Chapitre 3 : Un formalisme : UML

« Il existe deux manières de concevoir un logiciel. La première, c'est de le faire si simple qu'il est évident qu'il ne présente aucun problème. La seconde, c'est de le faire si compliqué qu'il ne présente aucun problème évident. La première méthode est de loin la plus complexe. » ***Antony R. Hoare, prix Turing 1980***

1. Objectifs du chapitre

Ce chapitre est consacré à la présentation du langage UML à travers ses principaux diagrammes. Il doit vous permettre de mesurer la puissance du langage pour décrire un logiciel malgré sa simplicité.
À l'issue de ce chapitre, vous serez en mesure de :

- écrire une modélisation partielle ou complète,
- relire une modélisation,
- échanger sur un problème ou sur une solution de modélisation avec d'autres développeurs.

1.1. Remarque

Le présent chapitre n'est pas une documentation UML. Il ne présente que les éléments importants et les plus utilisés dans les modélisations. Vous trouverez dans la littérature une documentation exhaustive sur UML, mais sachez que dans un développement agile, les éléments présentés ici sont suffisants.

2. Langage UML

UML est le langage de modélisation de logiciels :

- orienté objet,
- diagrammatique, c'est-à-dire essentiellement à base de diagrammes,
- débarrassé des contraintes syntaxiques.

Il possède les caractéristiques suivantes :

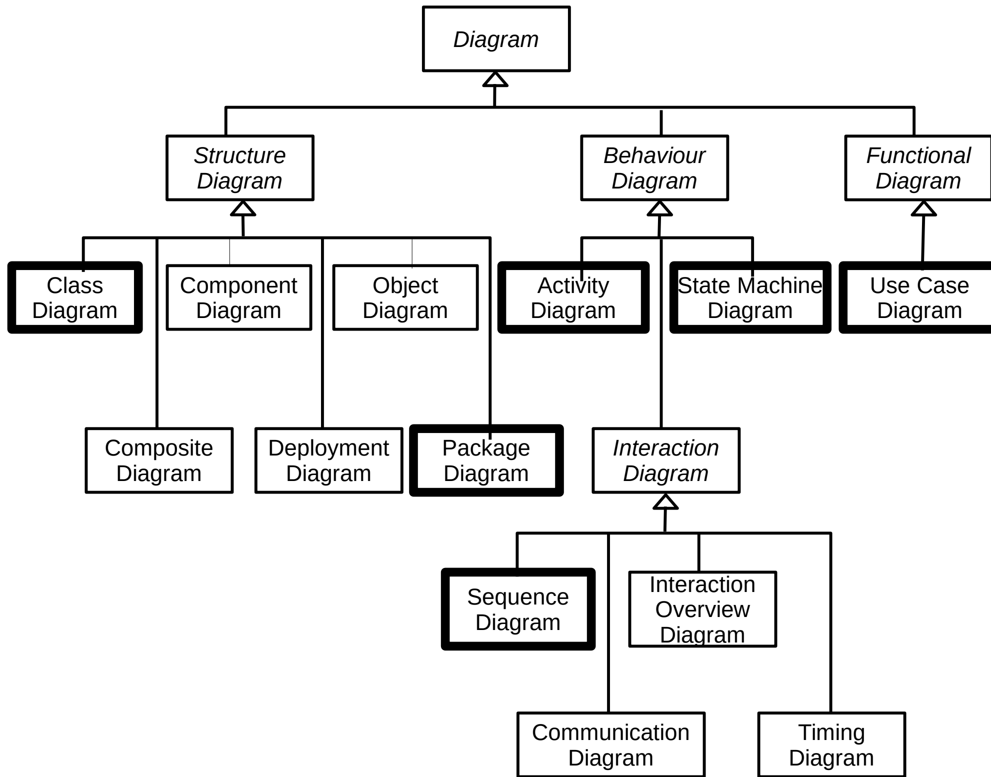
- normalisé par l'organisme OMG,
- indépendant des langages de programmation, mais ayant une forte correspondance avec les langages de programmation. On peut passer, assez directement, de UML à Java, C++, Php ou Python et le contraire.

2.1. Important

Il faut utiliser le langage UML pour parler des logiciels. Il est partagé par toute la communauté internationale des développeurs. Vous ne devez pas utiliser votre propre notation dont la sémantique n'est pas consensuelle. Elle peut alors mener à une incompréhension voire à un malentendu.

3. Vue d'ensemble des diagrammes

UML définit 13 diagrammes :



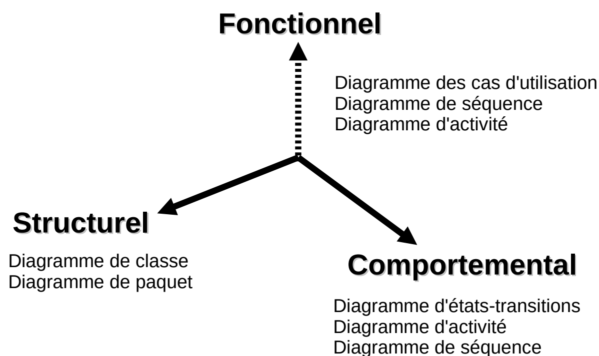
Parmi les 13 diagrammes, nous ne présentons dans la suite que les 6 diagrammes les plus utilisés :

- diagramme des cas d'utilisation
- diagramme de classe
- diagramme de paquet
- diagramme de séquence
- diagramme d'activité
- diagramme d'états-transitions

Les autres diagrammes sont soit très spécialisés (eg., diagramme de temps, diagramme de déploiement) soit une variante de ceux présentés (eg., diagramme de communication, diagramme d'interaction).

4. Points de vue sur la modélisation

Les six diagrammes permettent de modéliser 3 points de vue sur le système informatique à modéliser :



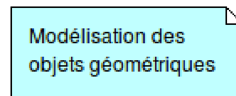
- axe fonctionnel : ce que fait le système.
- axe structurel : ce que le système utilise pour fonctionner.
- axe comportemental : comment le système fonctionne.

5. Éléments communs à tous les diagrammes

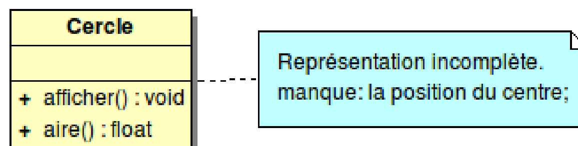
5.1. Les notes

Elles correspondent aux commentaires de modélisation. Ce sont des cartouches de texte qui permettent de traduire tout ce qui n'entre pas dans le champ d'UML. Une note peut être :

- libre ; la note porte sur le diagramme entier :



- associée à un élément quelconque ; la note est reliée à l'élément par un trait pointillé :



5.2. Les types primitifs

UML reconnaît tous les types primitifs des langages et ajoute certains types qui sont pourtant représentés par des classes à part entière dans certains langages mais qui se comportent comme des types primitifs :

- entier (*int*, *long*)
- réel (*float*, *double*)
- booléen (*bool*)
- caractère (*char*)
- chaîne de caractères (*string*)
- vide (*void*)
- temps (*time*)
- date (date)

6. Diagramme des cas d'utilisation

6.1. Intention

Le diagramme des cas d'utilisation doit permettre de :

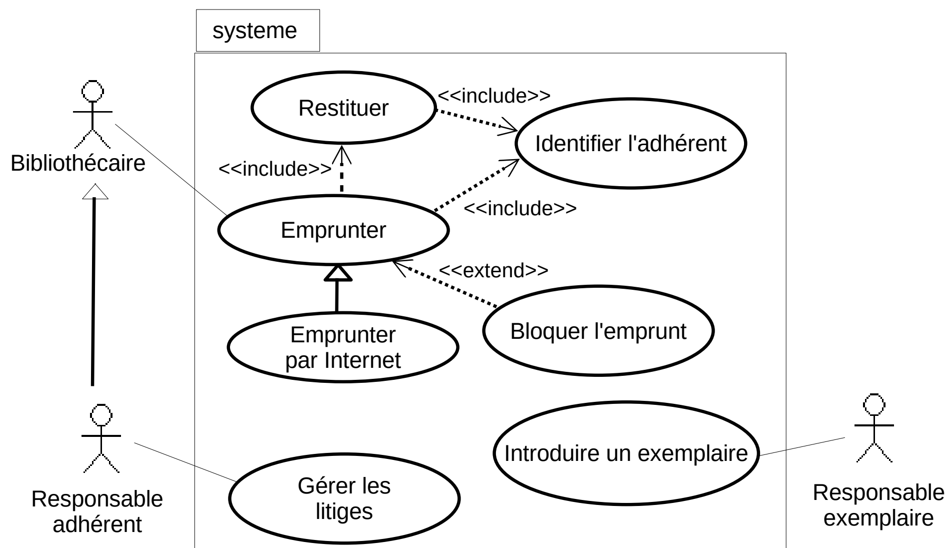
- Identifier les interactions entre les utilisateurs et le système de manière à déterminer :
  - Quel logiciel développer ?
  - Quels sont les utilisateurs ?
  - Quelles formes d'interaction de l'utilisateur avec le système ?
- Préciser les contours du futur système : ce qui est à l'intérieur devra être développé et ce qui est à l'extérieur devra être utilisé.

6.2. Contexte d'utilisation

Le diagramme des cas d'utilisation doit être utilisé dès la phase d'analyse des besoins de tout projet. Il fournit une vision **centrée sur les utilisateurs** qui est la raison d'être du futur logiciel. Il s'agit de lister les fonctionnalités mais en aucun cas à décrire la façon de les réaliser. Il ne faut pas confondre ce diagramme avec les diagrammes de séquence et d'activités. En particulier, il n'y a pas d'ordre entre les cas.
Il faut insister sur le fait qu'en dépit de son caractère apparemment trivial, le diagramme des cas d'utilisation est fondamental pour toutes les phases et tous les niveaux de la modélisation. Il est transversal, notamment il est utilisé aussi bien à la phase d'analyse qu'à la phase de tests de recette. Il permet notamment de se rappeler que ce n'est pas à l'utilisateur de s'adapter au logiciel mais bien au logiciel de s'adapter aux pratiques des utilisateurs.

6.3. Représentation UML

Exemple : gestion d'une bibliothèque

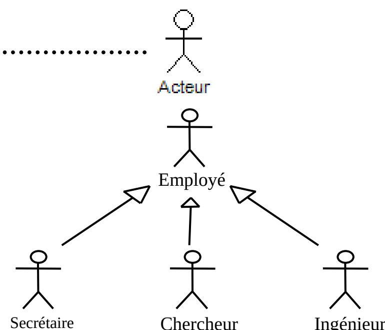


6.4. Éléments UML

6.4.1. Acteur

Il représente une entité **extérieure** au système en cours de modélisation.

- Il s'identifie par le rôle qu'il joue.
- Une même personne physique peut jouer le rôle de plusieurs acteurs.
- Notation UML d'un acteur :..
- Hiérarchie d'acteurs :..



- Un acteur n'est pas forcément une personne humaine.

- Exemples d’acteur :
  - utilisateurs finaux (rôles),
  - systèmes externes (coopérations),
  - événements relatifs aux temps (interruptions),
  - objets externes passifs (entités).

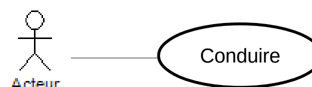
6.4.2. Cas d’utilisation

Un cas d’utilisation représente un ensemble d’actions utilisées pour satisfaire les besoins d’un acteur.

- Spécifié par une phrase intentionnelle (ie. avec verbe).
- Pourra induire un élément de l’interface graphique (IHM).
- Notation UML d’un cas :..

6.4.3. Relation entre acteur et cas

La relation exprime l’interaction existante entre un acteur et un cas d’utilisation.

- Notation UML : .... 

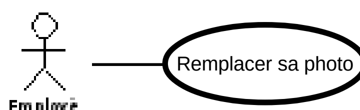
6.4.4. Relation entre cas

On distingue trois types de relation.

- Inclusion **obligatoire** d’un autre cas (flèche annotée par *include*) ....: <<include>> - - - - - - - - - - - - - - ->
- Extension **optionnelle** d’un cas (flèche annotée par *extend*) ....: <<extend>> - - - - - - - - - - - - - ->
- **Spécialisation** : relation de spécialisation d’un cas par un autre (flèche héritage) :

6.4.5. Documentation associée

Le diagramme UML n’explicite pas toutes les informations nécessaires à la description d’un cas d’utilisation. Il peut être nécessaire d’adjoindre à chaque cas une documentation textuelle qui peut prendre la forme du tableau ci-dessous. Considérons un exemple où un employé souhaite changer sa photo stockée dans l’annuaire de son organisation :



La description peut pendre la forme suivante :

Nom	Le nom décrit en une phrase intentionnelle le cas d’utilisation. p. ex. « *Remplacer sa photo* »
Acteur	p. ex. « *Employé* »
Description	Description plus complète que le nom. p. ex. « *Un employé souhaite changer sa photo stockée dans l’annuaire de son organisation* »
Pre-conditions	Quelles conditions doivent être vérifiées avant l’exécution de ce cas ? p. ex. « *L’employé est référencé dans le système* »
Déclencheur	Quel événement déclenche ce cas ? p. ex. « *L’employé a appuyé sur le bouton changer sa photo* »

Scénario nominal	Décrit la liste des étapes à enchaîner quand tout fonctionne correctement. p. ex. « 1. *Le système communique un formulaire d'identification.* » « 2. *L'employé renseigne les champs et valide le tout.* » « 3. *Le système transmet la photo de l'employé.* » « 4. *L'employé sélectionne une autre photo qu'il transmet au système.* » « 5. *Le système remplace l'ancienne photo par la nouvelle.* » « 6. *Le système confirme le succès de l'opération.* »
Scénarios alternatifs	Décrit les scénarios alternatifs possibles mais aussi les cas d'exception. p. ex. « 1 à 4. *L'employé annule l'opération, le cas s'arrête.* » « 3a. *L'employé n'est pas connu du système : retour au 1.* » « 4.1. *Le système ne reconnaît pas le format de l'image transmise par l'employé. Le système en informe l'employé et retour au 3 ?* »
Problèmes ouverts	p. ex. « *Faut-il se préoccuper de la taille des images transmises ?* »

7. Diagramme de classe

7.1. Intention

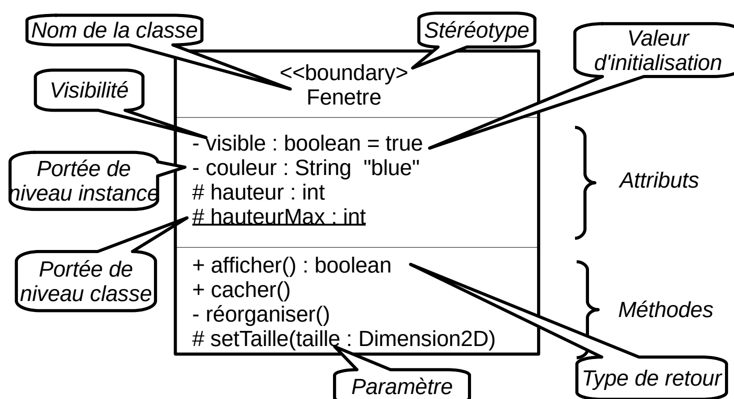
Ce diagramme est le plus utilisé. Il vise à décrire la structure statique du système en termes de classes et de relations entre ces classes.

7.2. Contexte d'utilisation

Il se retrouve à toutes les étapes de la spécification et de la conception. Il faut faire autant de diagrammes de classes que nécessaire pour garder une lisibilité suffisante.

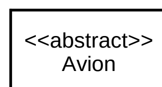
7.3. Représentation UML

7.3.1. Classe



7.3.2. Classe abstraite

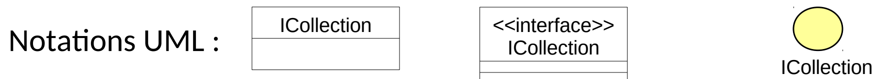
Notation UML :



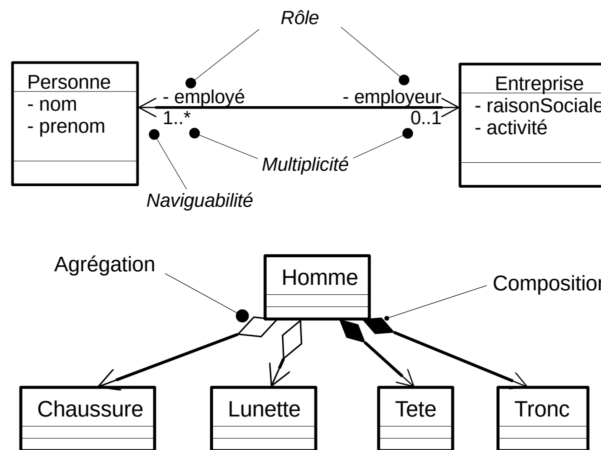
7.3.3. Interface

Convention : les noms des interfaces qui jouent le rôle de type sont souvent suffixés par 'able' (`Printable`, `Comparable`) ou préfixés par le caractère I (`ICollection`).

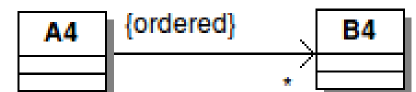
Notations UML :



7.3.4. Relation

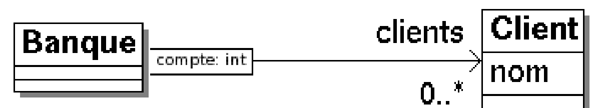


7.4. Association ordonnée



L'association correspond à une liste **ordonnée** d'éléments.

7.5. Association qualifiée



Le qualificateur (ici `compte`) permet la consultation d'une instance (ici `Client`) ou d'un sous-ensemble d'instances par une clé. Ainsi, l'accès à un client se fait par `this.clients[1079]` où `1079` est le numéro de compte du client et pas l'indice dans la liste. Cela se traduit typiquement par un tableau associatif ou un dictionnaire dans un langage de programmation.

7.6. Remarques

1. Un diagramme peut être plus ou moins détaillé (p. ex. une classe peut ne comporter que le nom sans autre élément).
2. Pour éviter toute polysémie sur le mot Interface, une classe qui fera partie de l'interface graphique sera stéréotypée <<`boundary`>> ou sera représentée avec l'icône suivant :

8. Diagramme de paquet

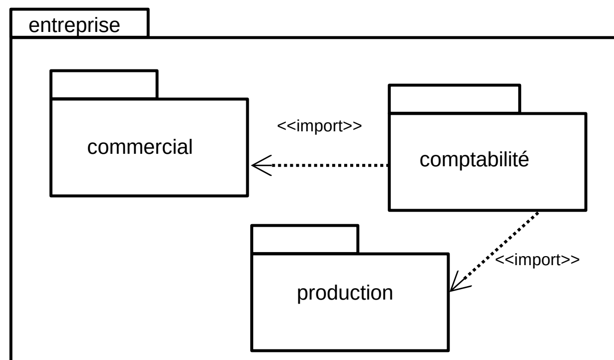
8.1. Intention

Ce diagramme fournit une **vision logique** de l'organisation du projet.

- Diviser le projet global en sous-parties avec des responsabilités différentes.
- Attribuer les paquets à des équipes de développement différentes.

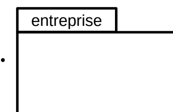
Un paquet permet le regroupement des éléments de modélisation. Il définit un espace de noms où les noms des identificateurs sont locaux aux paquets et n'entrent pas en conflit avec les noms des autres paquets.

8.2. Représentation UML



8.3. Éléments UML

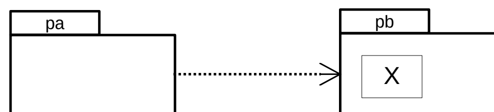
8.3.1. Paquet

Notation UML : un rectangle avec le nom du paquet... 

Convention : les noms des paquets sont écrits en snakecase. Ils peuvent être mis dans l'onglet ou dans le rectangle. Pour rappel, en Java, le nom correspond à l'adresse Web du site de l'équipe de développement à l'envers, p. ex. `fr.ensicaen.ecole.projet.paquet1`.
Les paquets ont une traduction directe dans le code : **package** en Java, **namespace** en C++ et C#.

8.3.2. Dépendance entre paquets

Un paquet B dépend d'un paquet A si au moins des éléments de B dépend d'un élément de A. Il est préférable d'éviter les dépendances circulaires.



- En Java, la dépendance se marque par le mot clé `import`. e.g. `import pb.X; X a = new X();`

8.3.3. Hiérarchie de paquets

Les paquets peuvent contenir d'autres paquets, de la même façon qu'un dossier peut contenir des sous-dossiers.

9. Diagramme de séquence

9.1. Intention

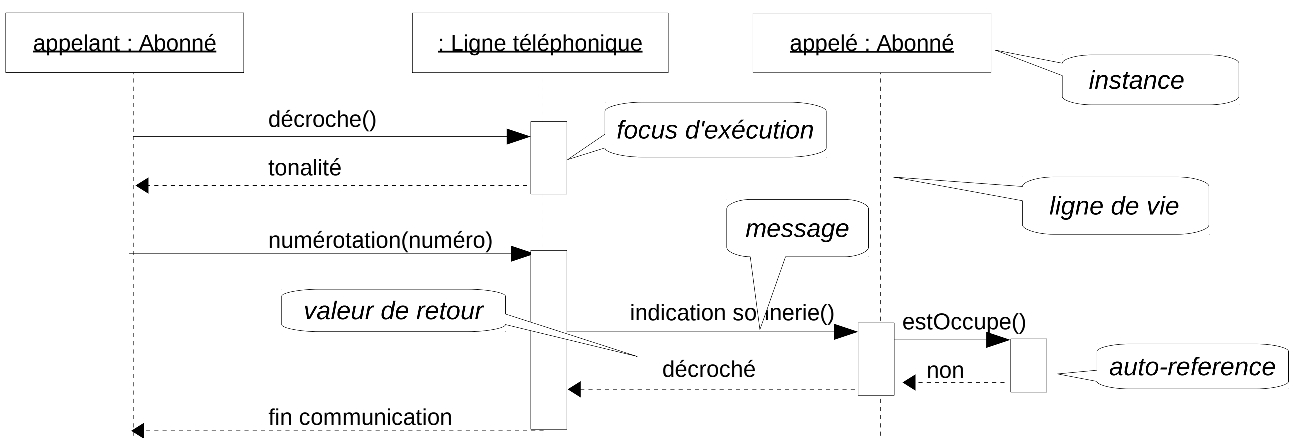
Ce diagramme décrit une séquence d'interaction entre plusieurs objets, dans un contexte d'exécution donné du système.

9.2. Contexte d'utilisation

Il est souvent utilisé pour décrire un **cas d'utilisation** ou une **activité** en se focalisant sur l'enchaînement des messages entre objets pour réaliser un scénario correspondant au cas d'utilisation ou à l'activité. De ce fait, on distingue deux types de diagramme de séquence :

- Le diagramme de séquence système, où on s'intéresse aux messages échangés entre les acteurs et le système vu comme un objet unique.
- Le diagramme de séquence « objet », où on s'intéresse à la communication entre plusieurs objets faisait partie du système.

9.3. Représentation UML



9.4. Éléments UML

9.4.1. Les instances

Chaque instance est positionnée de la gauche vers la droite.

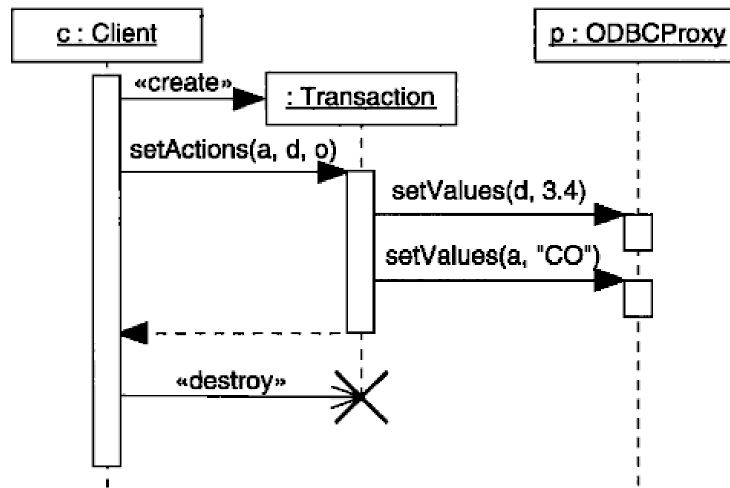
9.4.2. La ligne de vie

Elle est représentée par une ligne verticale en pointillée qui prolonge la représentation d'une instance.

9.4.3. Les messages

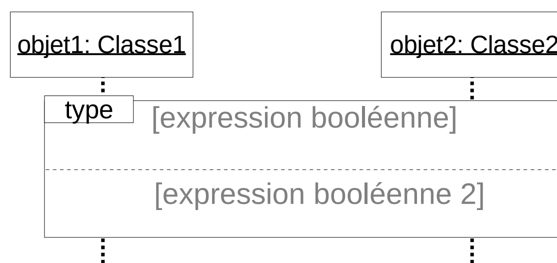
- Types de messages :
  - Message de type indéterminé (synchronisé ou non).........................
  - Message asynchrone pour les systèmes concurrents........................
  - Message synchrone..
  - Message réflexif...
  - Message de retour d'une opération...
- Messages particuliers :

- Création d'objet : faire pointer le message création sur le rectangle symbolisant l'objet ou utiliser le stéréotype <<`create`>> sur le message.
- Destruction d'objet : interrompre la ligne de vie par une croix ou utiliser le stéréotype <<`destroy`>> sur le message.



9.5. Fragments combinés

Un bloc auquel on affecte une structure de contrôle.



Principaux types de structure de contrôle :

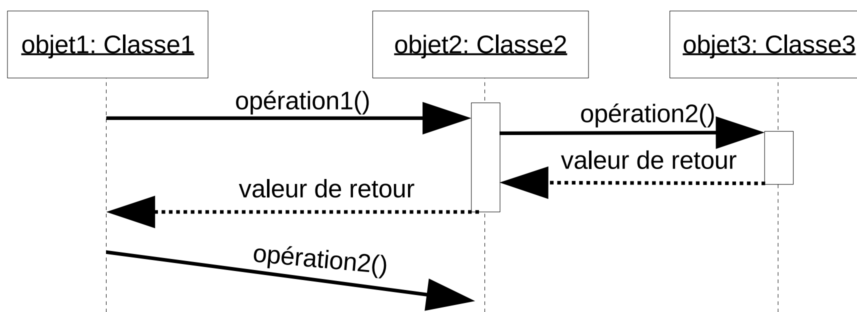
- opt : fragment parcouru si une condition est vérifiée.
- loop : répétition du fragment tant que la condition est vérifiée.
- alt : une alternative de type si … alors… sinon.

9.6. Temporalité

Le focus d'exécution représente une période d'activation correspondant au temps pendant lequel l'objet effectue l'action :

- matérialisé par un rectangle plein sur la ligne de vie,
- peut-être source de nouveaux messages à l'intérieur.

Si le délai de transmission est non négligeable, il sera représenté par une flèche oblique (e.g. `operation2`).



10. Diagramme d'activité

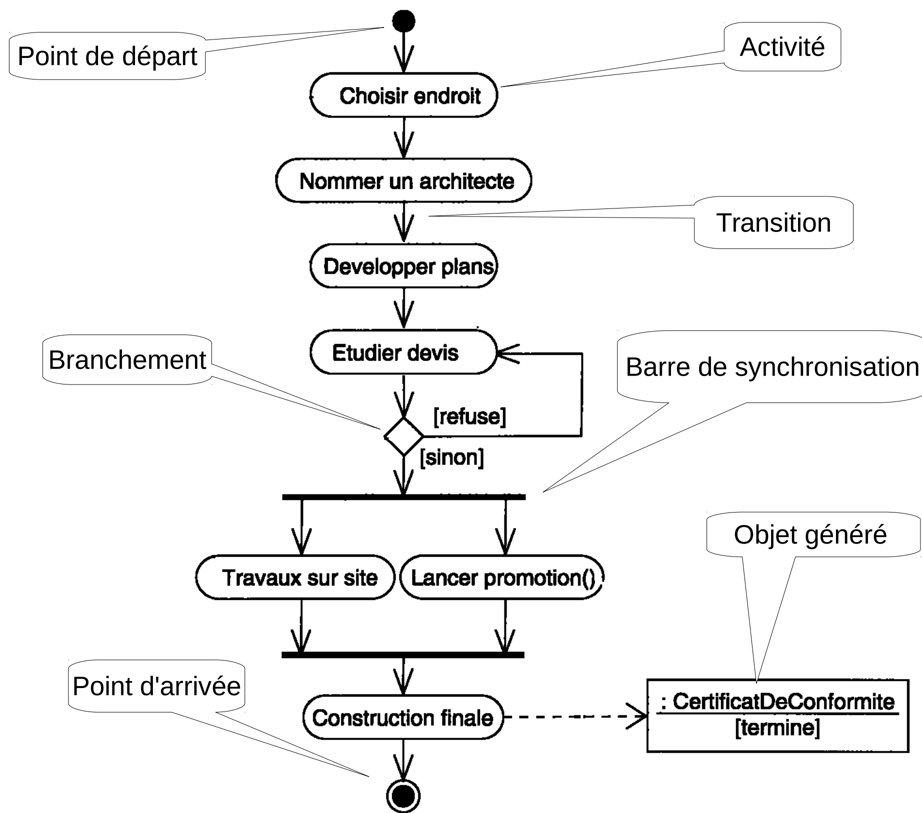
10.1. Intention

Ce diagramme permet de modéliser un processus sous la forme d'un « **flot de contrôle** ».

10.2. Contexte d'utilisation

- Détailler un **cas d'utilisation** (ie. détailler une fonctionnalité).
- Décrire un **algorithme** (p. ex. détailler une méthode d'une classe).
- Modéliser le flux de travail (*workflow*) d'un processus métier.

10.3. Représentation UML



10.4. Éléments UML

10.4.1. Activité

C'est une étape particulière dans l'exécution du processus à modéliser. Elle correspond à une opération **interruptible** qui **dure** un certain temps dans un état particulier.

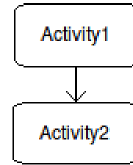
- Notation UML :
  - Représentée par un cartouche .... ouvrir la porte   i:=i+1
  - Un point d'entrée...
  - Un ou plusieurs points de sortie...
- Une activité peut contenir :

- L'appel d'une opération.
- L'envoi d'un signal.
- La création/destruction d'un objet.
- Un simple calcul.

10.4.2. Transition

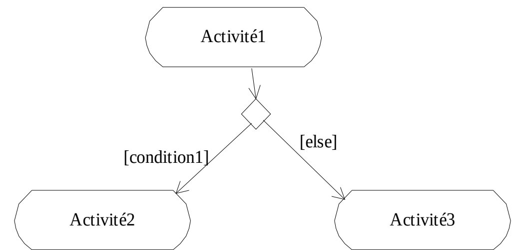
Passage automatique d'une activité à une autre.

- Notation UML :
  - représentée par une flèche ouverte sans nom.



10.4.3. Branchement conditionnel

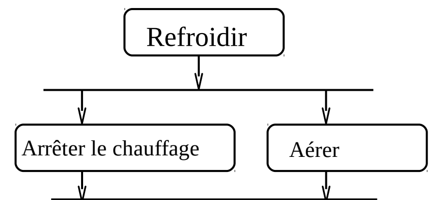
- La condition est représentée par un losange (avec l'expression booléenne à côté si nécessaire).
- Une garde (condition exprimée par une expression booléenne) est mise sur la transition d'un branchement.
  - Exemples : [t = 15 s] ; [code incorrect & essai > 3]



10.4.4. Synchronisation

La barre de synchronisation permet d'ouvrir et de fermer des branches parallèles au sein d'un flot d'activités.

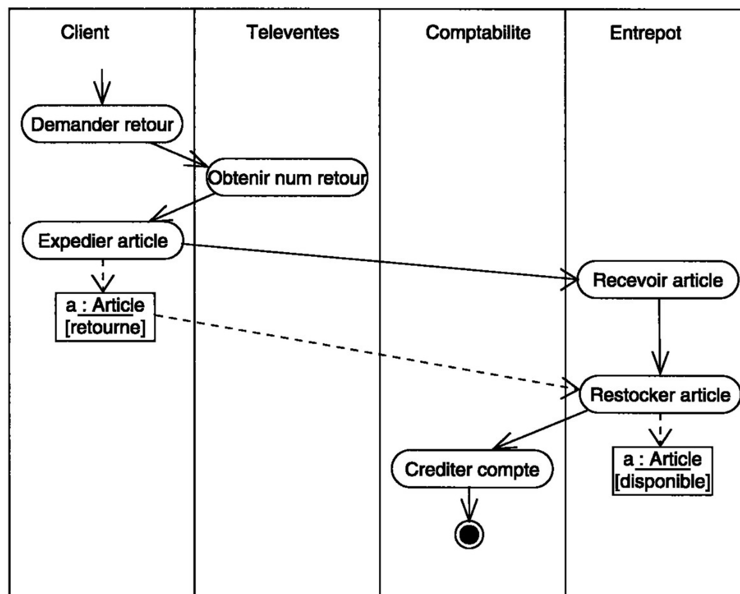
- Notation UML :
  - séparation parallèle : une barre.
  - fusion : une barre.



10.4.5. Les travées (ou couloirs)

Dans ce cas, le modèle est agencé selon des travées verticales correspondant aux classes.

- Exemple : Retour et remboursement d'une marchandise.



11. Diagramme d’états-transitions

11.1. Intention

Ce diagramme permet de décrire les différents états que peuvent prendre une classe en fonction des opérations qui lui sont appliquées et les événements qui provoquent le passage d’un état à un autre.
Il est important de noter qu’un diagramme d’états-transitions est lié à **une classe et une seule**.

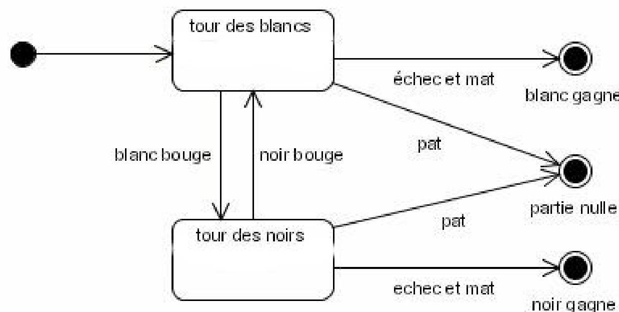
11.2. Contexte d’utilisation

L’intention du digramme est de :
- détailler le comportement complexe d’une classe,
- décrire le comportement d’un objet à travers plusieurs cas d’utilisation.

11.3. Éléments UML

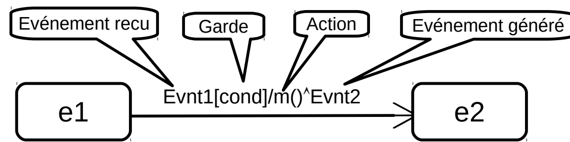
11.3.1. État

Un état est défini par des valeurs d’attributs de la classe. Deux états différents doivent diverger d’au moins une valeur d’attribut.
- Un état décrit une condition ou une situation qui survient au cours de la vie d’un objet :
  - pendant laquelle l’objet satisfait à certaines conditions,
  - pendant laquelle l’objet exécute une activité,
  - pendant laquelle un objet attend un événement.
- Notation UML
  - Un rectangle aux coins arrondis.
  - Un ou plusieurs points d’entrée et un ou plusieurs points de sortie.

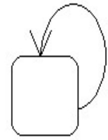


11.3.2. Transition

Une transition est un passage **instantané** d’un état vers un autre déclenché par un événement.
Une transition comporte 5 parties :
- L’état source.
- L’événement déclencheur (facultatif).
- La condition de garde de déclenchement (facultatif) qui permet de valider la transition.
- L’action (facultatif) qui est la méthode effectuée lors de la transition.
- L’état cible.

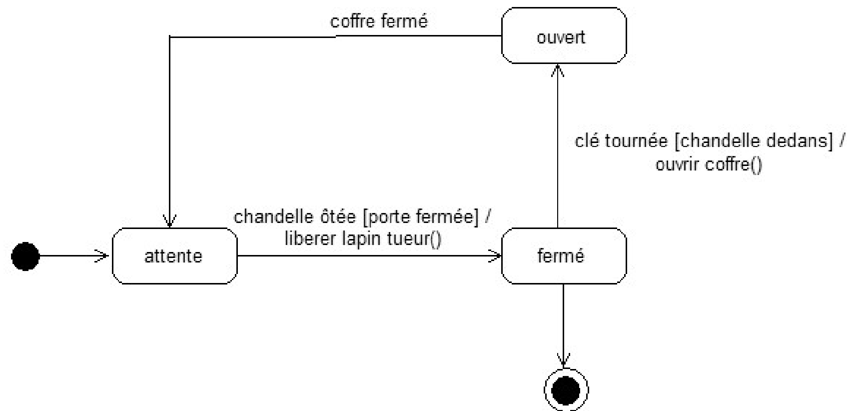


- L'auto-transition représente un événement qui fait rester dans l e même état :



11.3.3. Événement

Un événement est un stimulus qui peut déclencher une transition d'état. Il se produit à un instant donné. Il n'a pas de durée.



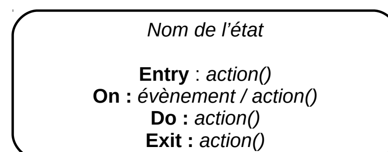
On distingue 4 types d'événement :

- Un événement de type ***signal*** représente un signal asynchrone envoyé par un composant extérieur.
- Un événement de type ***appel*** représente le déclenchement d'une méthode synchrone d'une classe.
- Un événement ***temporel*** représente l'écoulement du temps (`after`).
- Un événement de ***modification*** représente un changement d'état (`when`).

11.3.4. Modélisation d'un état

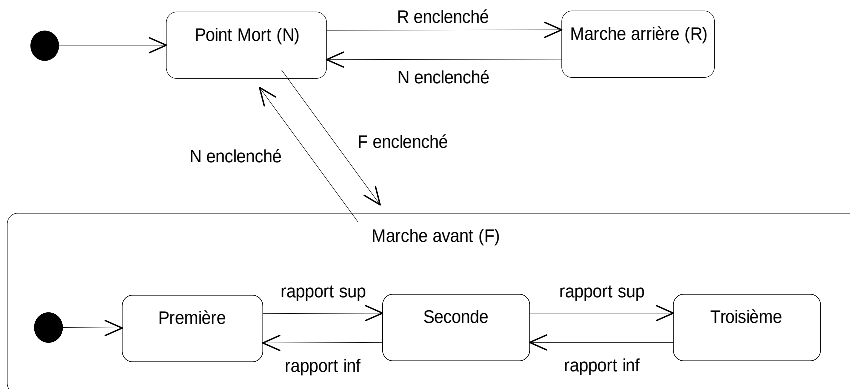
Une **action** correspond typiquement à une opération de la classe mais qui ne peut pas être interrompue durant son exécution.

- Action d'entrée : `entry`
- Action d'entrée par un événement donné : `on` *`evenement`*
- Action de sortie : `exit`
- Activité : `do`
  - Le séquencement est possible : `do / operation1(); operation2(); operation3()`.



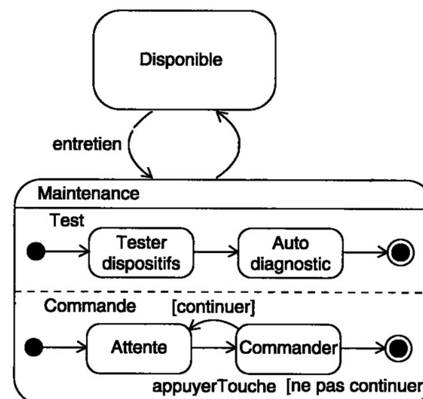
11.4. Sous-état

Un sous-état permet une modélisation hiérarchique d'états complexes (super-état raffiné en sous-états). Par exemple, ci-dessous, l'état « marche avant » est en fait un état complexe.



11.4.1. Sous-états concurrents

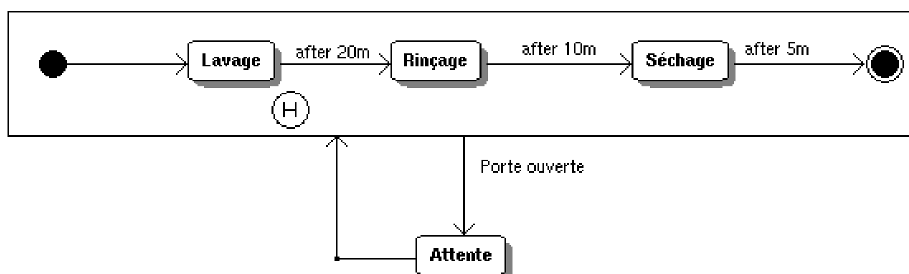
Permet d'invoquer le parallélisme :



11.4.2. Sous-état avec historique

Permet de revenir à l'état quitté précédemment. Cela suppose un moyen de sauvegarder l'état courant.

- Représentation UML : un bloc noté avec un H cerclé.

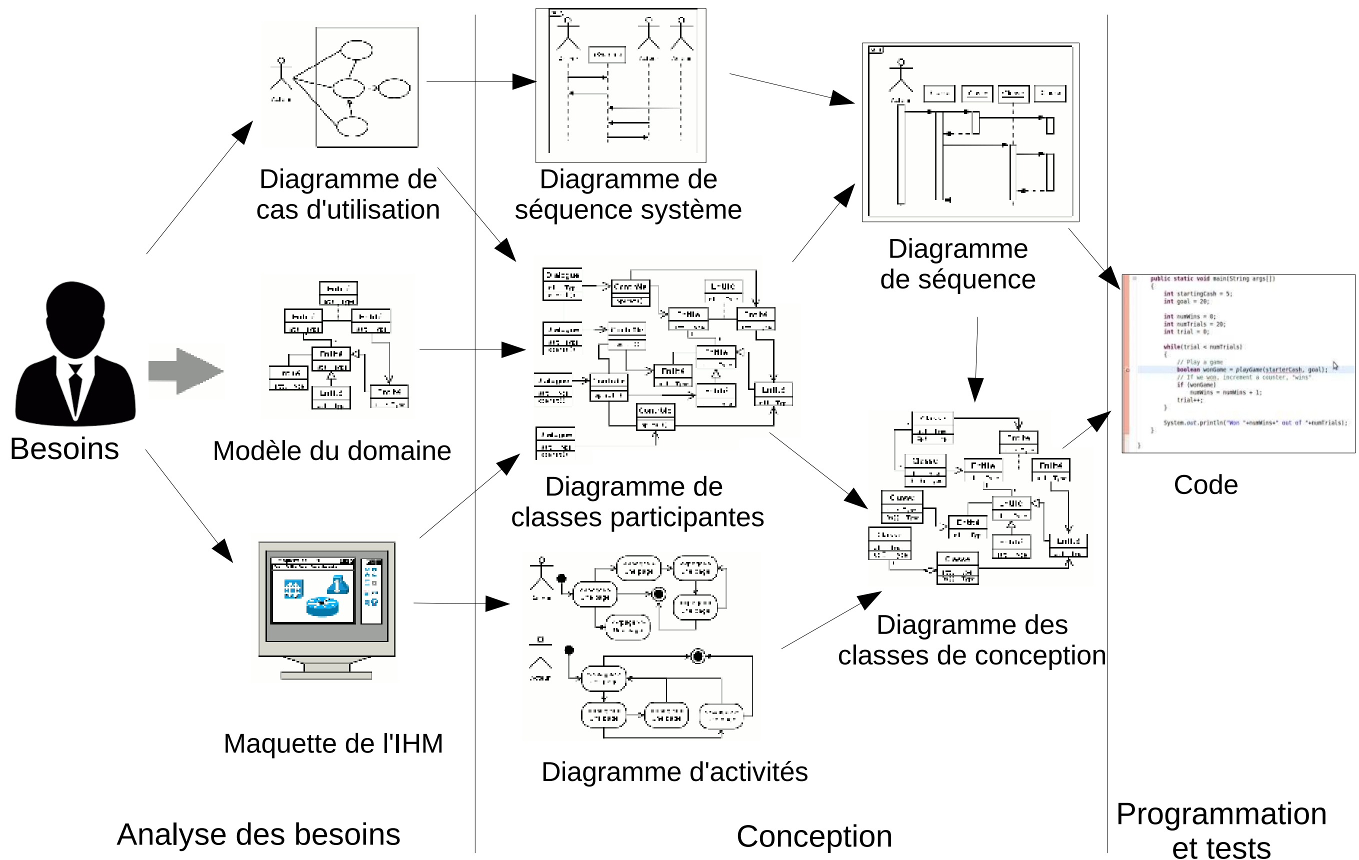


12. Développement et diagrammes UML

12.1. Quand utiliser les diagrammes UML ?

Le schéma suivant localise l'emploi des diagrammes en se référant aux étapes de développement du cycle de

vie en cascade.



12.2. Analyse des besoins

- Diagramme des cas d'utilisation.
  - Poser le problème en termes d'interaction entre les utilisateurs et le système.
  - Identifier les fonctionnalités à réaliser.
- Maquettes des écrans de l'IHM.
  - Une maquette est le dessin des écrans des futures interactions, faits sur papier ou avec un créateur d'interface eg. *SceneBuilder en JavaFX*).
- Modèle du domaine.
  - Une première analyse du domaine à partir des concepts manipulés dans le domaine.

12.3. Conception

- Diagramme de séquence système : scénario d'utilisation du système impliquant les acteurs et le système vu comme une boîte noire. Il permet de donner un ordre entre les cas d'utilisation contrairement au diagramme des cas d'utilisation.
- Diagramme de classes participantes. Parce qu'il est difficile de trouver les classes du futur logiciel il faut :
  - s'appuyer sur le « modèle du domaine ». Il fournit les premières classes.
  - modéliser les aspects fonctionnels avec les diagrammes dynamiques (séquence, activité, état-transition). Ils permettent de comprendre comment cela peut fonctionner.
  - ajouter les « classes participantes » qui ne sont pas du domaine mais nécessaires à l'implémentation informatique (p. ex. base de données). On obtient le diagramme de « classes de conception ».

- Les diagrammes d’activité sont utilisés pour décrire les algorithmes et les processus métiers.
- Les diagrammes de séquence sont utilisés pour détailler les communications entre les objets du domaine pour implémenter une fonctionnalité.

12.4. Conception modulaire

« Diviser pour régner » : face à des problèmes complexes, rechercher la modularité afin d’obtenir un ensemble de modules plus simples et plus facile à gérer.
En conception orientée objet, la notion de module se décline en :

- **Méthode** : implémentation de services.
- **Classe** : abstraction de données et de services.
- **Paquet** : groupement de classes dans une seule collection.

Mais la décomposition d’un système en modules est une tâche difficile en l’absence de théorie. Or ces modules jouent un rôle important pour la flexibilité, la fiabilité et la robustesse du système. Il existe plusieurs méthodologies à partir du cahier des charges. Nous en présentons deux.

12.4.1. Méthode 1 : Analyse de texte (Dennis, 2002)

- Un nom commun → une **classe**
- Un nom propre ou une référence directe→ un **objet**
- Un nom collectif → une **classe**
- Un adjectif → un **attribut**
- Un verbe “faire” → une **méthode**
- Un verbe “être”→ un **héritage** ou une **instanciation**
- Un verbe “avoir” → une **association**
- Un verbe transitif → une **méthode**
- Un verbe intransitif → une **exception**
- Une phrase verbale prédicative → une **méthode**
- Un adverbe → un attribut d’une **méthode**

12.4.2. Méthode 2 : les « cartes CRC » Classes, Responsabilités et Collaborations

Le diagramme de classes est réalisé sur un tableau à partir d’un ensemble de fiches cartonnées de taille A5. On utilise une fiche par classe. La fiche contient :

- Le nom de la classe en haut.
- Les responsabilités de la classe avec en face les collaborations pour assurer cette responsabilité.
  - On assigne à un objet *o* les responsabilités pour lesquelles *o* possède toutes les informations nécessaires pour remplir ces responsabilités (dans certains cas, il collabore avec d’autres objets pour cela).
- Les cartes sont collées sur un tableau et forment ainsi l’architecture du logiciel.

Cette méthode fait la part belle au principe d’encapsulation. Il n’est nullement question d’attributs et une classe est définie par la liste de ses services.