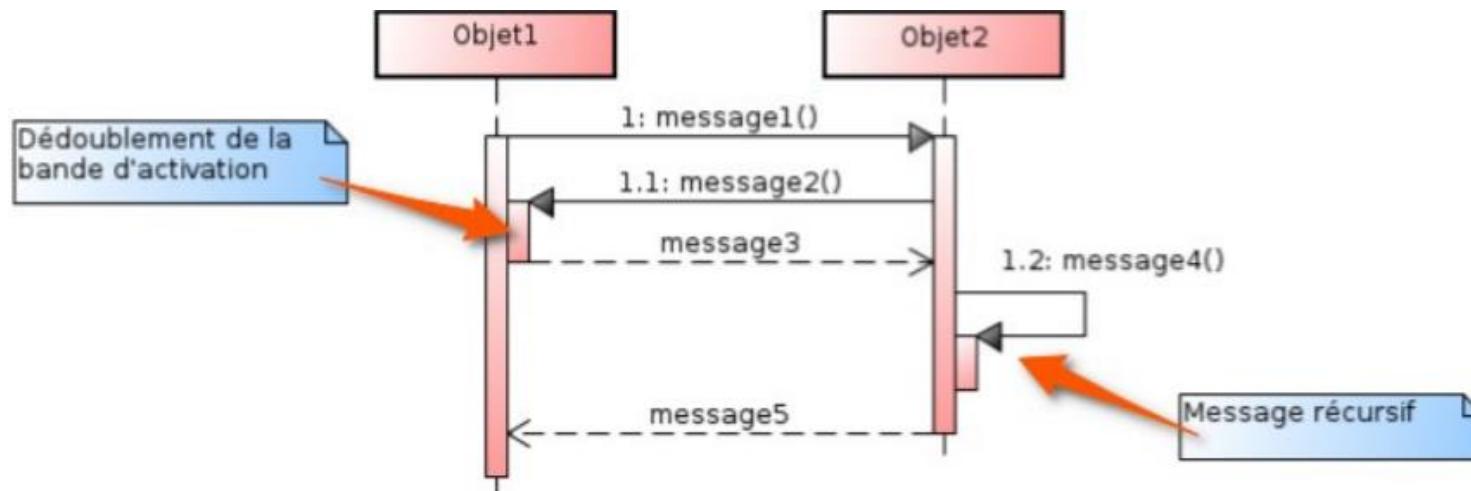
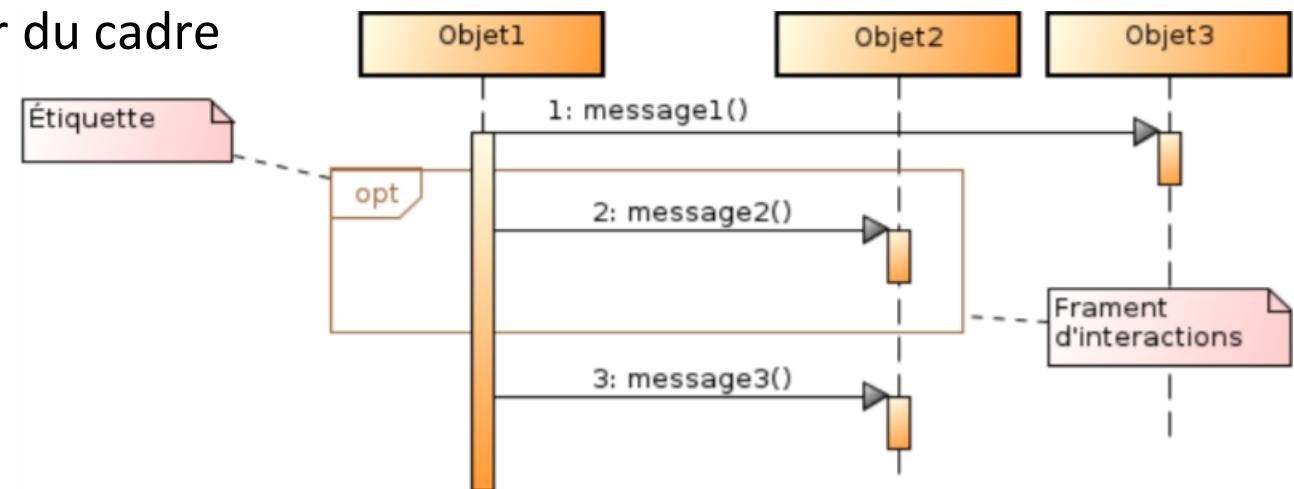


Diagramme de séquence

Représentation du diagramme de séquence :

8. Fragments d'interactions combinés : Un fragment d'interactions 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



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, ..

Diagramme de séquence

Représentation du diagramme de séquence :

8.1. Fragment d'interaction avec opérateur « opt » : L'opérateur option (**opt**) comporte un opérande et une condition de garde associée. Le sous-fragment s'exécute si la condition de garde est vraie et ne s'exécute pas dans le cas contraire

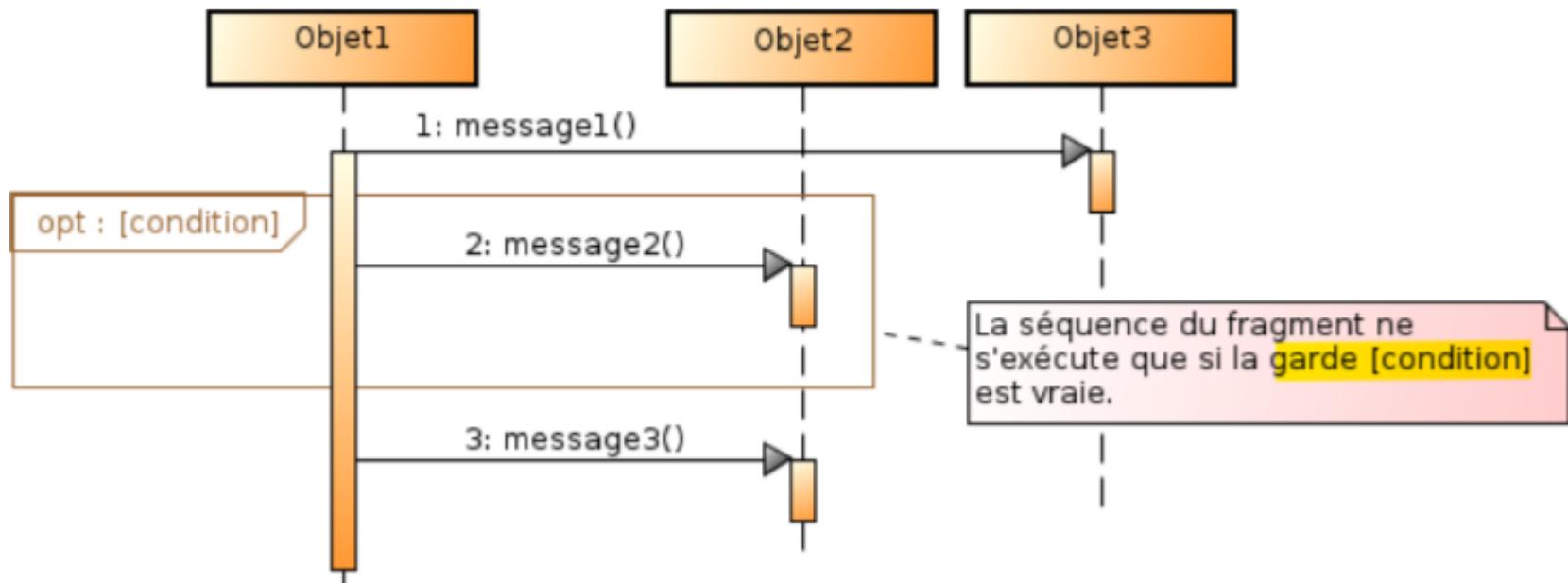


Diagramme de séquence

Représentation du diagramme de séquence :

8.2. Fragment d'interaction avec opérateur « alt » : L'opérateur alternatives (**alt**) est un opérateur conditionnel possédant plusieurs opérandes séparés par des pointillés. C'est l'équivalent d'une exécution à choix multiples. **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

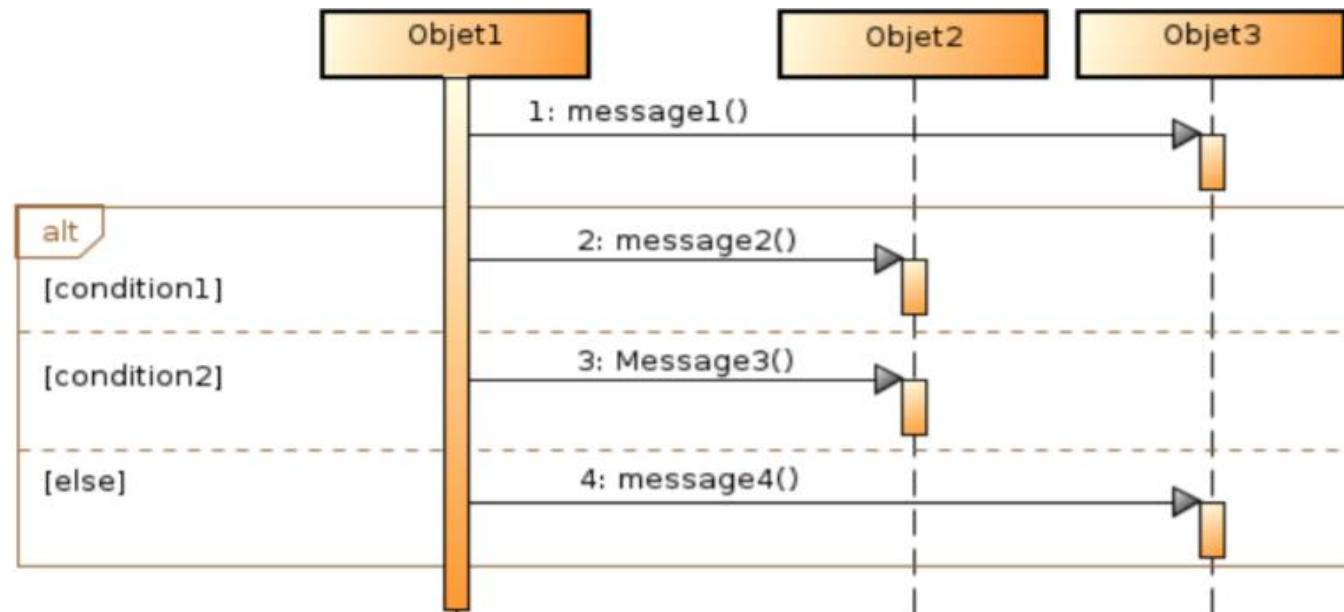


Diagramme de séquence

Représentation du diagramme de séquence :

8.3. Fragment d'interaction avec opérateur « loop » : L'opérateur de boucle (**loop**) exécute une itérative dont la séquence qu'elle contient est exécutée tant que la garde qui lui est associée est vraie

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** est vrai et que le nombre d'exécution de la boucle ne dépasse pas **max** fois

Exemple de gardes :

loop[3] → La séquence s'exécute 3 fois

loop[1, 3, code=faux] La séquence s'exécute 1 fois puis un maximum de 2 autres fois si code=faux

Diagramme de séquence

Représentation du diagramme de séquence :

8.3. Fragment d'interaction avec opérateur « loop » :

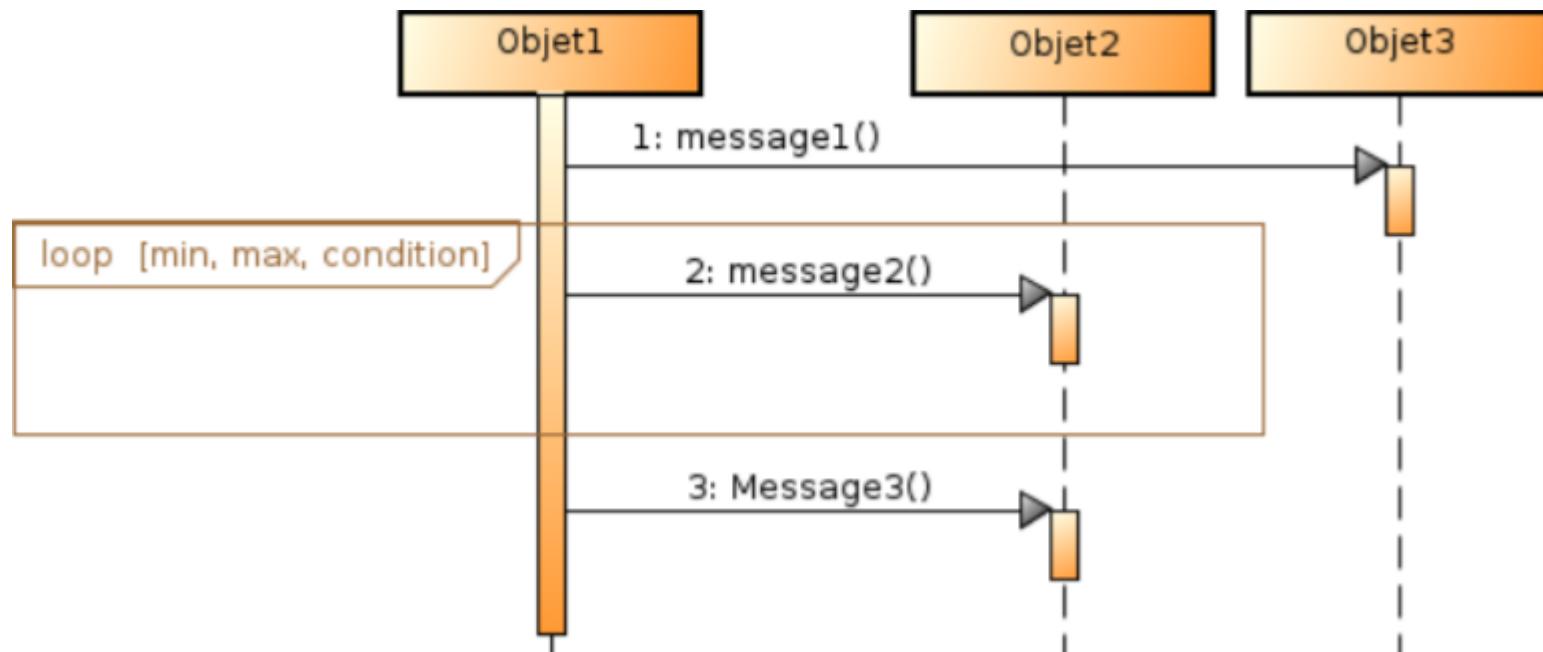


Diagramme de séquence

Représentation du diagramme de séquence :

8.4. 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)

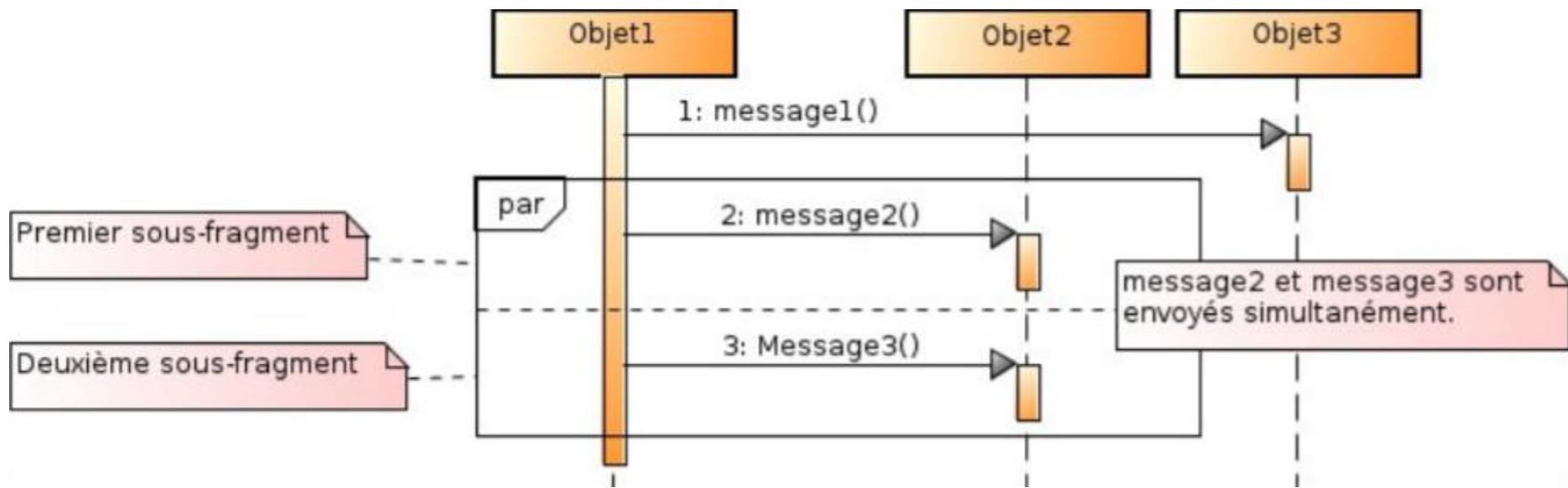


Diagramme de séquence

Représentation du diagramme de séquence :

8.5. Autres fragments d'interactions:

- **ignore** et **considere** : pour les fragments facultatifs ou obligatoires
- **critical** : pour les fragments qui doivent se dérouler sans être interrompus
- **break** : pour les fragments représentants des scenarii exceptionnels ou de ruptures (ex appui sur la touche « Esc »). Le scénario de rupture est exécuté si une condition de garde est satisfaite
- **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 »)
- **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)
- **strict** : pour les fragments dont les messages doivent se dérouler dans un ordre bien précis
- **neg** : pour indiquer que la séquence à l'intérieur du fragment n'est pas valide
- **ref** : permet de faire appel à un autre diagramme de séquence

Diagramme de séquence

Exemple d'un diagramme de séquence :

SA1: Le code est erroné pour la première ou la deuxième fois

SA2: Le montant demandé est trop élevé

SA3: Le ticket est refusé

SE1: Carte non valide

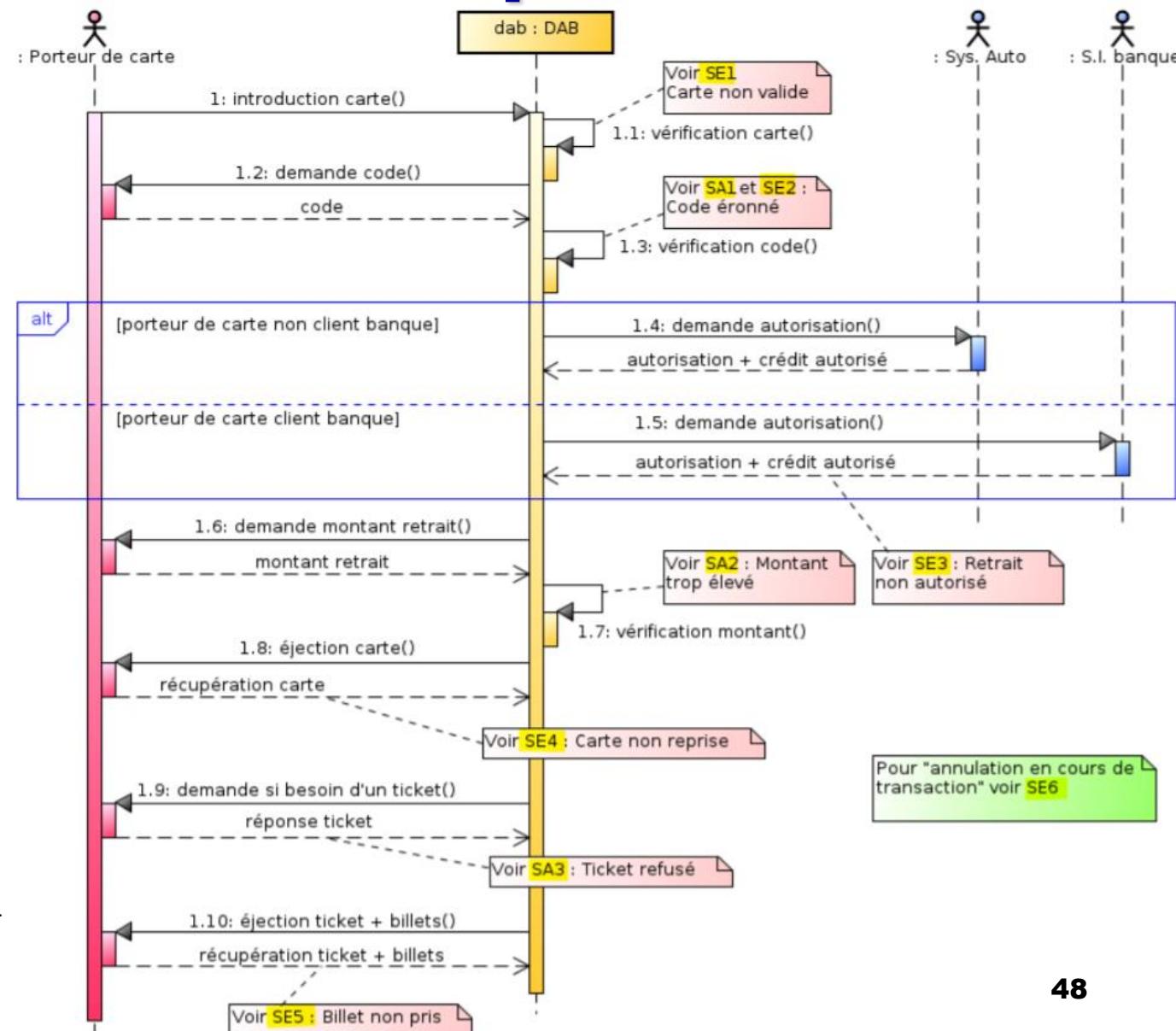
SE2: Le code est erroné pour la troisième fois

SE3: Retrait non autorisé

SE4: Carte non reprise

SE5: Billets non pris

SE6: Annulation de la transaction



6. Diagramme de collaboration

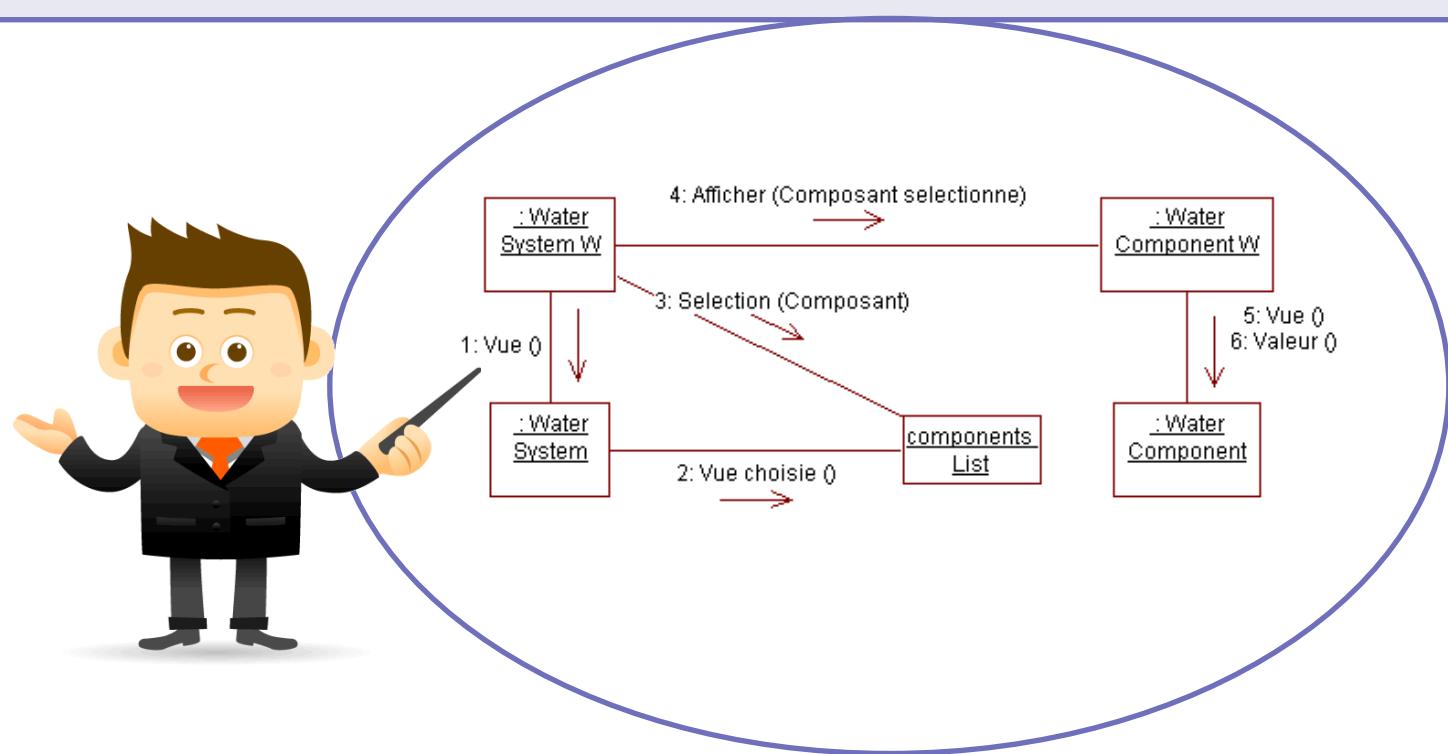


Diagramme de Collaboration

- Un diagramme de **collaboration** (*communication*) représente la vue statique et la vue dynamique d'un ensemble d'éléments
- Les diagrammes de collaboration montrent des interactions entre objets (instances de classes) et acteurs
- Ils permettent de représenter le contexte d'une interaction, car on peut y préciser les états des objets qui interagissent

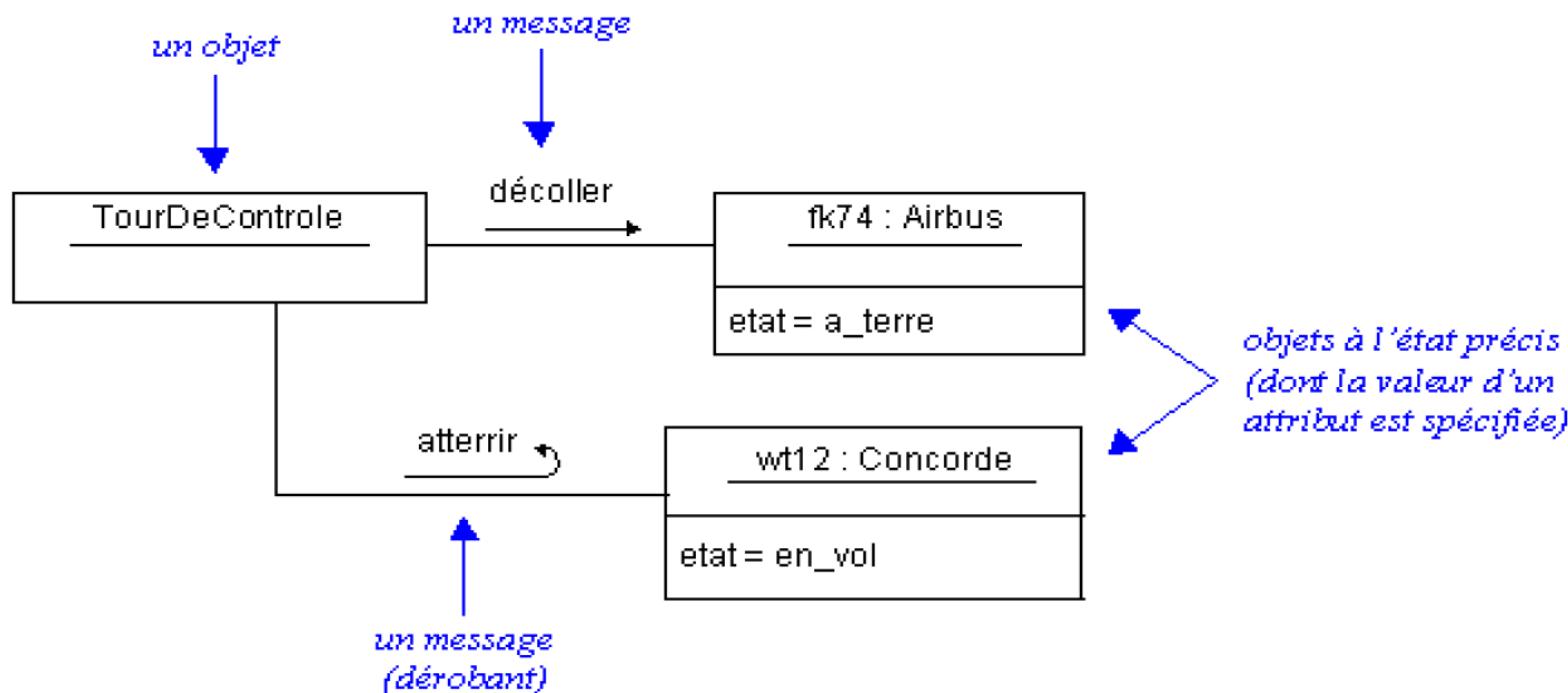


Diagramme de Collaboration

- Un diagramme de collaboration représente une collaboration, c'est-à-dire un ensemble de **rôles** d'objet liés dans un contexte particulier, et une **interaction**, qui représente l'ensemble des **messages** échangés entre les objets pour effectuer une opération ou produire un résultat
- Il s'agit d'un ***diagramme d'interaction*** qui montre les interactions et les liens entre objets
 - Il affiche pour les cas d'utilisation la structure de collaboration entre objets
 - Il représente le contexte d'une interaction, car on peut y préciser **les états** des objets qui interagissent
- Ce diagramme est équivalent au diagramme de séquences. Cependant, l'aspect temporel n'apparaît pas, mais l'aspect chronologique est présent. La chronologie des interactions est indiquée par la numérotation de messages pour indiquer leur ordre d'envoi

Diagramme de Collaboration

Composants du diagramme de collaboration :



- Les **objets** (instances de classe)
- Les **acteurs**
- Les **liens** entre objets (comme dans le diagramme de classe) ↗
- Les **interaction** entre objets sous forme d'une suite de **messages** ↗

Diagramme de Collaboration

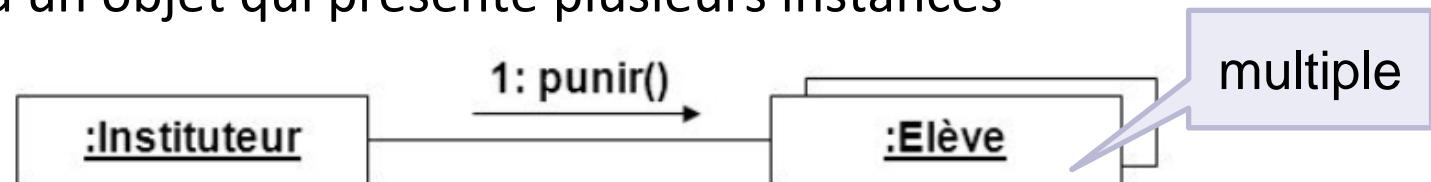
Objet: Un objet est une instance d'une classe

- Chaque objet est décrit par :

Nom-objet :classificateur ou **nom-objet :nom-classe**

- Un **multiple** définit un jeu d'instances. Il s'agit d'une représentation graphique d'un objet qui présente plusieurs instances

Exemple :



Acteurs: La définition d'un acteur dans le contexte d'un diagramme de collaboration est la même que celle dans les diagrammes de cas d'utilisation ou de séquence. Elle spécifie un utilisateur externe, ou un ensemble d'utilisateurs liés qui interagissent avec un système

Diagramme de Collaboration

Liens : Un lien entre objets représente une connexion entre deux objets, il permet de modéliser la collaboration entre objets, d'où le nom de diagramme de collaboration

- Il est représenté sous forme d'un trait plein entre :
 - Deux objets
 - Un objet et un acteur (ou vice-versa)
- Un lien entre objets peut être une instance d'association entre classes



Diagramme de Collaboration

Message : Dans un diagramme de collaboration il est possible de spécifier de manière très précise l'ordre et les conditions d'envoie des messages

Pour chaque message, il est possible d'indiquer :

1. son numéro d'ordre qui indique le rang du message par rapport aux autres messages

- Les messages sont numérotés à l'aide de chiffres séparés par des points. Ainsi, il est possible de représenter le niveau d'emboîtement des messages et leur précédence
- **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

Diagramme de Collaboration

Message :

- 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

2. le prédicat qui conditionne son envoi, spécifiée sous forme d'expression booléenne ?

- la liste des messages prédecesseurs, composée d'une liste de numéros d'ordre suivie de '/', la liste des prédecesseurs définit quels messages doivent être échangés avant que le message courant ne puisse être envoyé
- **Exemple** : numéros de séquence 1, 2, 4 avant 3 = '1,2,4/ 3' ?

Diagramme de Collaboration

Message :

3. Heure de début / Heure de fin, heures définies par l'utilisateur et utilisées pour définir des contraintes

□ **Exemple** : [heure = midi] 1 : manger() Ce message n'est envoyé que s'il est midi

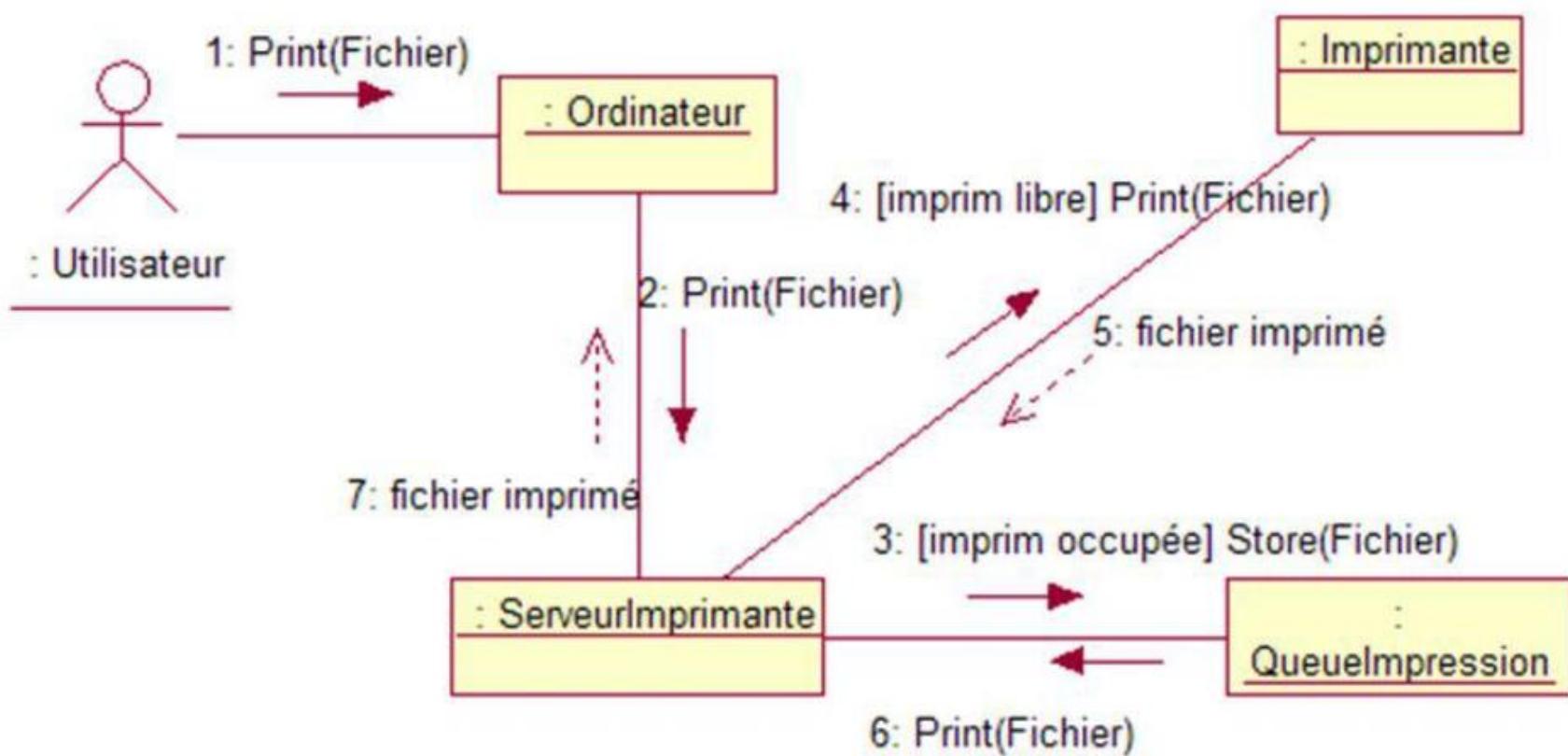
4. récurrence du message: Permet de spécifier l'envoi répétitif de messages en mode séquentiel (ou en parallèle, avec "||")

□ **Exemple1** : 1..3..6*:ouvrir() Ce message est envoyé de manière séquentielle un certain nombre de fois

□ **Exemple2** : 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

Diagramme de Collaboration

Exemple d'un diagramme de collaboration :



7. Diagramme d'activités

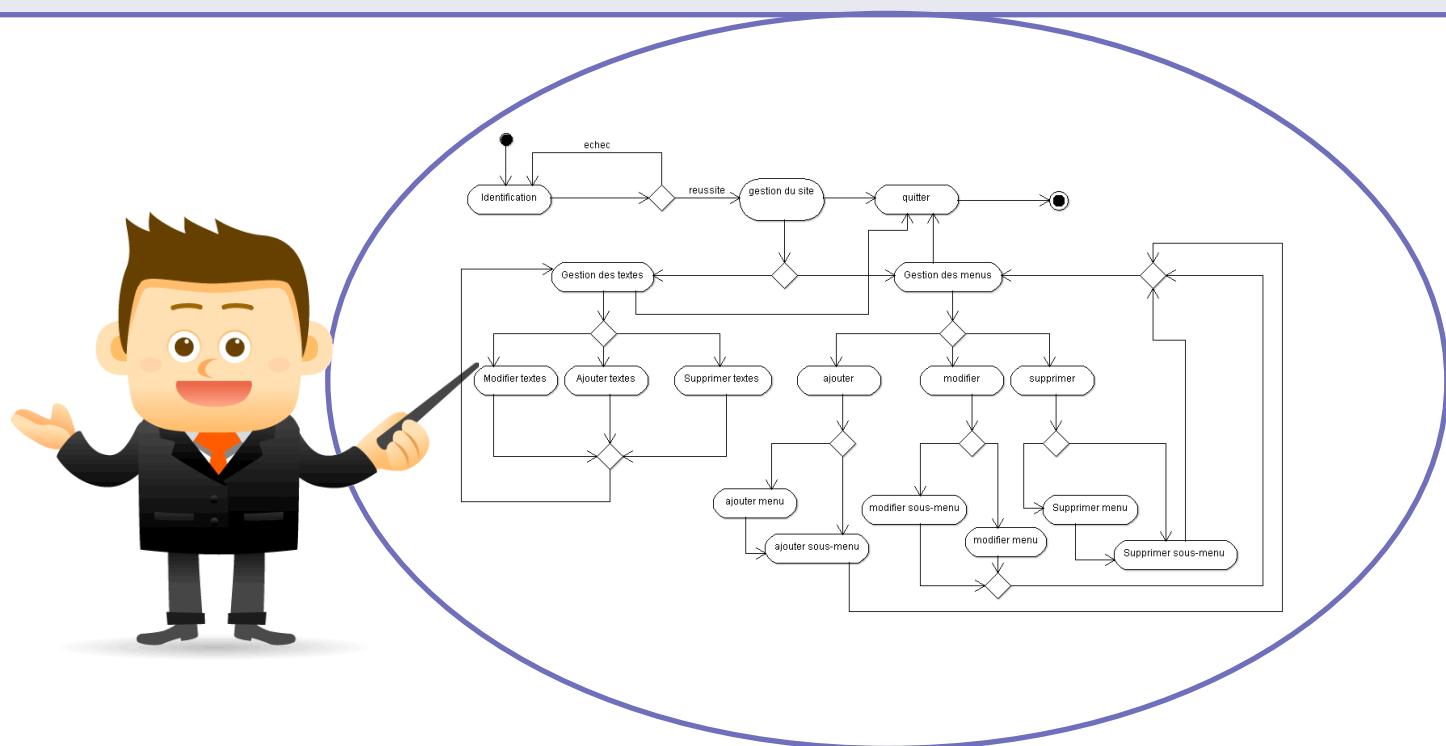


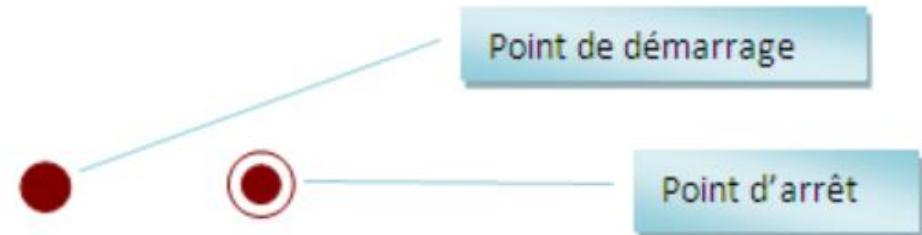
Diagramme d'Activités

- Un **diagramme d'activité** représente la vue dynamique d'un ensemble d'éléments sous forme de flux d'exécution
- Une activité représente une exécution d'un mécanisme, un déroulement d'étapes séquentielles
- Le passage d'une activité vers une autre est matérialisé par une transition
- Les transitions sont déclenchées par la fin d'une activité et provoquent le début immédiat d'une autre (elles sont **automatiques**)
- Les mécanismes dynamiques peuvent être décrits par un diagramme d'activités
 - seuls les mécanismes complexes ou intéressants méritent d'être représentés
- Peut servir à décrire le déroulement d'un cas d'utilisation
- Une variante des diagrammes d'états-transitions

Diagramme d'Activités

Point de démarrage et d'arrêt :

- Le diagramme est composé d'un point de démarrage et d'un point d'arrêt



Les actions et les transitions :

- Le diagramme d'activité est une formalisation graphique des actions qui sont réalisées dans un cas d'utilisation
- Il est organisé en actions réalisées soit par un acteur, soit par le système, reliées par une flèche indiquant l'enchaînement des actions

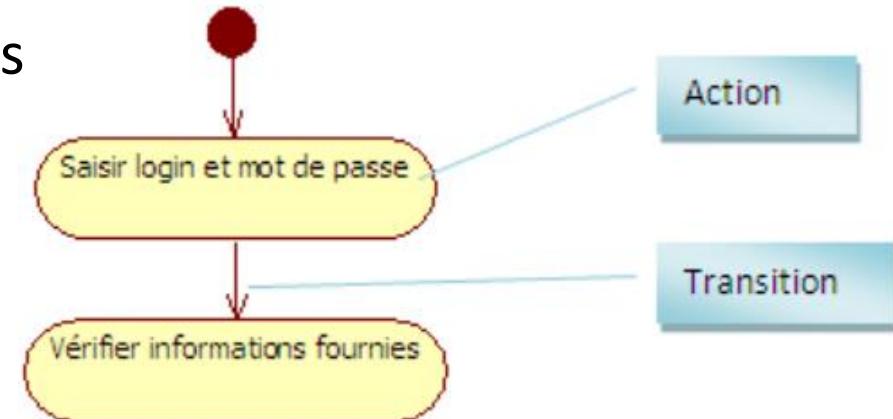
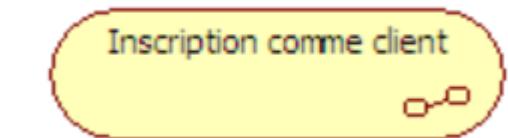


Diagramme d'Activités

Le lot d'actions, ou autre cas d'utilisation :

- Si une action du cas d'utilisation correspond à l'appel d'un cas d'utilisation interne (lié par une relation de type «**include**» ou «**extend**»); elle est représentée par une action contenant un signe spécial : deux cercles reliés par un trait



Les transitions conditionnelles (alternatives) :

Elle permet d'indiquer les différents scénarios du cas d'utilisation dans un même diagramme

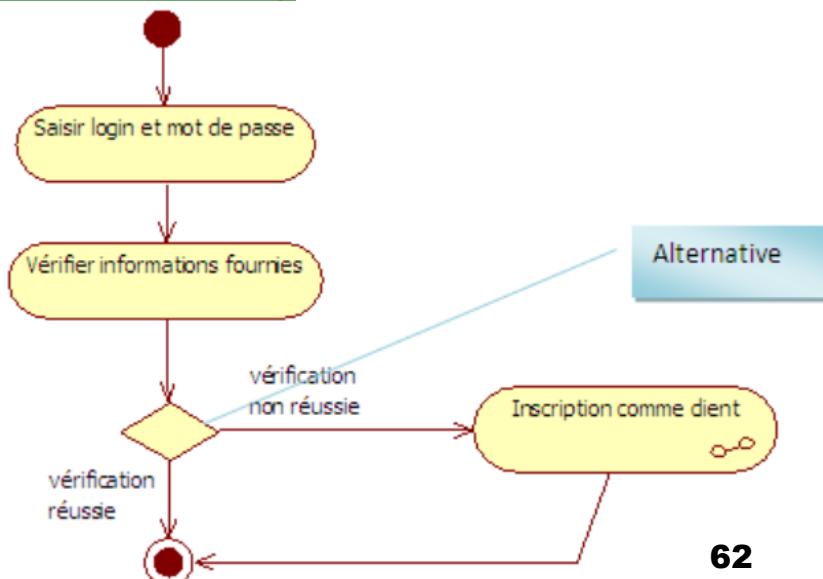
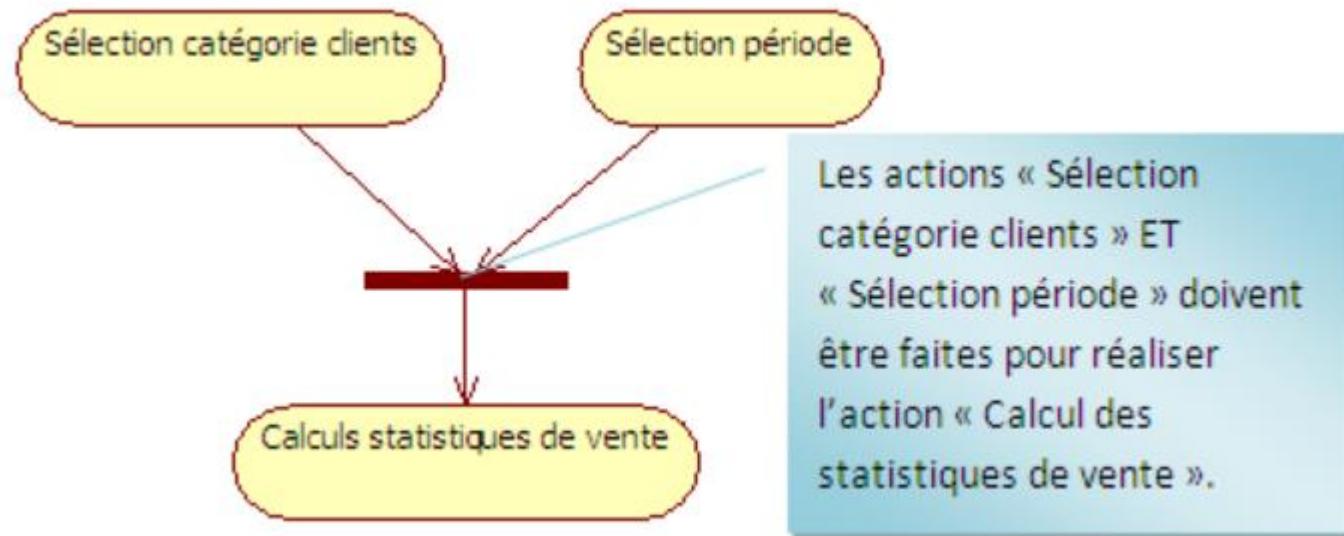


Diagramme d'Activités

La synchronisation :

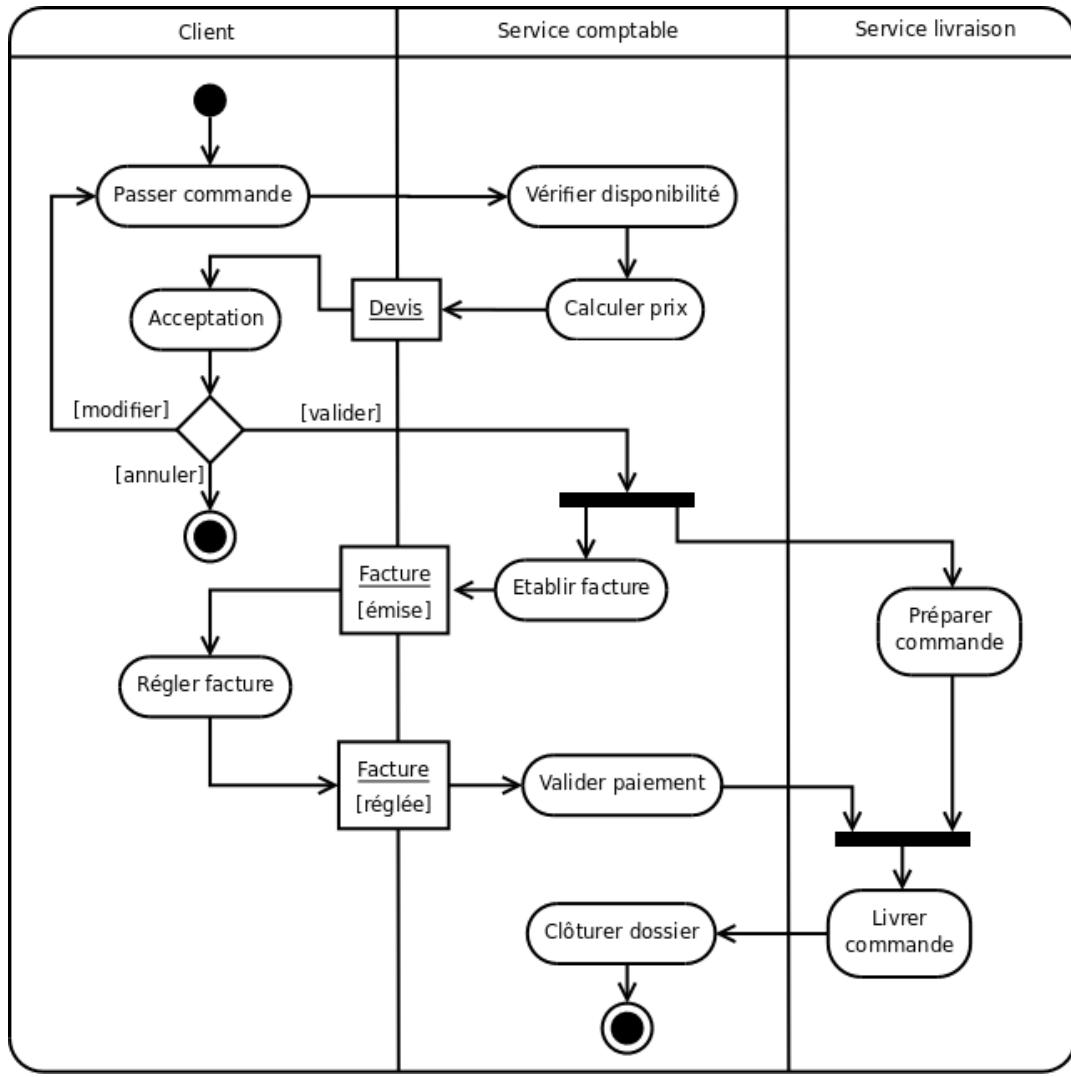
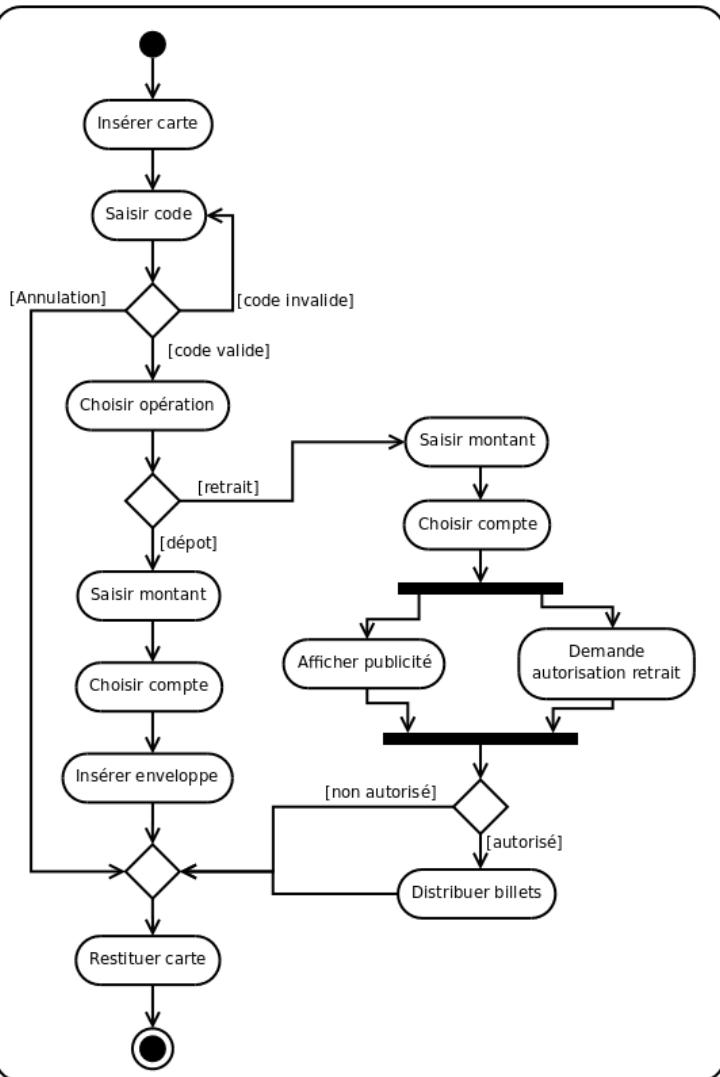
- Elle indique qu'il faut avoir réalisé deux actions pour pouvoir réaliser la troisième en-dessous



- Une barre de synchronisation permet d'ouvrir et de fermer des branches parallèles au sein d'un flot d'exécution

Diagramme d'Activités

Exemples d'un diagramme d'activités



8. Diagramme de composants

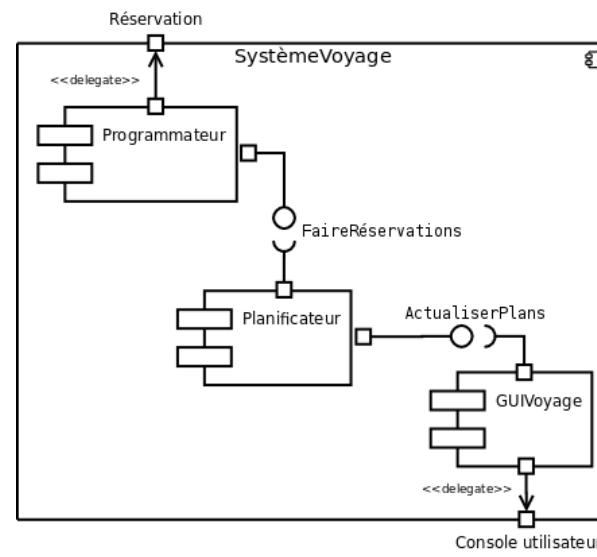


Diagramme de composant

Notion de composant (component) et d'interface :

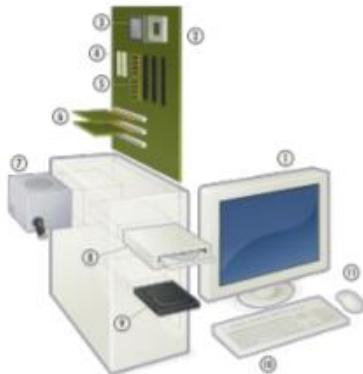
En UML, un **composant** est un élément **logiciel** remplaçable et réutilisable qui fourni ou reçoit un service bien précis. Il peut être vu comme une pièce détachée du logiciel. Les plugins, les drivers, les codecs, les bibliothèques sont des composants

Ex : le codec vidéo



Pour lire un film codé en **Divx**, le logiciel **Window Media Player** a besoin d'un **codec divx**.

- ♦ Ce codec peut aussi être utilisé par un autre logiciel de lecture vidéo (**RealPlayer, QuickTime, Wamp...**).
- ♦ Ce codec peut être remplacé par un codec plus récent et plus performant.



Ex : parallèle avec les composants d'un ordinateur.

Un ordinateur est un ensemble de composants modulaires qui fournissent et reçoivent des services (carte mère, carte graphique, disque dur, clavier, écran...). Chacun de ces composants est remplaçable par un autre composant (pas forcement identique) à condition qu'il ait des interfaces compatibles (nous ne pouvons pas mettre un écran avec une connexion VGA à la place d'un écran avec une connexion HDMI).

ATTENTION : ceci n'est qu'un parallèle pour faire comprendre la notion de composant. **En UML les composants ne sont pas des éléments matériels mais des éléments logiciels.** Éléments logiciels qui par contre seront installés sur des éléments matériels (ce que nous verrons lorsque nous aborderons le diagramme de déploiement).

Diagramme de composant

Notion de composant (component) et d'interface :

- Les **diagrammes de composants** permettent de décrire l'architecture logicielle et statique d'une application en termes de composants et de dépendances entre ces composants

Les composants fournissent des services via des interfaces

- Un composant peut être remplacé par n'importe quel autre composant compatible (ayant les mêmes interfaces)
- Un composant peut évoluer indépendamment des applications ou des autres composants qui l'utilise à partir du moment où les interfaces sont respectées

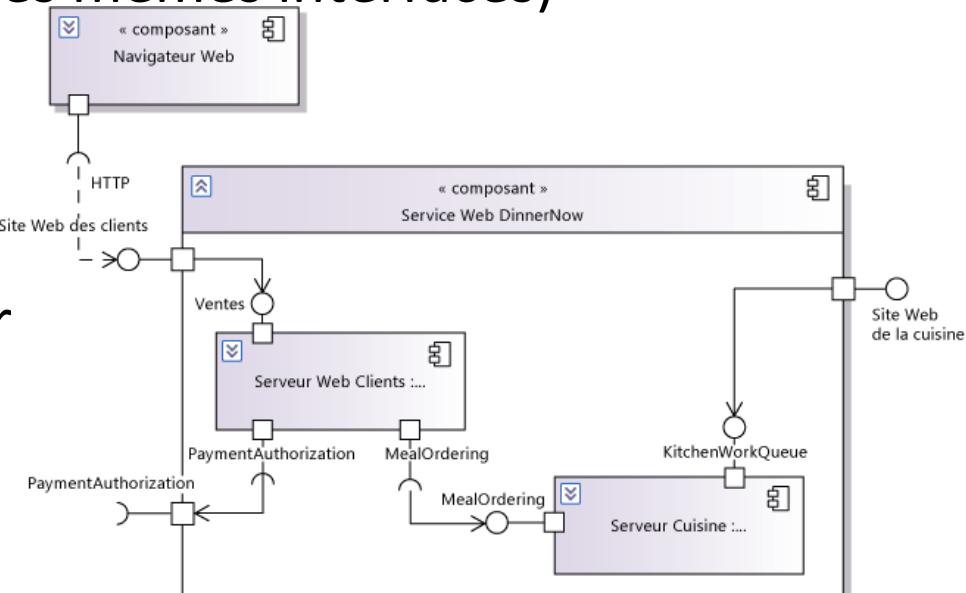
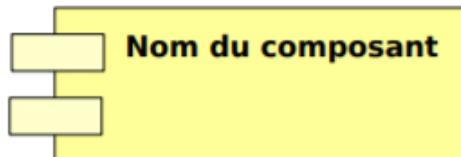


Diagramme de composant

Représentation graphique

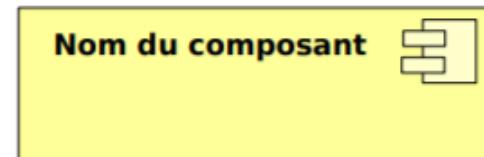
1. Les composants

Il existe plusieurs possibilités pour représenter un composant



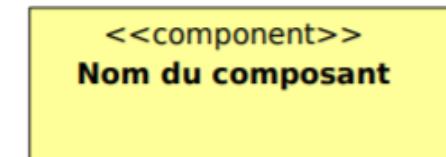
Un rectangle dans lequel figure :

- Le nom du composant.
- 2 petits rectangles l'un au dessus
- de l'autre à cheval du côté gauche.



Un rectangle dans lequel figure :

- Le nom du composant.
- Le symbole en haut à droite.



Un rectangle dans lequel figure :

- Le stéréotype <<component>>.
- Le nom du composant.

Cette représentation peut être différente en fonction du modeleur UML utilisé

Diagramme de composant

Représentation graphique

2. Les interfaces : Il existe deux types d'interface :

- ✓ **Les interfaces requise** : Ce sont des interfaces qui fournissent un service au composant et dont il a besoin pour fonctionner
- ✓ **Les interfaces fournies** : Ce sont des interfaces par lesquels le composant fourni lui-même un service
- Il existe plusieurs possibilités pour représenter une interface

2.1 Intégrées dans la représentation du composant : grâce aux stéréotypes « *required interface* » et « *provided interface* »

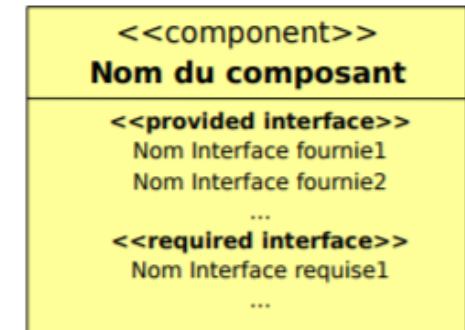
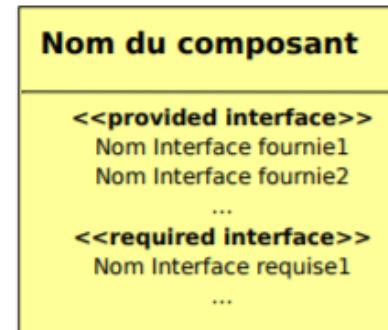
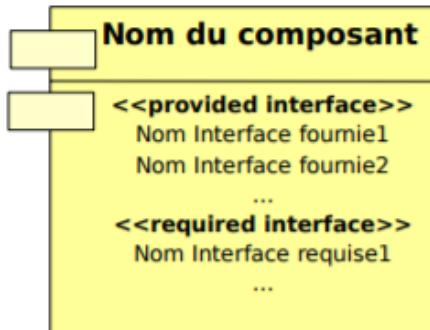


Diagramme de composant

Représentation graphique

2.2 Dans un classeur séparé du composant dans lequel sont listés les différents services :

- ✓ **Les interfaces requises** sont reliées au composant par une flèche en pointillées sur laquelle figure le stéréotype « **use** »
- ✓ **Les interfaces fournies** sont reliées au composant par une flèche en pointillées sur laquelle figure le stéréotype « **realize** » (le bout de la flèche est un triangle vide).

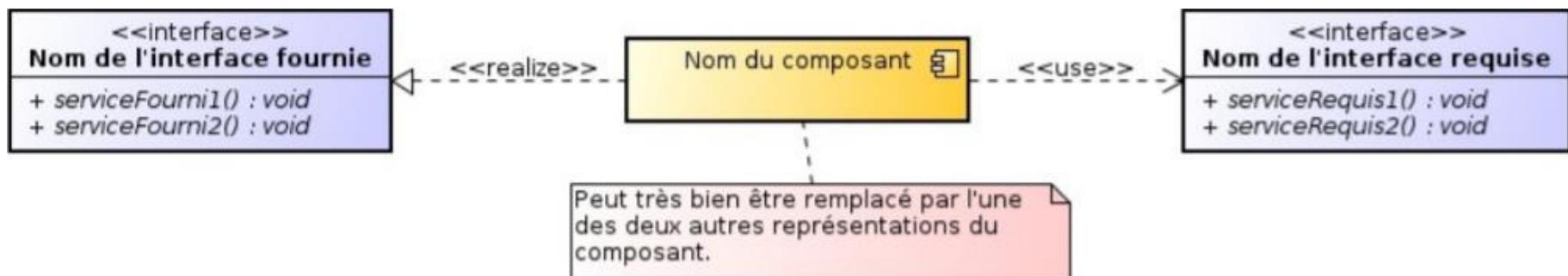


Diagramme de composant

Représentation graphique

2.3 Avec des connecteurs d'assemblage :

- ✓ Les interfaces requises sont représentées par un demi-cercle
- ✓ Les interfaces fournies sont représentées par un cercle
- ✓ Les interfaces sont raccordées au composant par un trait

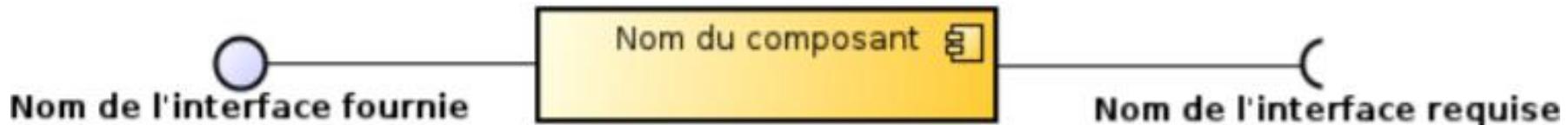
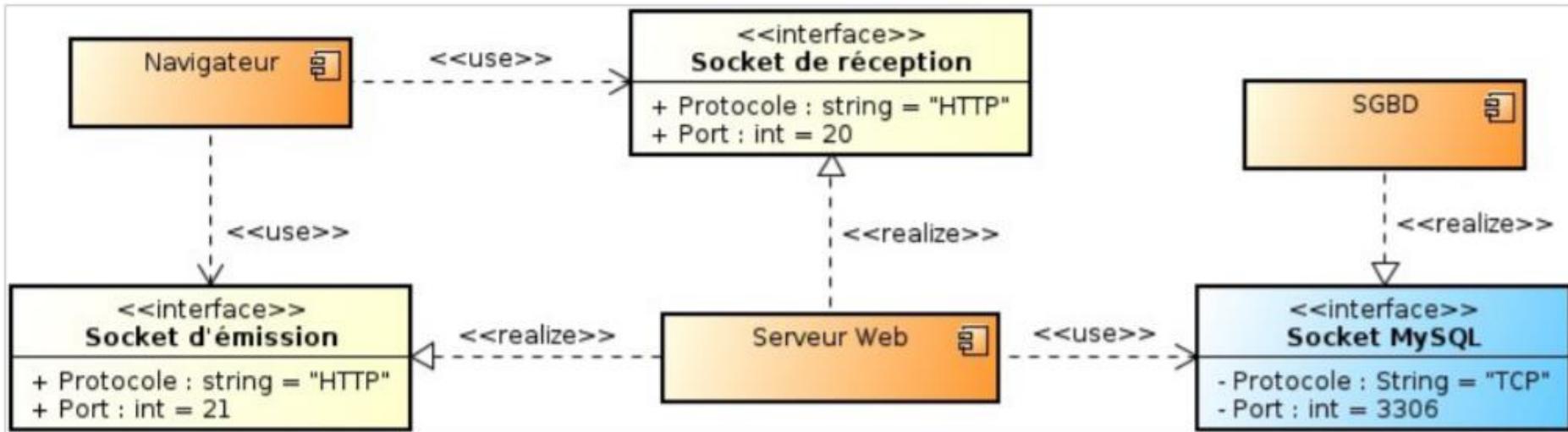


Diagramme de composant

Exemple : Transfert de données par internet



Ou

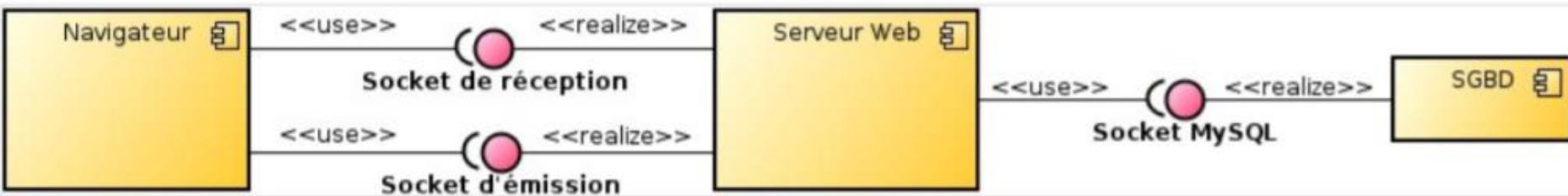


Diagramme de composant

Représentation graphique

3. Les ports :

Le port est le point de connexion entre le composant et son environnement, il est la matérialisation de l'interface. Nous le représentons par un petit carré à la périphérie du composant

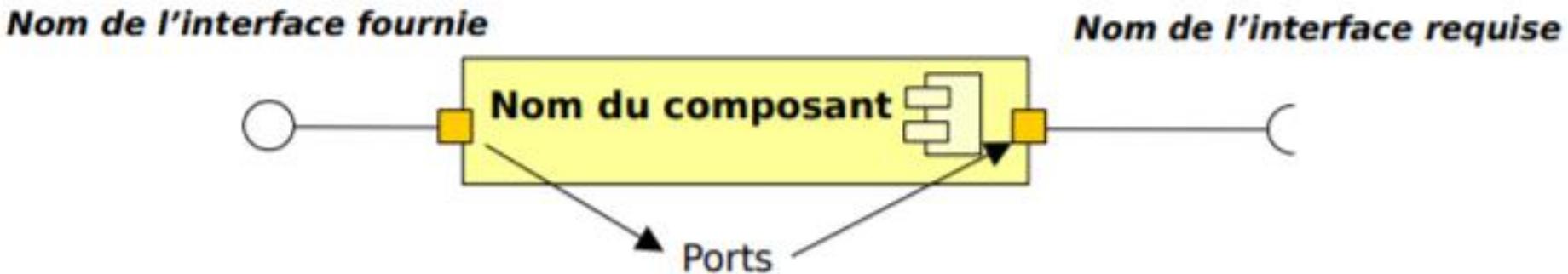
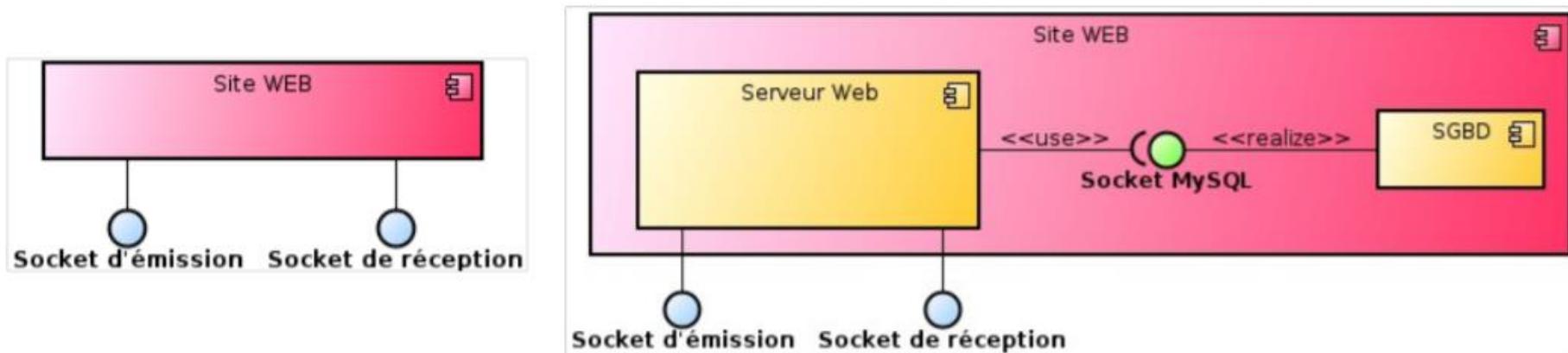


Diagramme de composant

Boite noire – boite blanche

Un composant peut être vu de 2 manières :

- Comme une **boite noire** dont nous ne connaissons pas le contenu et auquel nous accédons via les interfaces qui sont la seule partie visible
- Comme une **boite blanche** en spécifiant les objets qui constituent le composant et en indiquant leurs relations.



9. Diagramme de déploiement

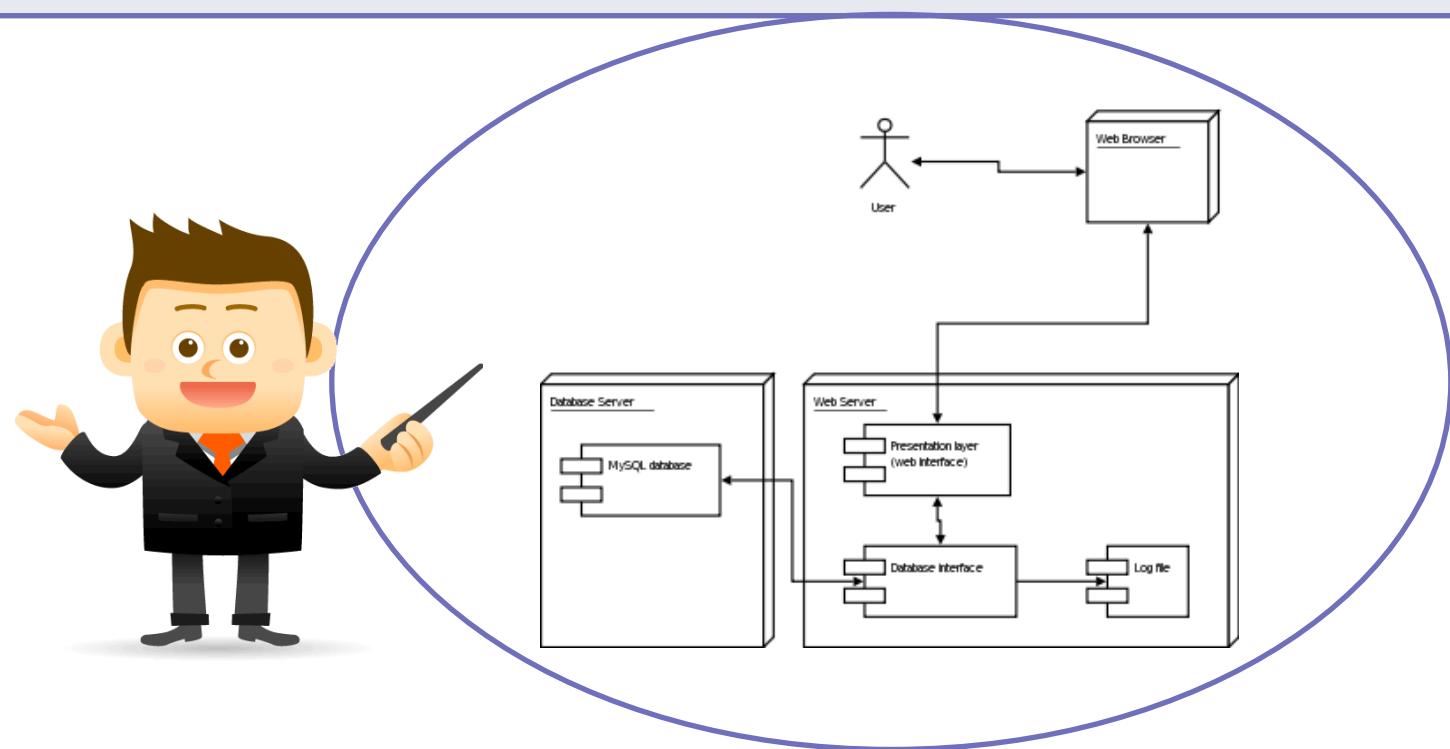


Diagramme de déploiement

Le diagramme de déploiement fait parti des diagrammes structuraux (**statique**), il représente :

- La disposition **physique** des **ressources matérielles** qui constituent le système et montre la répartition des composants (élément logiciels) sur ces matériels
- La nature des connexions de communication entre les différentes ressources matérielles
- Les éléments du diagramme de déploiement sont :
 - Les nœuds
 - Les chemins de communication

Diagramme de déploiement

1. Les nœuds :

- Une ressource matérielle est représentée par un **nœud**
- Les ressources matérielles sont quelquefois représentées avec le stéréotype « **device** » (comme par exemple les périphériques ou les ordinateurs)
- Un nœud est représenté par un **parallélépipède rectangle** dans lequel figure son nom



Diagramme de déploiement

1. Les nœuds :

- Un nœud possède des attributs (quantité de mémoire, vitesse de processeur, marque, type...) que nous pouvons spécifier à l'intérieur du parallélépipède

<<device>>
Ordinateur

Vitesse processeur
Ram
Capacité disque dur
...

Ordinateur-Salle5 : Ordinateur

Vitesse processeur : 3GHz
Ram : 4Go
Capacité disque dur : 1To
...

Diagramme de déploiement

1. Les nœuds :

- Pour montrer qu'un composant est affecté sur un nœud, il faut :
 - Soit placer le composant dans le nœud,
 - Soit le relier à l'aide d'une relation de dépendance (flèche en pointillées) stéréotypée «**support**» orientée du composant vers le nœud

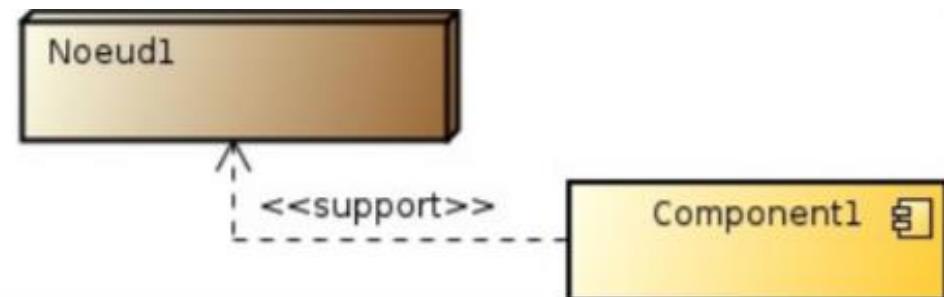
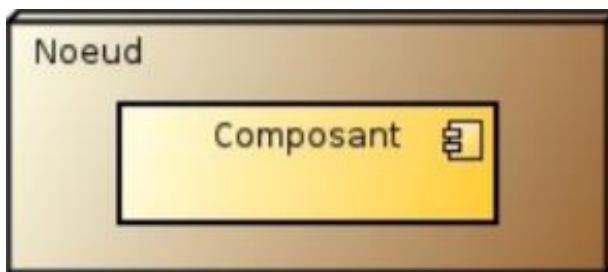


Diagramme de déploiement

2. Les chemins de communications :

Les différents nœuds qui apparaissent dans le diagramme de déploiement sont connectés entre eux par des lignes qui symbolisent un support de communication. Ce sont les **chemins de communications**

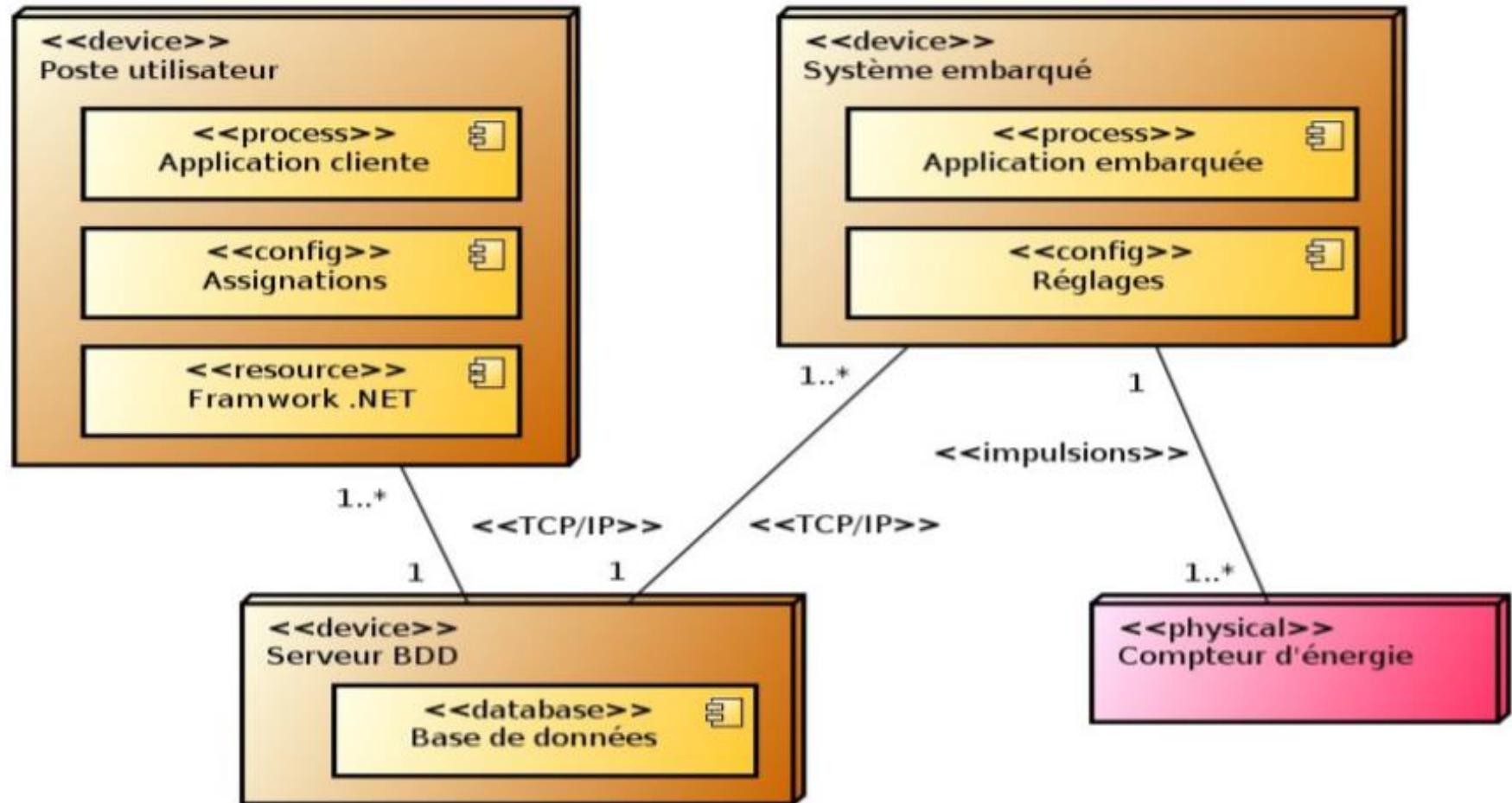
- ❑ Le chemin de communication est un lien qui permet de modéliser de façon simpliste la communication entre deux nœuds (liaison Ethernet, USB, série...)
- ❑ Il est possible de faire figurer sur ce lien :
 - ❑ les cardinalités ;
 - ❑ des contraintes entre accolades (pour indiquer par exemple qu'un accès est sécurisé) ;
 - ❑ le type de réseau et/ou son débit en l'indiquant comme un stéréotype...



Diagramme de déploiement

Exemple d'un diagramme de déploiement :

Projet de mesure et d'analyse de consommation énergétique...



Outils UML standards

- ❑ Besoin d'outils pour manipuler le méta-modèle UML
 - ❑ Parcours de tous les éléments de modèle d'un certain type (ex : accès à toutes les classes)
 - ❑ Navigation des liens entre éléments (ex : accès à toutes propriétés d'une classe)
 - ❑ Création d'éléments de modèle
 - ❑ ...
- ❑ Il est communément établie qu'un outil UML standard est un outil qui :
 - ❑ Respecte intégralement la notation UML
 - Même si tous les diagrammes ne sont pas supportés
 - ❑ Dispose d'un format de représentation interne compatible avec le méta-modèle UML standard
 - La difficulté de ce point s'illustre avec XMI

Exemples d'outils standards

- Rational Rose
 - Outil le plus important du marché acheté par IBM
 - <http://www-306.ibm.com/software/rational>
- Together
 - Outil fortement couplé avec Java (Borland)
 - <http://www.borland.fr/together>
- ArgoUML
 - Outil Open Source
 - <http://argouml.tigris.org>
- Visio
 - Outil non complet de microsoft
 - <http://www.microsoft.com/office/visio>
- Enterprise Architect
 - Outil UML2 : Sparx Systems
 - <http://www.sparxsystems.com.au/products/ea/index.html>
- Objecteering
 - Outil UML2 : Softeam
 - <http://www.objecteering.com>

Outils TP ...

Objecteering

<http://www.objecteering.com/>

Objecteering the model-driven development tool

Solutions Products Services & Support Resources Downloads About us

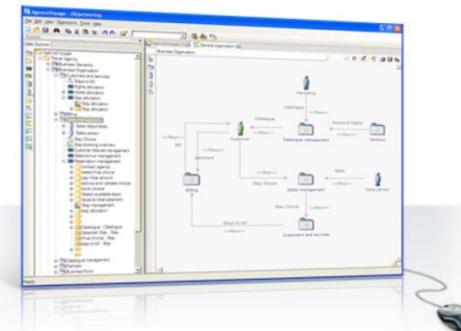
The UML, SOA, BPMN, EA and MDA convergence for Model Driven Engineering

Exclusive!

Objecteering Packaging
10 good reasons for choosing Objecteering
All 14+ diagrams Objecteering supports
Complete system modeling
EA, SOA, BPM
UML2: Complete support
MDA technology: the latest innovations
Java, C#, C++
Goals, Business rules, Dictionary
Requirements analysis
Teamwork with no limits
Objecteering Tutorials



Enterprise Architecture Modeling



Events

Modelio, new modeling tool and successor of Objecteering

Modeliosoft and Objecteering Software are pleased to announce the convergence of two modeling tools, Modelio and Objecteering.

References

“ Objecteering has helped us to better formalize our system specifications in UML, thereby saving a considerable amount of time during the programming phase. ”

(Ronan Viel - R&D Unit Manager, France Télécom)





HOME DOWNLOADS ▾ RESOURCES ▾ COMMUNITY ▾ DOCUMENTATION ▾ ABOUT MODELIO ▾

MODELIO

An open source modeling environment supporting the main standards:

UML, BPMN, MDA, SysML, ...



Download **Forum** **Forge** **Store**

Modelio

<https://www.modelio.org/>