

conception UML

Le Langage de Modélisation Unifié, de l'anglais Unified Modeling Language (UML), est un langage de modélisation graphique à base de pictogrammes conçu comme une méthode normalisée de visualisation dans les domaines du développement logiciel et en conception orientée objet.

Avantages de la modélisation normalisée :

Lorsque nous parlons de normes, nous parlons de règles.

Le respect et la connaissance de ces règles, fait que tout le monde parle d'une même voix, et donc tout le monde se comprend.

Sachant que le résultat d'une modélisation est un ensemble de dossiers que nous fournirons au client, aux programmeurs, aux testeurs, aux intégrateurs etc.., il est nécessaire d'utiliser une méthodologie normalisée (comme Merise), et un mode de représentation normalisé de cette méthodologie, c'est ici qu'intervient UML.

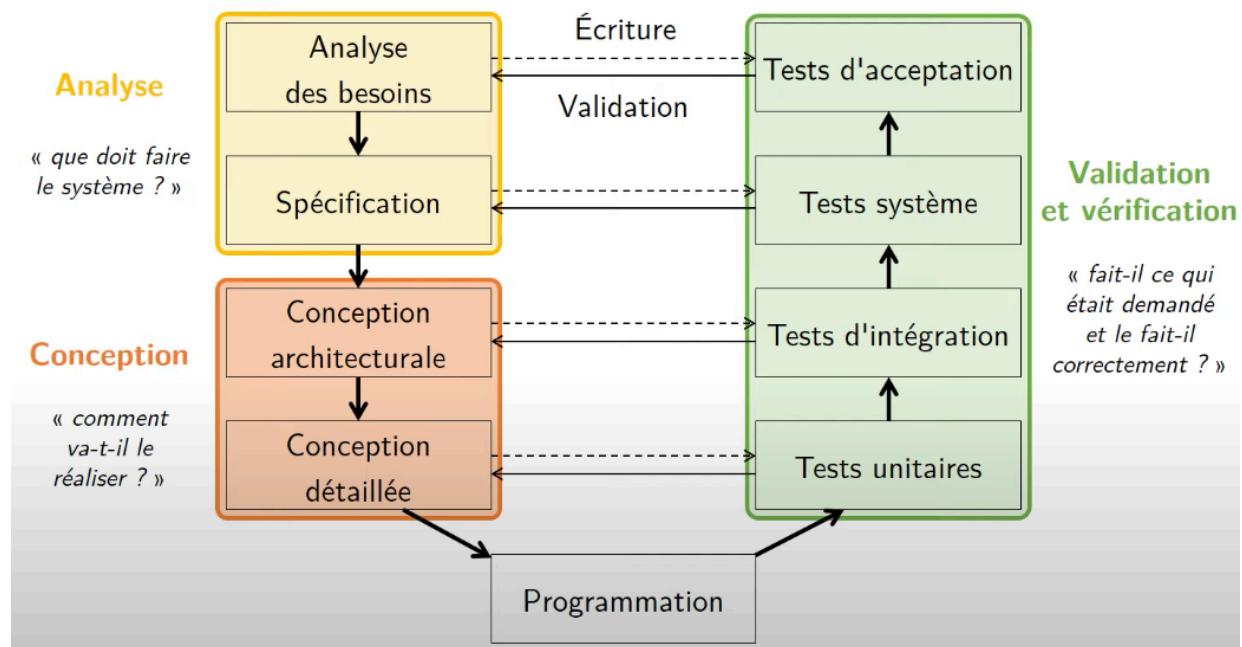
Diagrammes de cas d'utilisation

Analyse des besoins

Point de départ du précessus de développement, ce diagramme se situe dans l'analyse des besoins.

A quoi et et à qui va servir le système que nous allons développer ?

Processus de développement en V



Le but de cette analyse c'est de comprendre les besoins du client pour rédiger le cahier des charges fonctionnel.

Trois questions :

- Définir les utilisateurs principales du système : à quoi sert-il ?
- Définir l'environnement du système : qui va l'utiliser ou interagir avec lui ?
- Définir les limites du système : où s'arrête sa responsabilité ?

Scénario d'utilisation

Suite d'interactions entre un ou plusieurs utilisateurs et le système qui va permettre à l'utilisateur d'effectuer une tâche complexe :

- décrire une interaction entre l'utilisateur et le système
- permettant de réaliser un objectif

Système : Site de vente en ligne

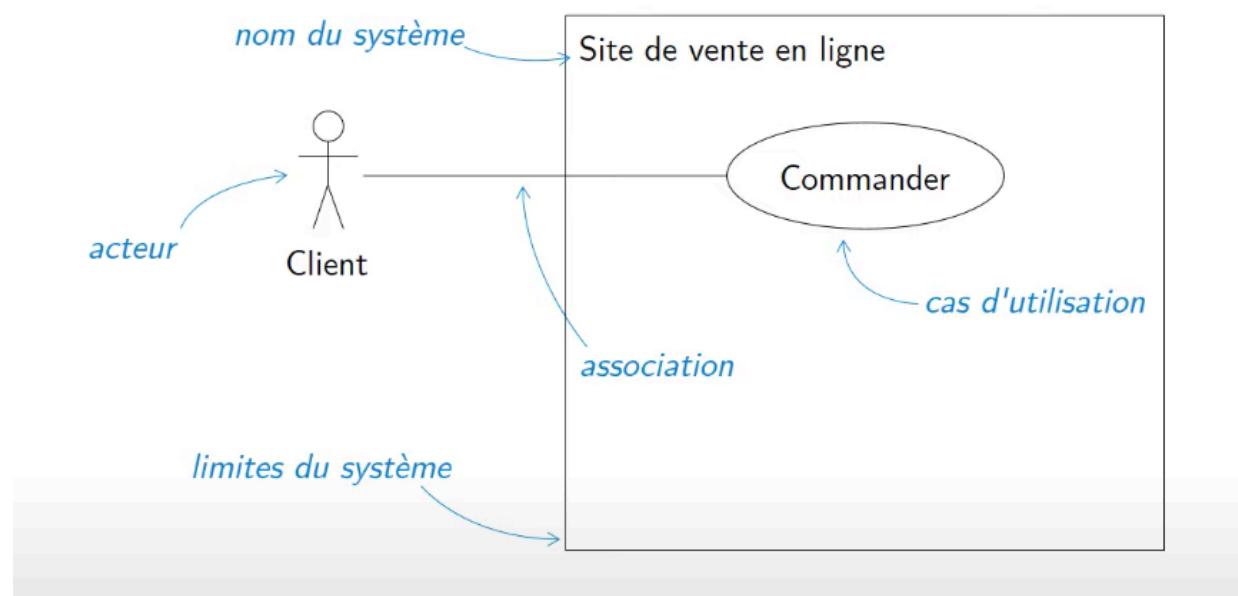
Scénario : Commander

Le client s'authentifie dans le système puis choisit une adresse et un mode de livraison. Le système indique le montant total de sa commande au client. Le client donne ses informations de paiement. La transaction est effectuée et le système en informe le client par e-mail.

Définir les fonctionnalités principales du système du point de vue extérieur :

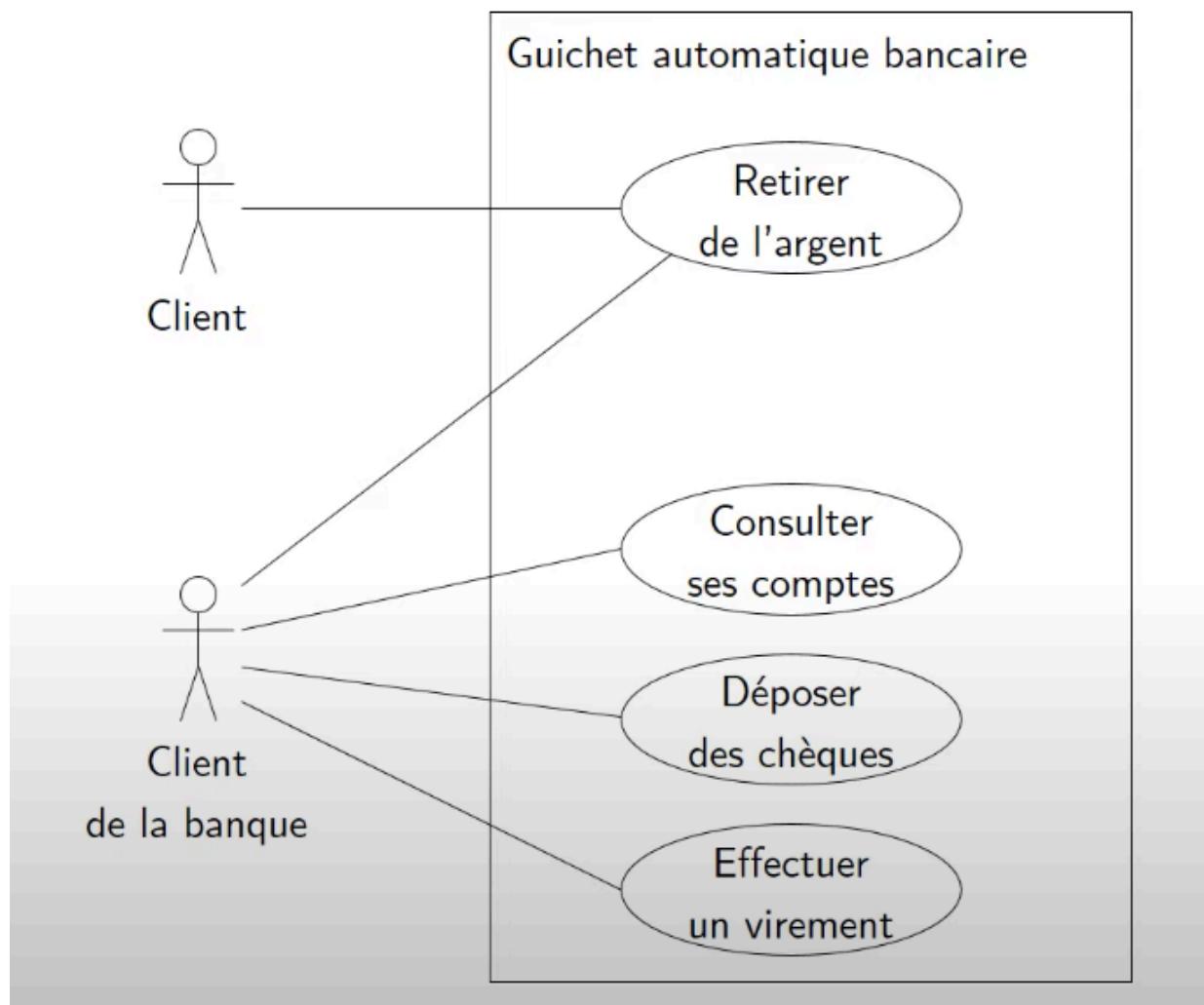
- Acteur : entité qui interagit avec le système
 - Personne, chose, logiciel, extérieur au système décrit
 - Représenté par un rôle (plusieurs rôles possibles pour une même entité)
 - Identifié par le nom du rôle
- Action : fonctionnalité visible de l'extérieur
 - Action déclenchée par un acteur
 - Identifié par une action (verbe à l'infinitif)

Diagramme (de cas d'utilisation)

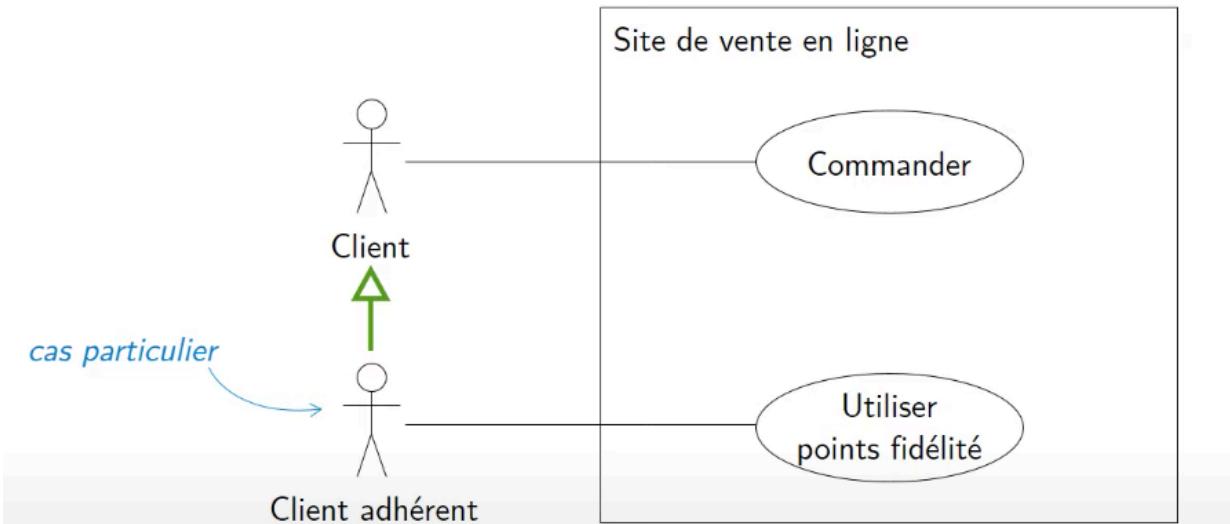


Attention au vocabulaire, car il sera utilisé pendant toute la conception. Un glossaire peut être fourni.

Exemple



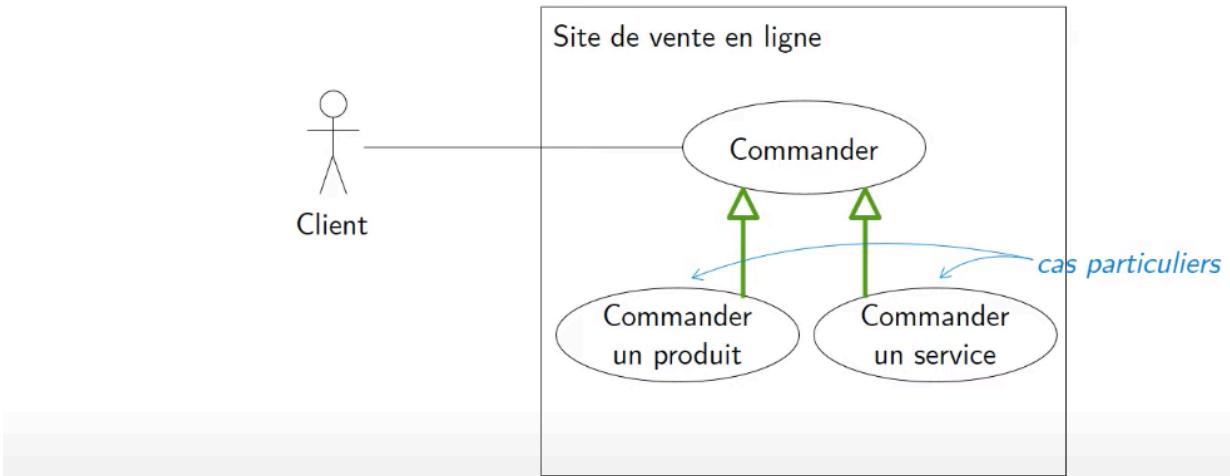
avec héritage



Situation : Y peut faire tout ce que fait X

Modélisation : Faire apparaître Y comme un cas particulier de X
(ou X généralisation de Y)

relation entre cas d'utilisation



Généralisation : X est un **cas particulier** de Y
Tout ou partie du scénario de Y est spécifique à X

Diagrammes d'activité

Les diagrammes d'activités permettent de déterminer des traitements a priori séquentiels. Ils offrent un pouvoir d'expression très proche des langages de programmation objet:

spécification des actions de base (déclaration de variables, affectation etc.), structures de contrôle (conditionnelles, boucles), ainsi que les instructions particulières à la programmation orientée objet (appels d'opérations, exceptions etc.).

Ils sont donc bien adaptés à la spécification détaillée des traitements en phase de réalisation. On peut également utiliser de façon plus informelle pour décrire des enchaînements d'actions de haut niveau, en particulier pour la description détaillée en cas d'utilisation.

Les diagrammes d'activités présentent plusieurs avantages pour les utilisateurs :

- Démontrer la logique d'un algorithme
- Décrire les étapes effectuées dans un cas d'utilisation d'UML
- Illustrer un processus métier ou un flux de travail entre les utilisateurs et le système
- Simplifier et améliorer n'importe quel processus en clarifiant les cas d'utilisation complexes
- Modéliser des éléments de l'architecture de logiciels, tels que la méthode, la fonction et l'utilisation

Composants de base d'un diagramme d'activités :

- **Action** : étape dans l'activité où les utilisateurs ou le logiciel exécutent une tâche donnée. Les actions sont symbolisées par des rectangles aux bords arrondis.
- **Nœud de décision** : embranchement conditionnel dans le flux, qui est représenté par un losange. Il comporte une seule entrée et au moins deux sorties.
- **Nœud de départ** : élément symbolisant le début de l'activité, que l'on représente par un cercle noir.
- **Nœud de fin** : élément symbolisant l'étape finale de l'activité, que l'on représente par un cercle noir avec un contour.

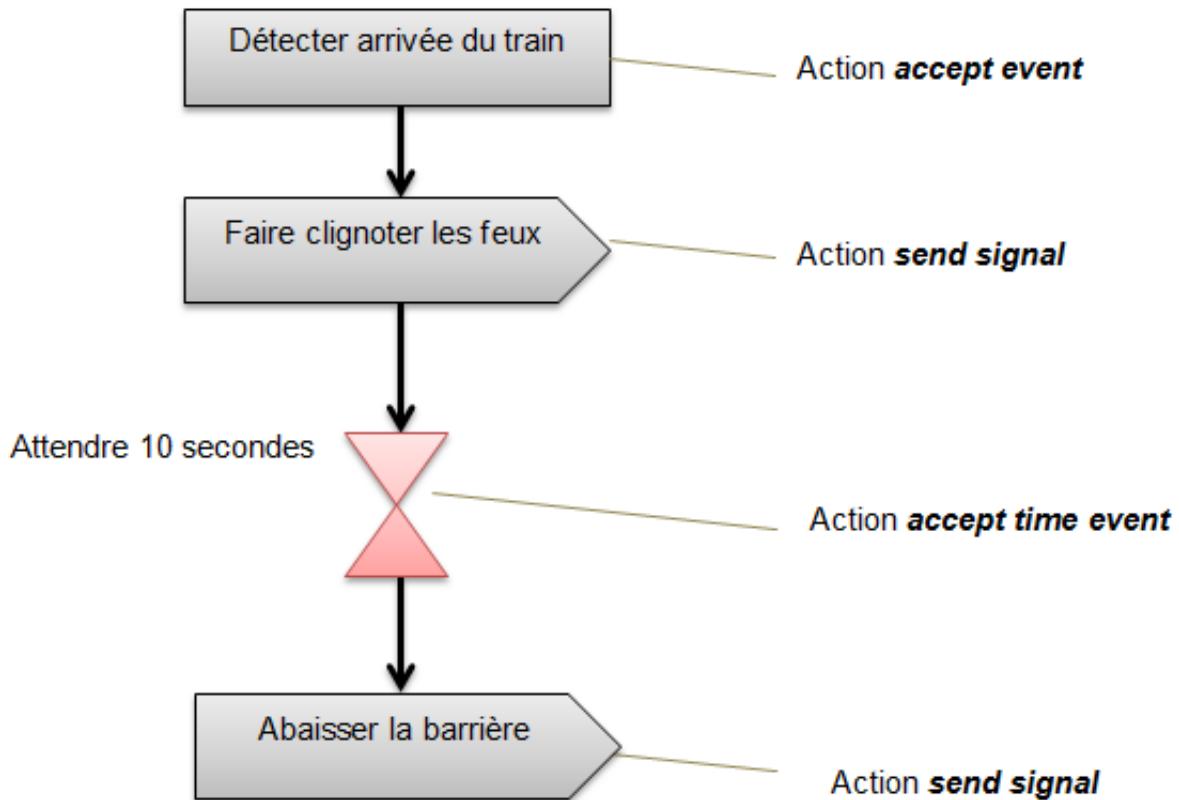
Action

Une action est le plus petit traitement qui puisse être exprimé en UML. Une action a une incidence sur l'état du système ou en extrait une information. Ce sont des étapes discrètes à partir desquelles se construisent les comportements. La notion d'action est à rapprocher de la notion d'instruction élémentaire d'un langage de programmation. Par exemple, une action peut être:

- la création d'un nouvel objet ou lien

- l'émission d'un signal
- une affectation de valeur à des attributs
- la réception d'un signal
- un calcul arithmétique simple etc.

Les différents types d'actions



Premièrement, on détecte l'arrivée du train; cette action représente l'action "accept event" c'est-à-dire qu'on reçoit le signal de l'arrivée du train. Deuxièmement, "faire clignoter les feux" est une action "send signal", cela veut dire qu'on envoie un signal qui est transmis à un objet cible sans attendre que ce dernier ait bien reçu le signal.

Ensuite, l'action "time event" est un événement temporel déclenché après l'écoulement d'une certaine durée. Enfin, "abaisser la barrière" est une action "send signal", un message est envoyé et transmis à la cible.

On distingue graphiquement les actions associés à une communication: send signal, accept event et accept time event. Cela permet de mieux mettre en valeur les échanges entre les diagrammes de la spécification.

Activité

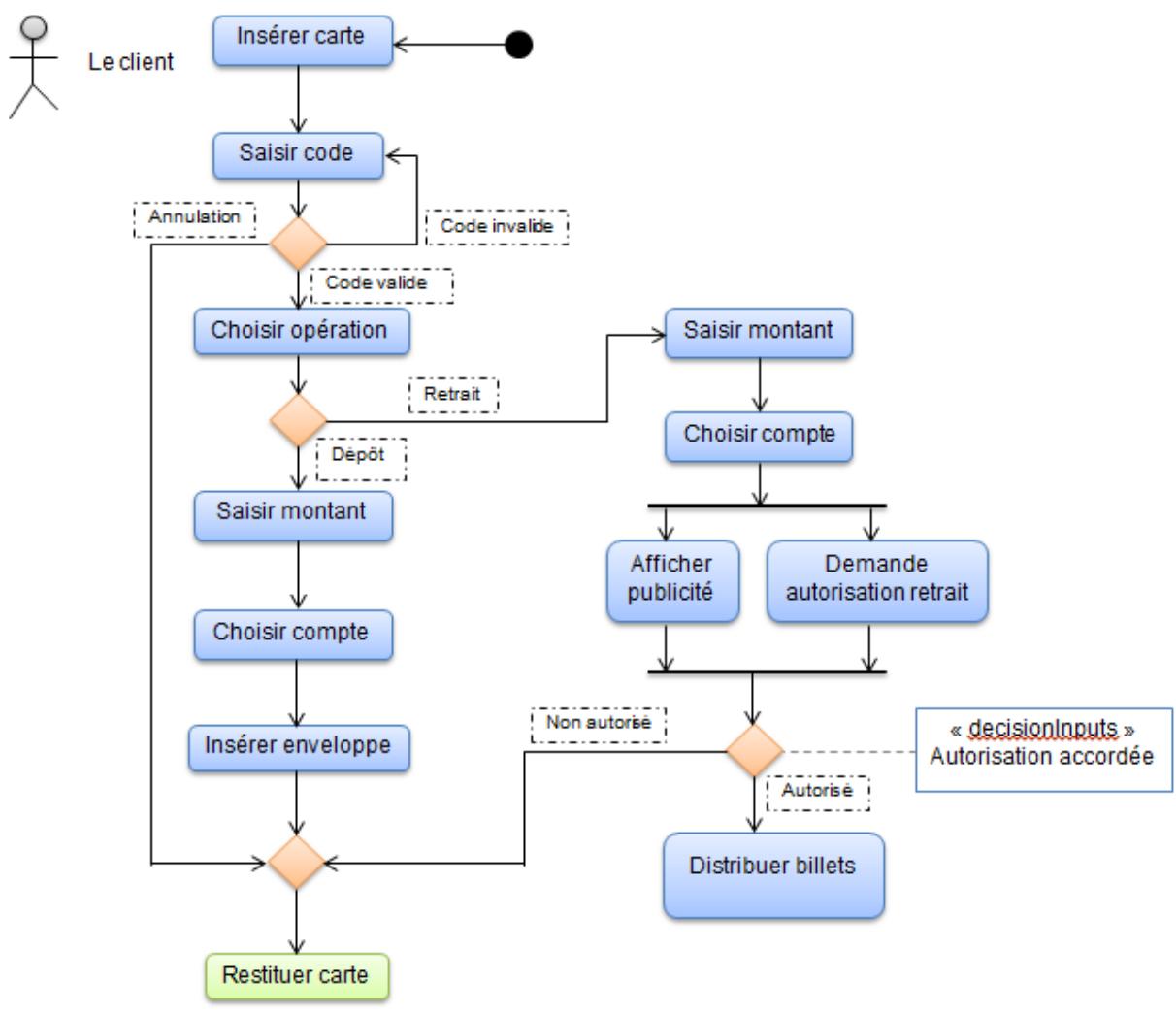
Une activité définit un comportement décrit par une série organisée d'unités dont les

éléments simples sont les actions. Le flot d'exécution est modélisé par des nœuds reliés par des arcs (transitions). Le flot de contrôle reste dans l'activité jusqu'à ce que le traitement soit terminé.

Une activité est un comportement et peut comporter des paramètres en entrée ou en sortie, ainsi que des variables locales au même titre qu'une opération.

Une activité peut regrouper des nœuds et des arcs, c'est ce qu'on peut appeler "groupe d'activités". Les nœuds et les arcs peuvent appartenir à plusieurs groupes. Le groupe d'activités représente un système qui est générique regroupant des activités pouvant être utilisé de façon variée.

Un diagramme d'activité est lui-même un groupe d'activités. Voici ci-dessous un exemple de diagramme d'activités qui représente le fonctionnement d'une borne bancaire:



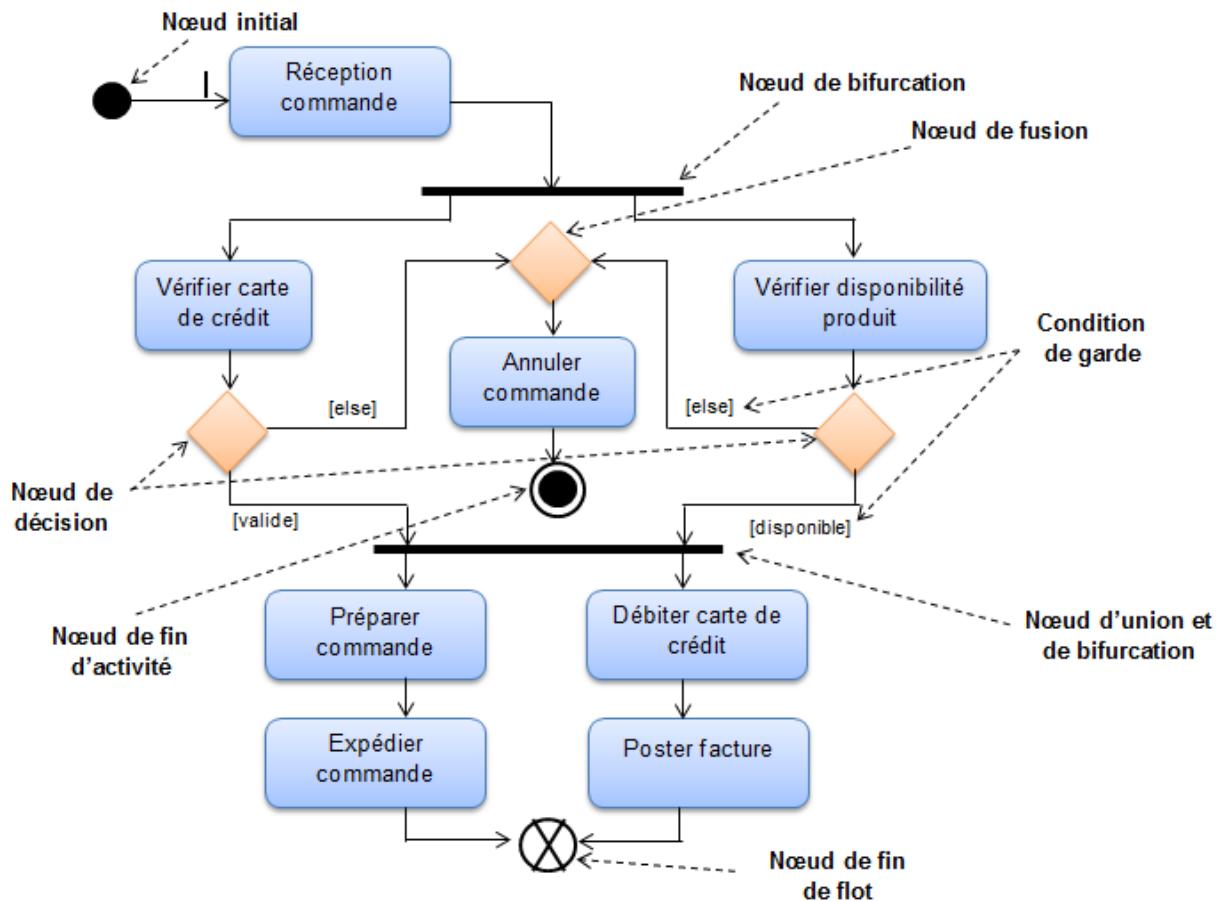
Cet exemple illustre différentes représentations des actions. Après avoir saisi le code, deux activités sont déclenchées: on choisit l'opération souhaitée si le code est valide ou la carte est restituée si on annule l'opération. Après avoir choisi l'opération, on trouve deux alternatives: choisir un montant qu'on dépose (dépôt) ou retire (retrait).

Dans les deux cas, nous choisissons le compte par la suite. Si nous avons choisi de déposer des billets, nous devons insérer une enveloppe ce qui nous donne droit à la restitution de notre carte à la fin de l'opération.

Si nous avons pris le choix d'effectuer un retrait de billets, nous avons le droit à deux options: afficher une publicité ou demander une autorisation de retrait. Enfin, la note portant le mot-clé "decisionInput" spécifie un critère de décision parmi plusieurs arcs d'activité, dont chacun doit satisfaire la variable "autorisation accordée" à "Non autorisé" ou "Autorisé" avant que la transition associée ne puisse être déclenchée

Nœuds

Après avoir expliqué le diagramme d'activités et son fonctionnement, nous abordons majoritairement des nœuds qui sont essentiels dans un diagramme d'activités. Elle permet d'évoquer davantage en détail le contenu d'un diagramme d'activités.



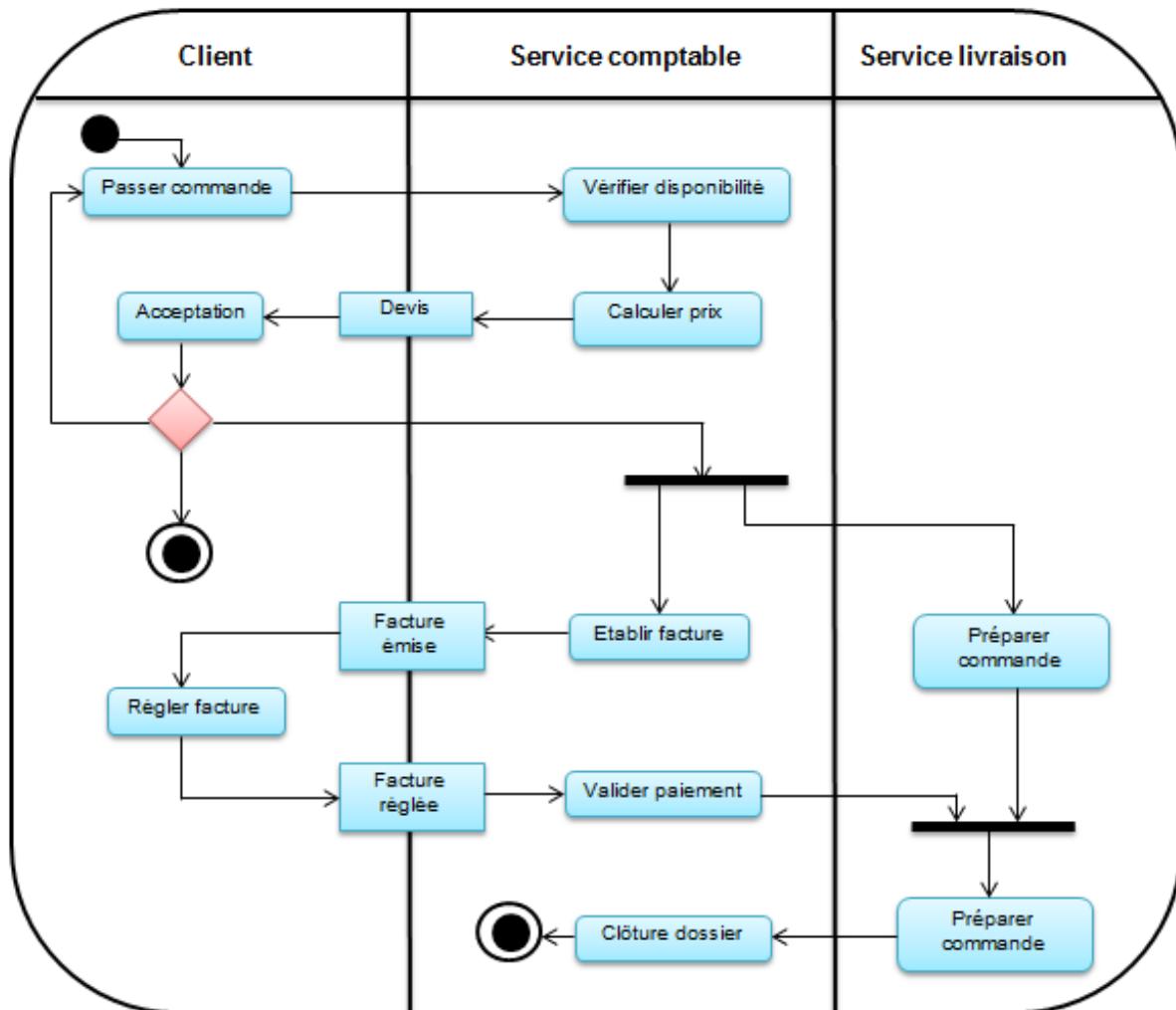
Partitions

Les partitions sont aussi des éléments essentiels dans un diagramme d'activités. En effet, elles sont appelées couloirs ou lignes d'eau. Elles ont pour but d'organiser les noeuds d'activités disposés dans un diagramme d'activités par le biais de regroupements. Ce sont

des unités d'organisation du modèle.

Ces partitions sont utiles lorsqu'on doit désigner la classe responsable qui rassemble un ensemble de tâches par exemple. La classe en question est donc responsable du comportement des noeuds à l'intérieur de la partition précédemment évoquée.

Graphiquement, les partitions sont représentées par des lignes continues. Elles peuvent prendre la forme d'un tableau. De plus, les noeuds d'activités doivent appartenir à une seule et unique partition et les transitions peuvent passer à travers les frontières des partitions.



Le diagramme d'état

Représentation des états et transitions.

Le diagramme d'état est un outil de modélisation puissant dans le langage UML (Unified Modeling Language) qui permet de représenter les différents états qu'un objet peut prendre au cours de son cycle de vie, ainsi que les transitions entre ces états. Ce diagramme met en évidence la dynamique du comportement d'un objet en illustrant comment il réagit aux événements et comment il évolue d'un état à un autre.

États

Les états, au sein d'un diagramme d'état en UML, servent à représenter les différentes conditions ou situations dans lesquelles un objet peut se trouver à un moment précis. Chaque état est généralement visualisé par un rectangle avec un nom évocateur. Prenons l'exemple d'un processus de réservation de vol. Dans ce contexte, les états pourraient inclure "En Attente", "Confirmé", "Annulé", etc. Chaque objet, lorsqu'il est observé, est dans un et un seul état à la fois, reflétant ainsi son état actuel.

Transitions

Les transitions, quant à elles, décrivent les chemins ou les voies par lesquels un objet évolue d'un état à un autre en réponse à des événements spécifiques. Ces transitions, représentées par des flèches dirigées entre les rectangles des états, permettent de visualiser comment un objet change de comportement suite à des événements déclencheurs. Par exemple, lorsqu'un vol est réservé avec succès, une transition pourrait être définie entre l'état "En Attente" et l'état "Confirmé". Il est important de noter que certaines transitions peuvent être conditionnelles et comporter des gardes. Les gardes sont des conditions qui doivent être remplies pour que la transition puisse s'exécuter.

Événements

Les événements constituent les déclencheurs qui engendrent des transitions entre les états d'un objet. Ils reflètent les changements dans l'environnement de l'objet ou dans son état interne. Ces événements peuvent être de nature externe, tels que des actions entreprises par des utilisateurs, des messages reçus ou des déclencheurs de système. D'autre part, ils peuvent également être internes, comme des temporiseurs ou des processus intrinsèques provoquant des changements d'état. Par exemple, dans le contexte d'un système de réservation de vol, un événement pourrait être "Réserver Vol", déclenché par un utilisateur.

Scénarios d'Interaction

Les scénarios d'interaction dépeignent comment un objet évolue à travers divers états en réponse à des événements particuliers. Dans le cas du processus de réservation de vol, un scénario pourrait se dérouler de la manière suivante :

1. L'objet est dans l'état "En Attente".
2. L'utilisateur déclenche l'événement "Réserver Vol".
3. L'objet effectue la transition vers l'état "Confirmé".

Ces scénarios d'interaction mettent en lumière la séquence des états parcourus par un objet suite à des événements spécifiques. Ils fournissent une vision claire et compréhensible du

comportement dynamique de l'objet au fil du temps et des influences extérieures ou intérieures.

Diagramme d'état vs machine à états

Le **Diagramme d'État** est souvent employé pour modéliser le comportement d'un objet individuel, offrant une vue détaillée de la façon dont cet objet répond aux événements et évolue entre différents états. Il sert à représenter la dynamique d'un objet spécifique au fur et à mesure de son cycle de vie. Dans un diagramme d'état, on peut observer les états dans lesquels un objet peut se trouver, les transitions entre ces états, ainsi que les événements qui déclenchent ces transitions. Ce type de diagramme met en avant le comportement spécifique d'un objet en mettant l'accent sur ses changements d'état.

D'un autre côté, une **Machine à États** est une représentation plus complexe qui intègre plusieurs diagrammes d'état pour modéliser le comportement d'un système plus vaste. Contrairement au diagramme d'état qui se concentre sur un objet individuel, la machine à états peut décrire le comportement d'un système dans son ensemble, y compris la gestion des interactions entre plusieurs objets et leurs transitions d'état. Une machine à états est particulièrement utile lorsque vous voulez modéliser des interactions complexes et des scénarios où plusieurs objets interagissent et influencent mutuellement leurs états. Elle offre une vue globale du comportement dynamique d'un système, au-delà de la perspective limitée à un seul objet.

En somme, le diagramme d'état est plus spécifique et ciblé, idéal pour modéliser le comportement d'un objet particulier, tandis que la machine à états est plus holistique, adaptée pour capturer le comportement global d'un système impliquant plusieurs objets et leurs interactions.

Représentation visuelle

Symboles et composants des diagrammes d'états-transitions

Vous pouvez inclure de nombreuses formes différentes dans un diagramme états-transitions, surtout si vous décidez de l'associer à un autre diagramme. Voici la liste des formes les plus courantes que vous trouverez :

État composite

État qui intègre des sous-états. Voir l'exemple de **diagramme états-transitions d'une université** ci-dessous. Dans cet exemple, « Inscriptions » représente l'état composite, car il englobe plusieurs sous-états dans le processus d'inscription.

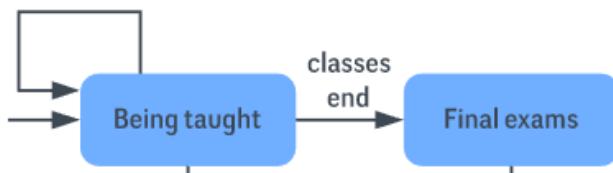
Pseudo-état choix

Losange qui indique un état dynamique avec des résultats potentiels variables.



Événement

Instance qui déclenche une transition. Son nom figure au-dessus de la flèche de transition applicable. Dans le cas présent, « fin des cours » est l'événement qui déclenche la fin de l'état « Enseigné actuellement » et le début de l'état « Examens finaux ».



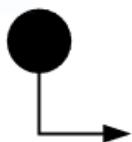
Point de sortie

Point auquel un objet quitte l'état composite ou l'automate, symbolisé par un cercle barré d'une croix. En règle générale, on l'utilise si le processus n'est pas terminé mais doit être quitté en raison d'une erreur ou d'un autre problème.



Premier état

Marqueur du premier état du processus, représenté par un cercle noir avec une flèche de transition.

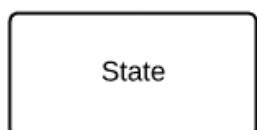


Garde

Condition booléenne qui autorise ou bloque une transition, inscrite au-dessus de la flèche de transition.

État

Rectangle aux coins arrondis qui indique la nature actuelle d'un objet.



Sous-état

État contenu dans la zone d'un état composite. Dans le [diagramme états-transitions de l'université](#) ci-dessous, « Ouvert aux inscriptions » est un sous-état du plus grand état composite intitulé « Inscriptions ».

Terminator

Cercle avec un point à l'intérieur, qui signifie qu'un processus est terminé.



Transition

Flèche allant d'un état à un autre et indiquant un changement d'état.

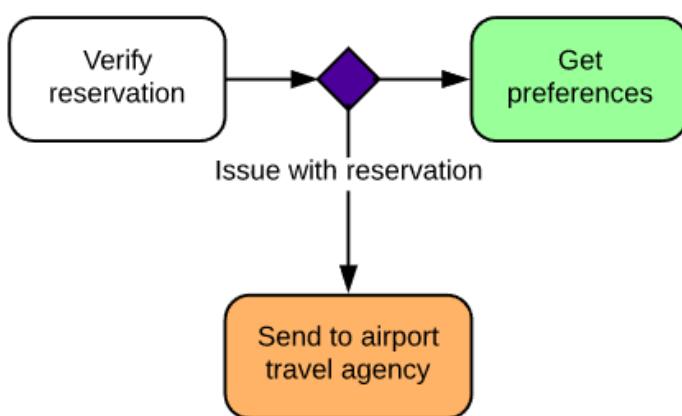


Comportement de transition

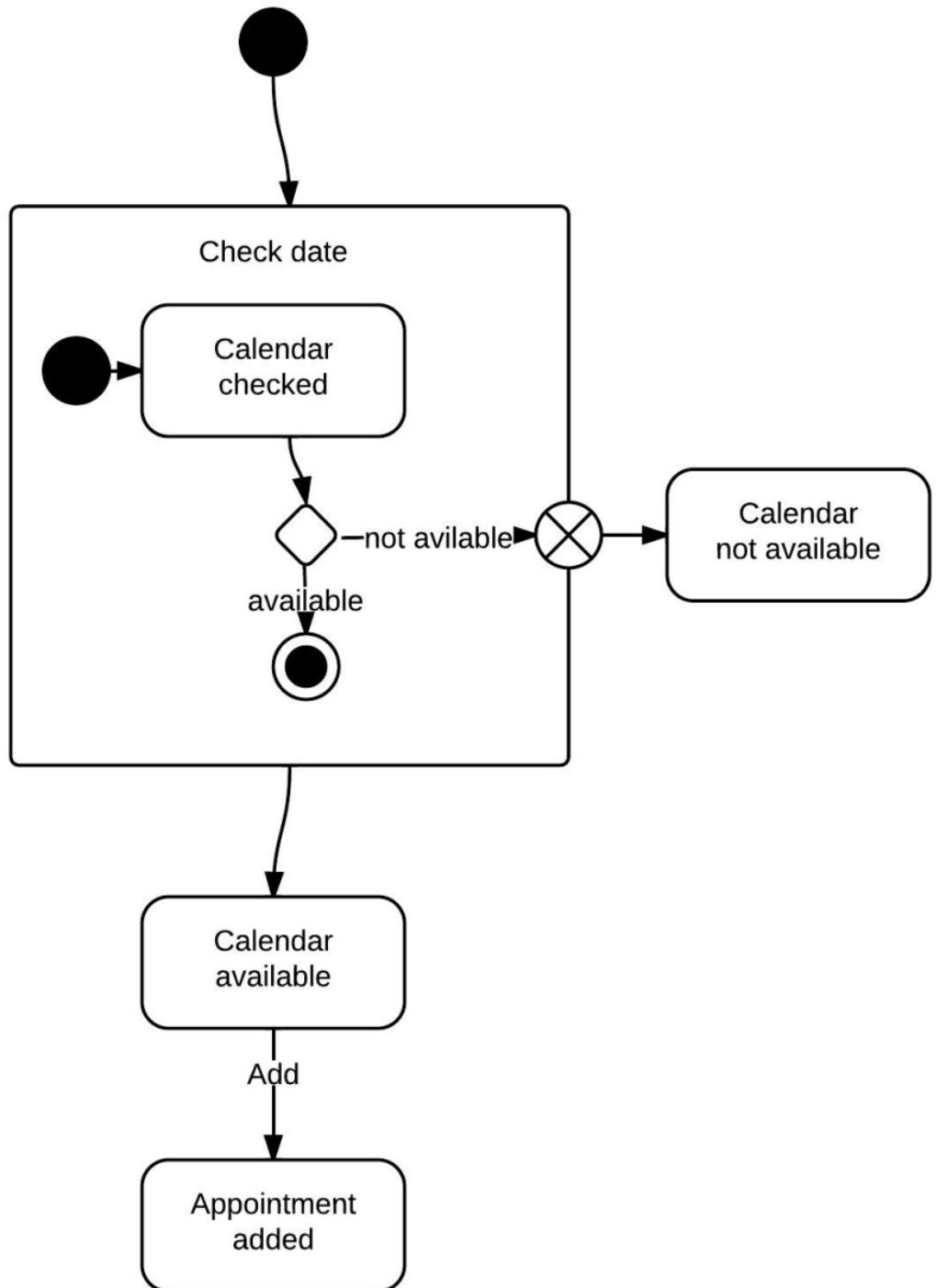
Comportement résultant de la transition d'un état, inscrit au-dessus de la flèche de transition.

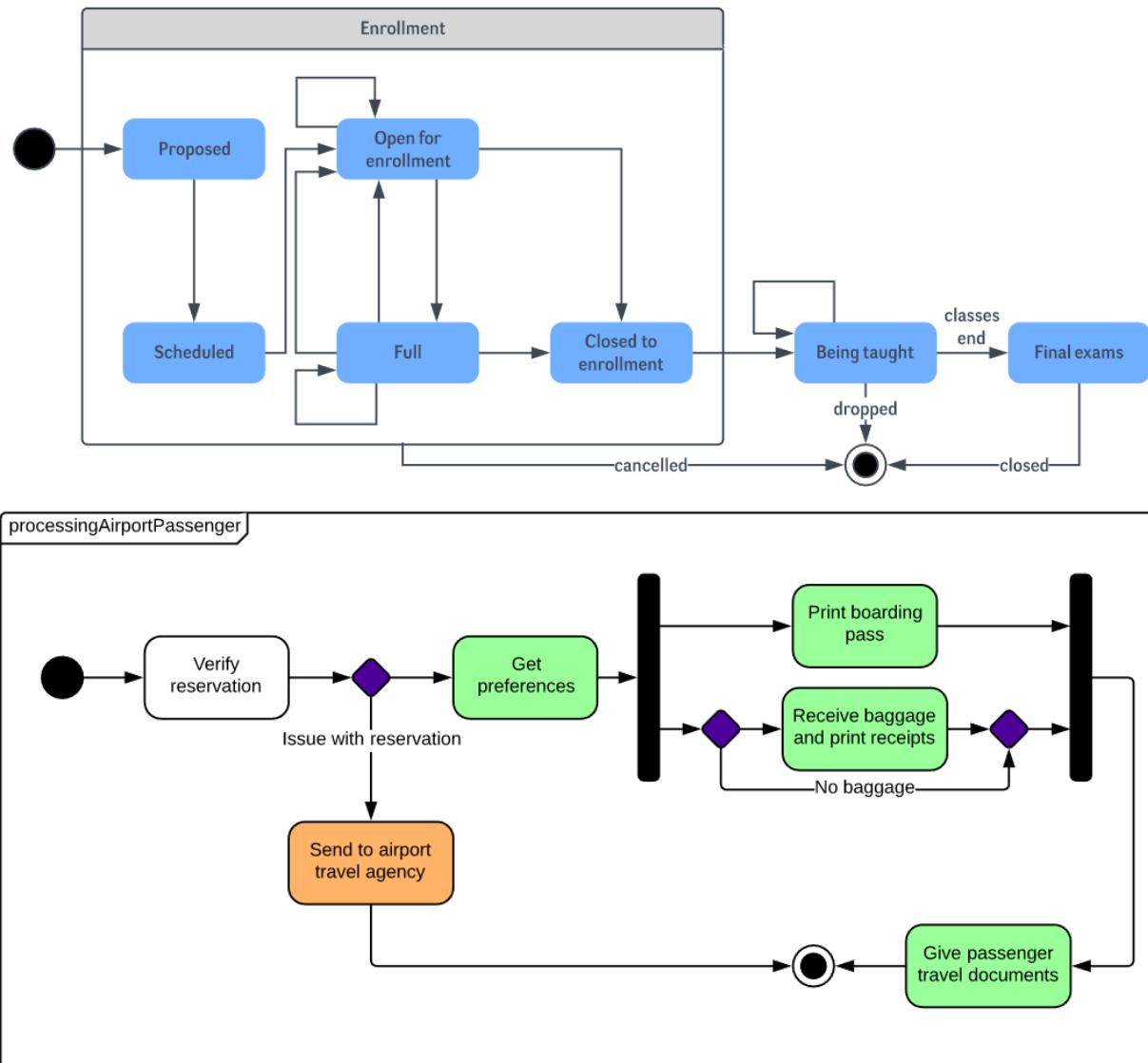
Déclencheur

Type de message qui déplace activement un objet d'un état à un autre, inscrit au-dessus de la flèche de transition. Dans cet exemple, « Problème avec la réservation » est l'élément déclencheur qui enverrait la personne à l'agence de voyage de l'aéroport au lieu de l'acheminer vers l'étape suivante du processus.



Exemples de diagramme Etats





Diagrammes de séquence

Représenter les communications avec et au sein du logiciel

- Représentation temporelle des interactions entre les objets
- Chronologie des messages échangés entre les objets et avec les acteurs

Eléments du diagramme de séquence :

- Acteurs
- Objets (instances)
- Messages (cas d'utilisation, appels d'opération)

Principe : Représentation graphique de la chronologie des échanges de messages avec le système ou au sein du système

- 'Vie' de chaque entité représentée verticalement

- Echanges de messages représentés horizontalement

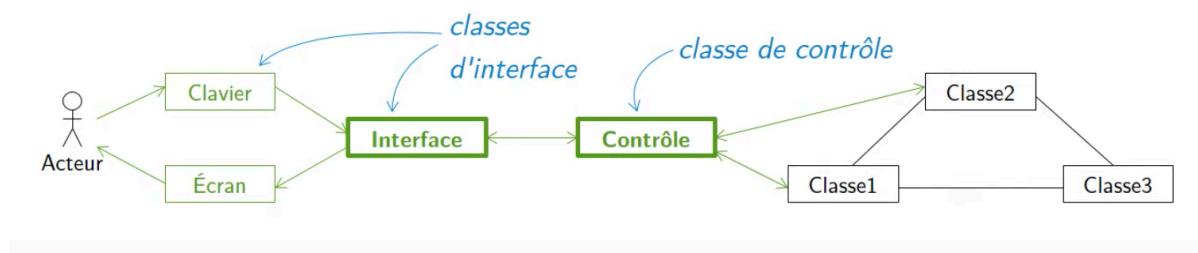
Problématique :



Le système est un ensemble d'objet, un acteur ne sait pas interagir directement avec des objets.

Donc la communication se fait via une interface (texte, web, boutons ...)

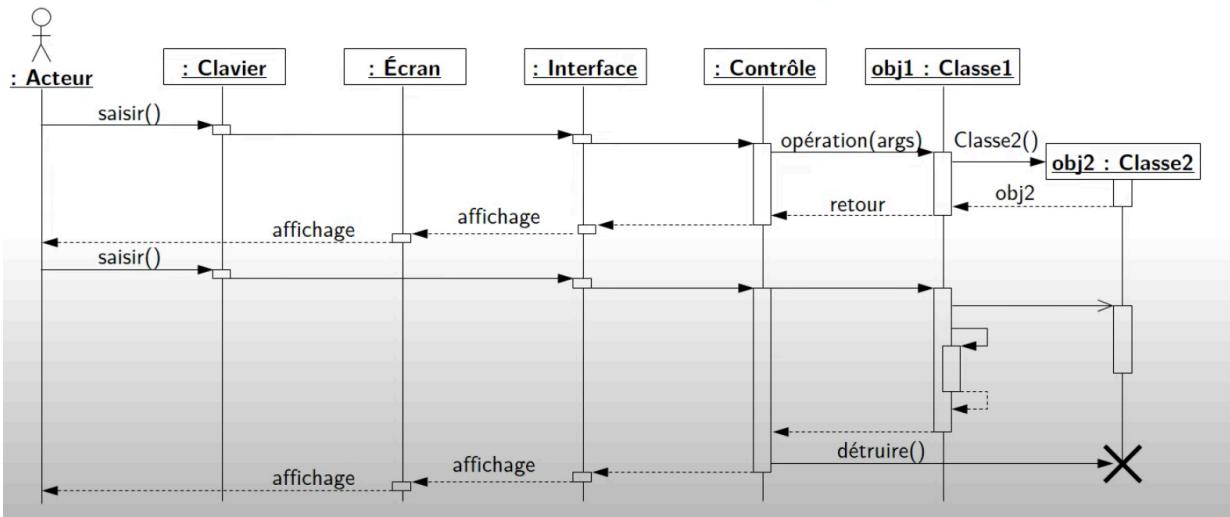
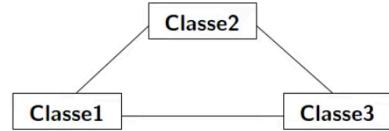
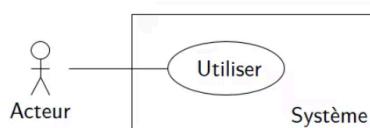
Solution :



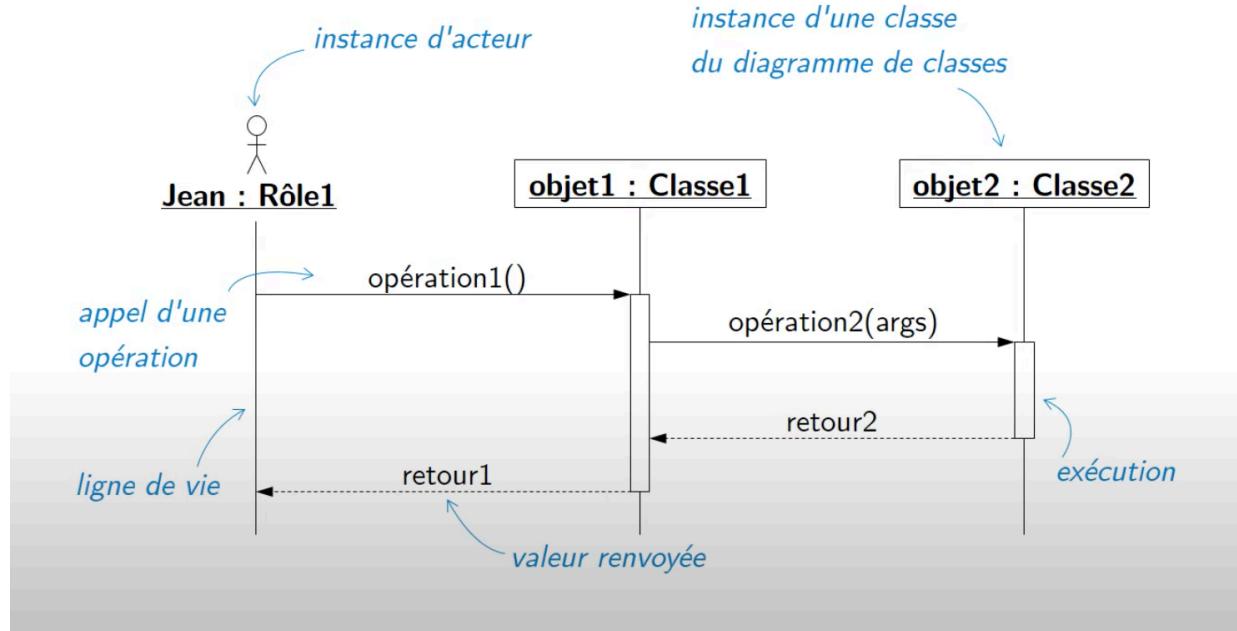
Création de classes de contrôle et de classes d'interface qui :

- gèrent les interactions avec les acteurs
- encapsulent le résultat des opérations

Exemple :



Eléments de base :



- trait plein : appel d'une opération
- trait pointillé : message de retour

Fragments

Les fragments sont des éléments clés dans un diagramme de séquence qui permettent de représenter des parties conditionnelles, optionnelles ou répétitives d'une séquence

d'interaction. Ils sont utilisés pour modéliser des scénarios plus complexes et pour montrer comment le flux d'exécution peut varier en fonction de certaines conditions. Les fragments ajoutent de la souplesse aux diagrammes de séquence, ce qui permet de capturer différentes voies de déroulement d'un scénario.

Fragments "alt" (Alternative)

Les fragments "alt" (alternative) modélisent différentes options d'exécution en fonction de conditions. Ils permettent de représenter des choix conditionnels dans une séquence d'interaction. Chaque branche du fragment "alt" correspond à une option possible, et seule l'une de ces branches sera exécutée, en fonction de la condition qui est vraie.

Fragments "opt" (Option)

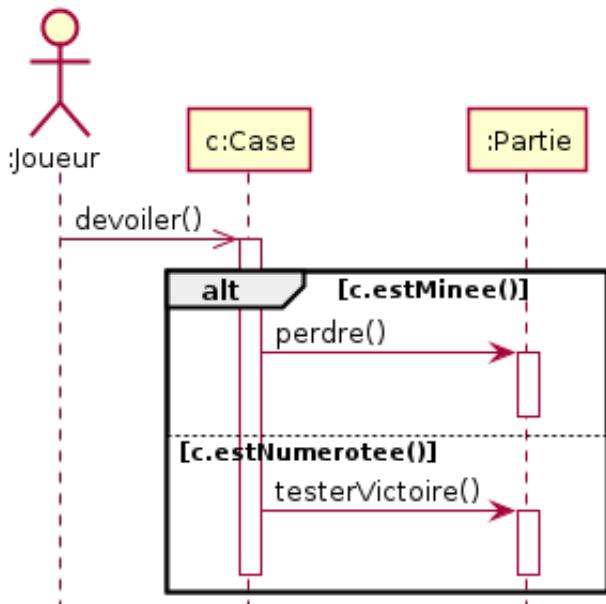
Les fragments "opt" (option) modélisent des parties optionnelles d'une séquence d'interaction. Ils indiquent qu'une séquence d'interaction peut ou non être exécutée, en fonction de la condition qui est vraie. Si la condition est satisfaite, la séquence optionnelle sera exécutée. Sinon, elle sera simplement ignorée.

Fragments "loop" (Boucle)

Les fragments "loop" (boucle) modélisent une séquence d'interaction répétée plusieurs fois tant qu'une condition est vraie. Ils permettent de modéliser des boucles dans le scénario d'interaction. La séquence contenue dans le fragment "loop" sera répétée jusqu'à ce que la condition spécifiée ne soit plus satisfaite.

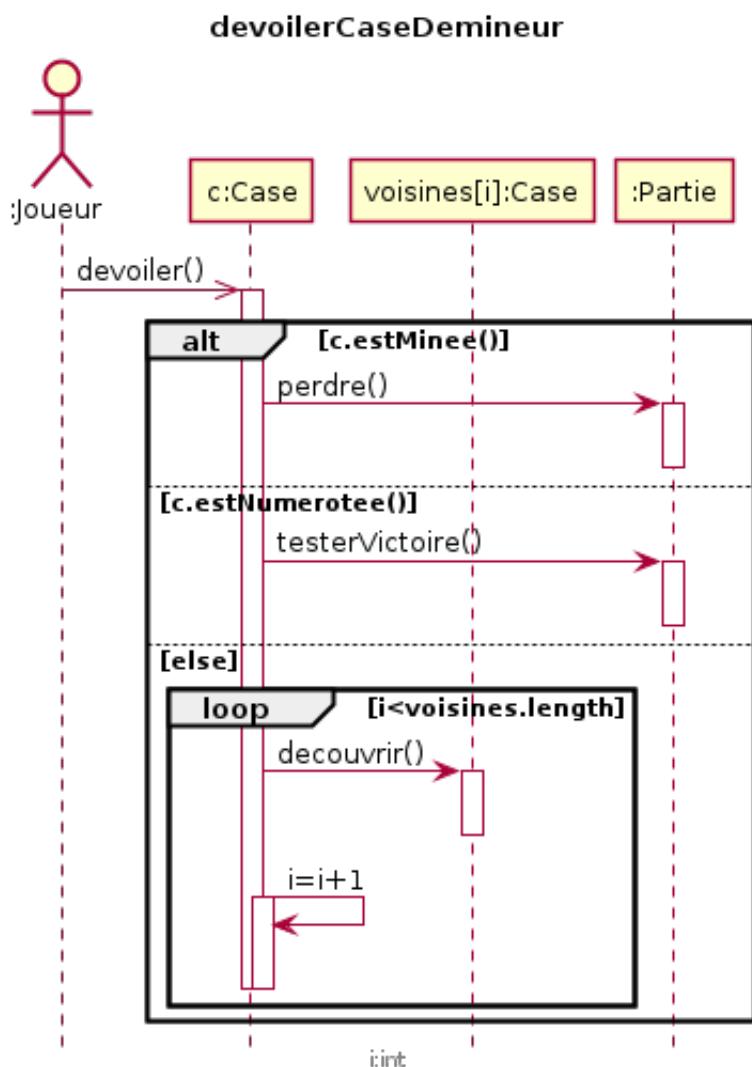
Fragments "par" (Parallèle)

Les fragments "par" (parallèle) modélisent des séquences d'interaction qui peuvent se produire simultanément. Ils représentent des interactions parallèles qui ne dépendent pas les unes des autres et qui peuvent être exécutées en même temps. Les fragments "par" sont utiles pour montrer comment différentes parties du système peuvent travailler de manière indépendante.

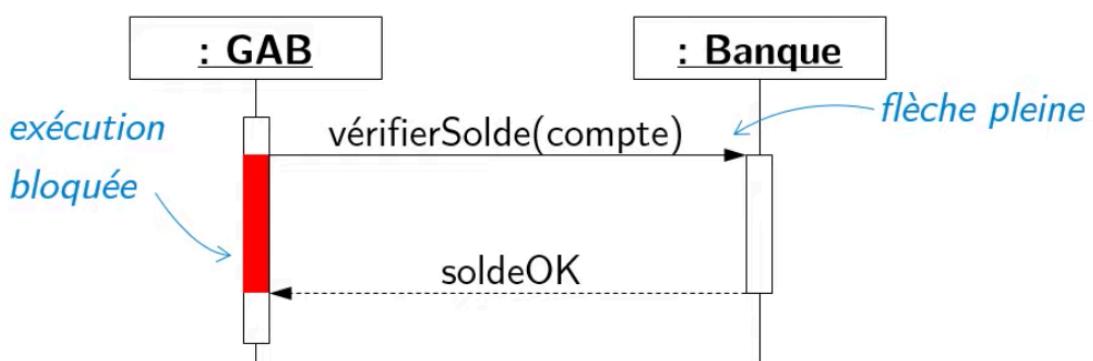


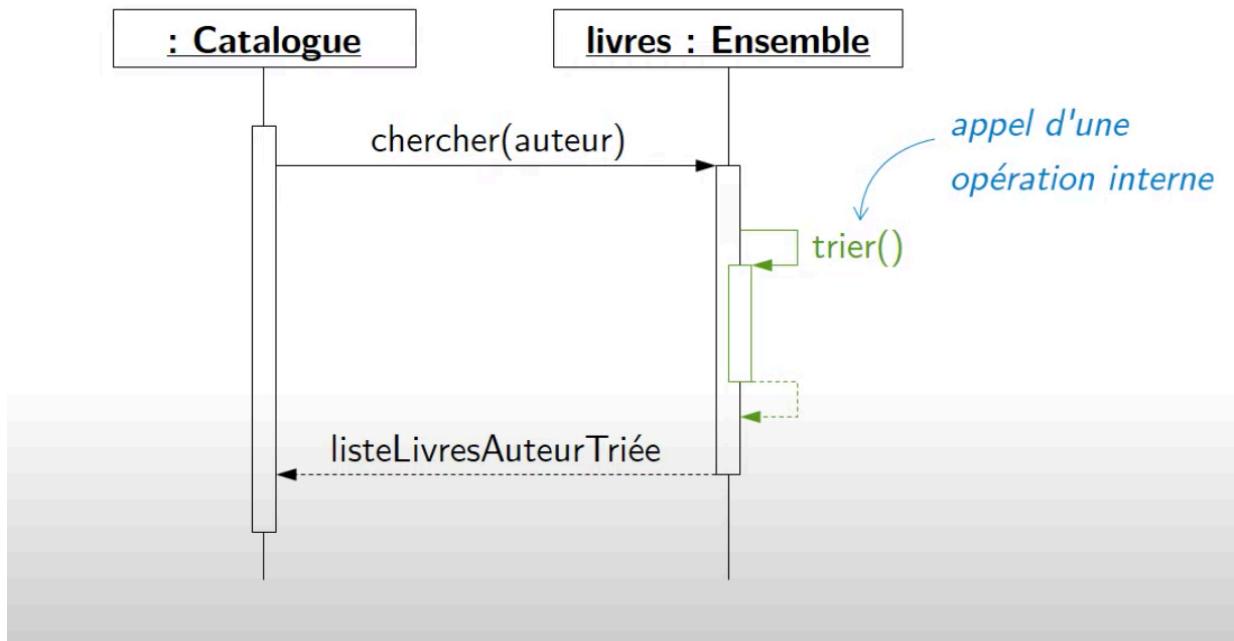
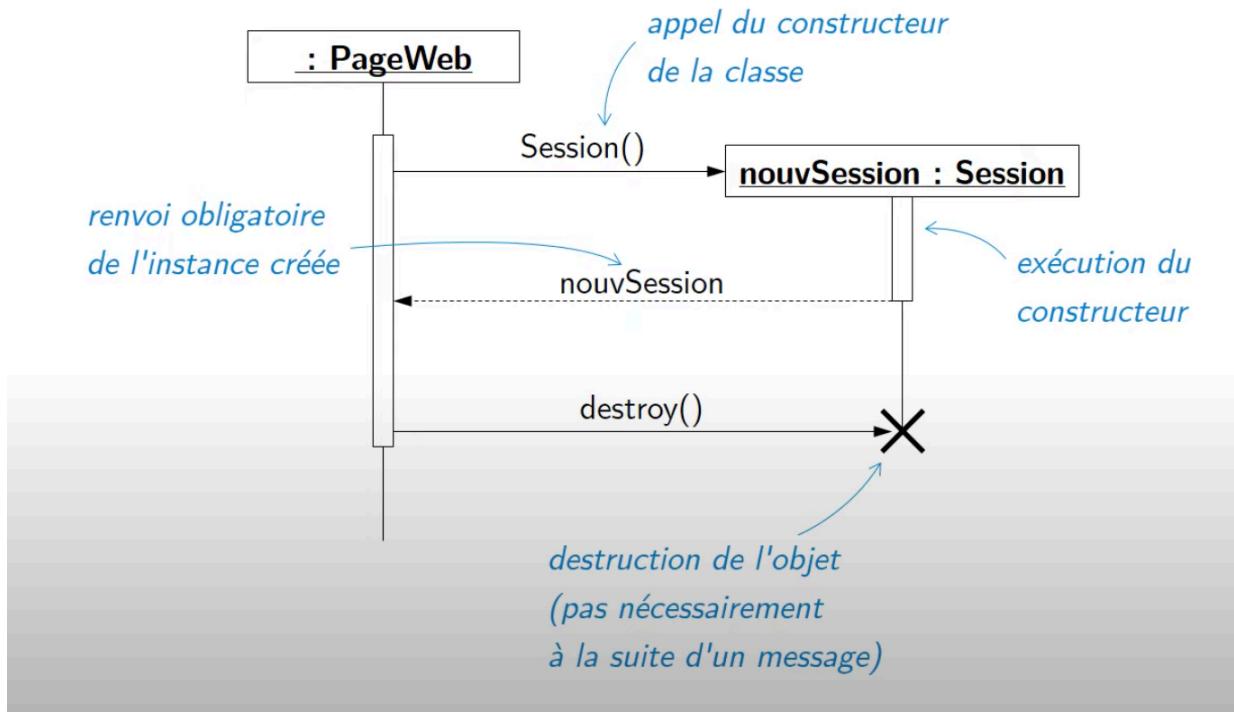
Combinaison de Fragments

Il est également possible de combiner plusieurs fragments pour modéliser des scénarios encore plus complexes. Par exemple, on peut avoir une branche "alt" à l'intérieur d'une boucle "loop" pour modéliser différentes options à chaque itération.



Autres Exemples :



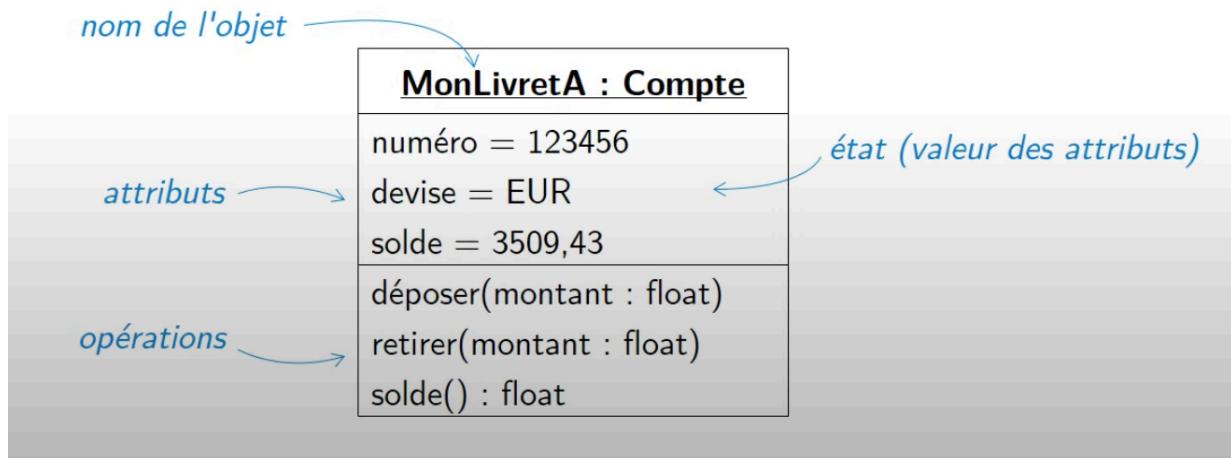


Diagrammes des classes

Conception orientée objet : Représentation du système comme un ensemble d'objets interagissant.

il s'agit d'une représentation de la structure interne du logiciel.

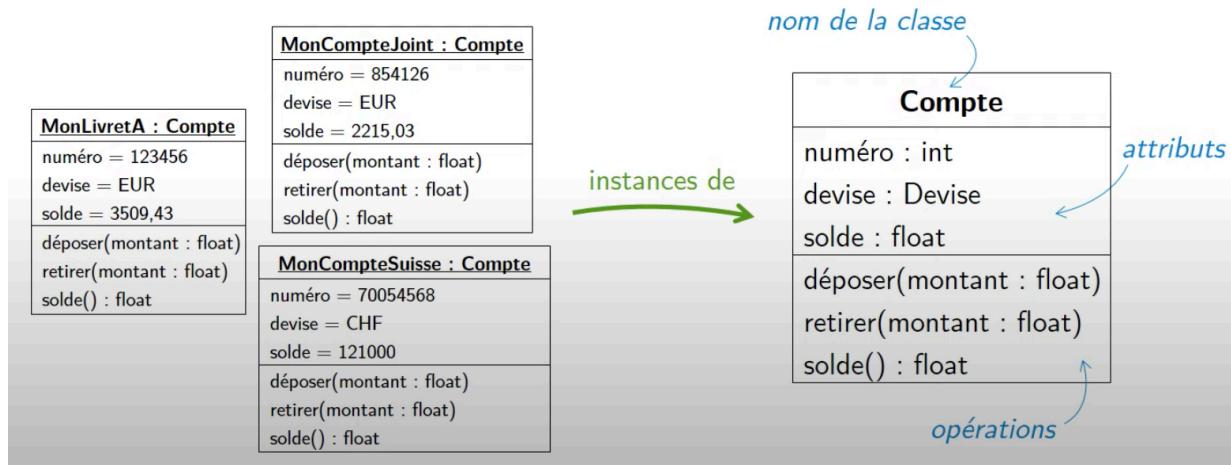
Rappel sur la définition d'un objet



Rappel sur la définition d'une classe

Regroupement d'objets de même nature (mêmes attributs + mêmes opérations)

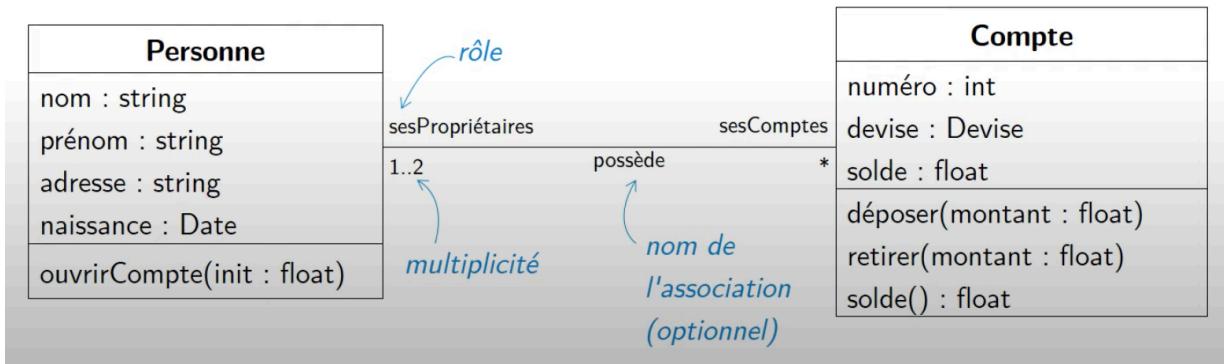
Objet = instance d'une classe



Relations entre classes

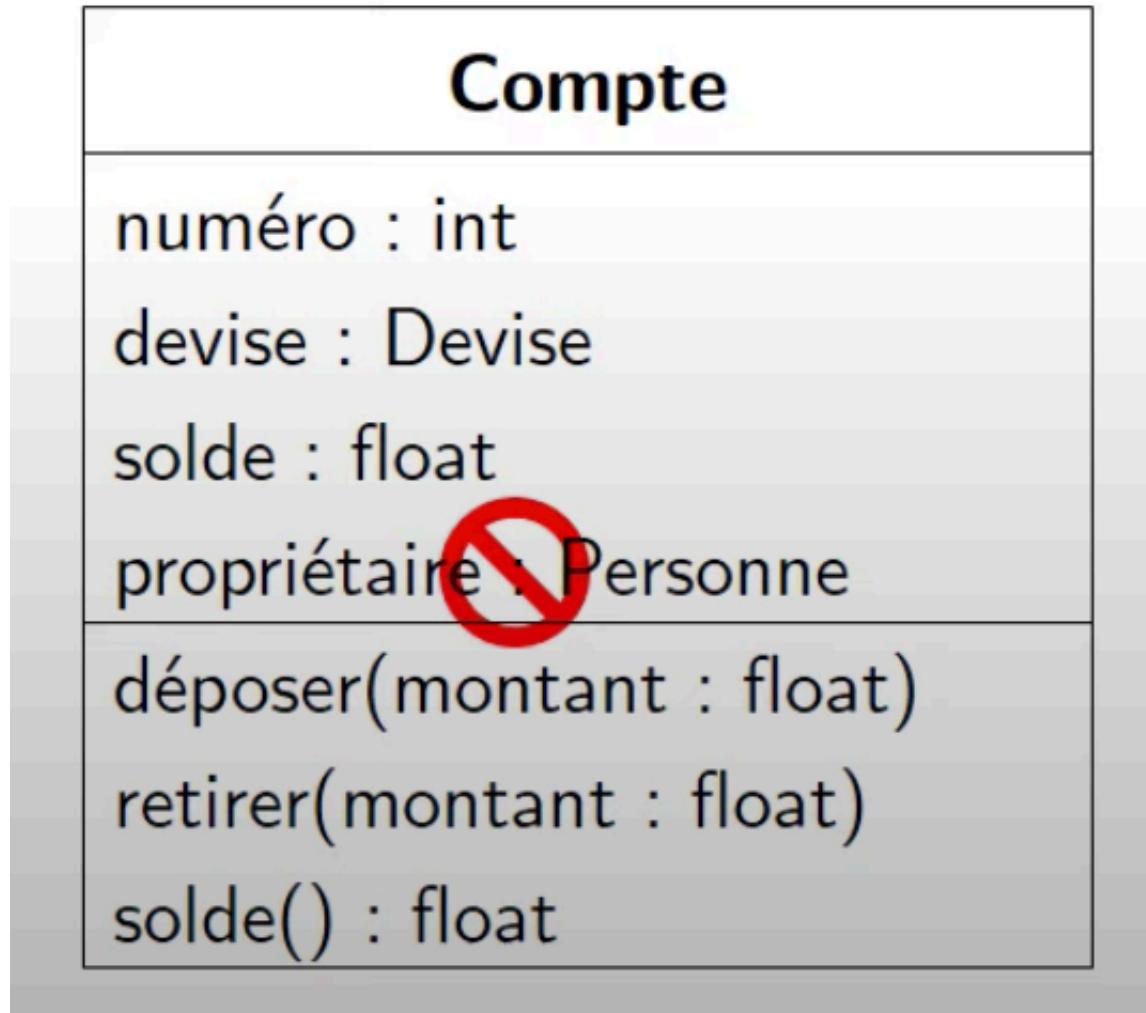
Rôle : Nomme l'extrémité d'une association, permet d'accéder aux objets liés par l'association à un objet donné.

Multiplicité : Contraint le nombre d'objets liés par l'association



Remarque

Pas d'attribut dont le type est une classe du diagramme



Multiplicités en pratique

Nombre d'objets de la classe B associés à un objet de la classe A



Exactement 1



Au plus 1 (0 ou 1)

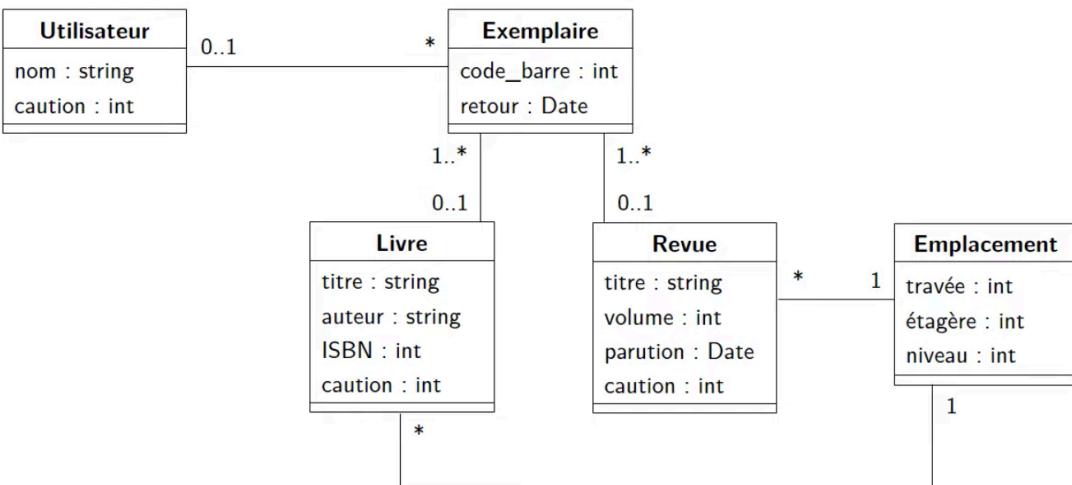


Au moins 1 (jamais 0)



0 ou plus

Exemple de diagramme



Notes : Si un exemplaire n'est pas emprunté, retour à la valeur *null*
 Un exemplaire est un exemplaire d'un livre ou d'une revue