03 Chapitre — Le formalisme UML

1I2AC1 : Génie logiciel et Conception orientée objet

Régis Clouard, ENSICAEN - GREYC

« 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

Plan du chapitre

Objectifs de ce chapitre

- Présentation du langage UML à travers ses principaux diagrammes.
- Démonstration de 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.

Langage UML

Langage de modélisation de logiciels.

- Orienté objet.
- Diagrammatique.
- Normalisé (OMG).
- Indépendant des langages de programmation.

Importance

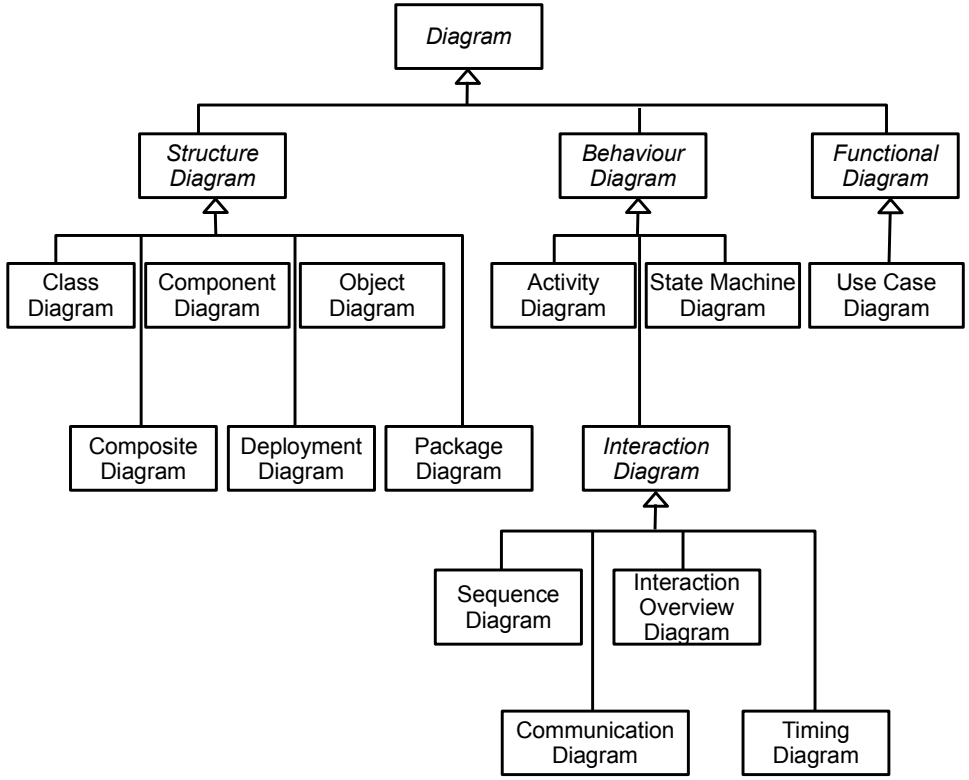
- Il faut utiliser le langage UML pour parler des logiciels.
- Il est partagé par toute la communauté internationale des développeurs.

UML Unified Modeling Language (normalisé).

- Visuel (à base de diagrammes).
- Débarrassé des contraintes syntaxiques.
- Indépendant de tout langage de programmation.
- Correspondance forte avec les langages de programmation.
  - UML ↔ Java, C++, Php, Python...

UML : Vue d'ensemble des diagrammes

13 diagrammes

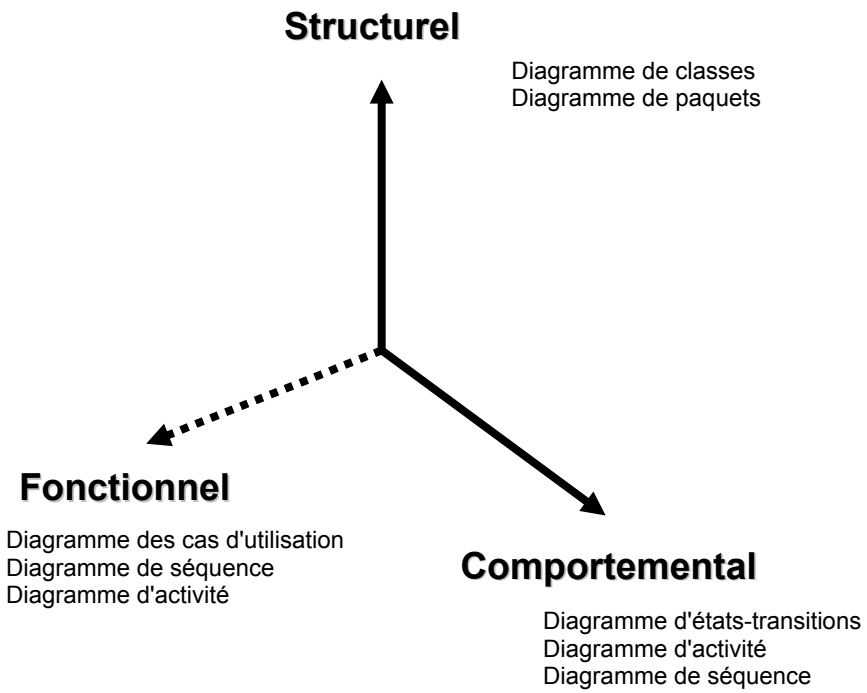


Note : sur les 13 diagrammes, nous ne présentons dans la suite que les 6 diagrammes les plus utilisés :

- diagramme des cas d'utilisation
- diagramme de paquets
- diagramme de classes
- diagramme de séquence
- diagramme d'activités
- diagramme d'états-transitions.

3 vues complémentaires sur la modélisation

- La modélisation d'un logiciel s'aborde par 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.



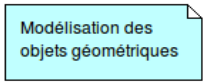
UML : les éléments communs

Les notes : les commentaires de modélisation

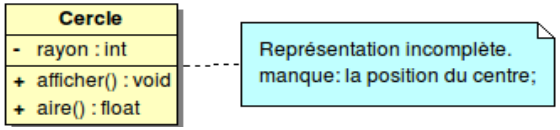
- Symboles graphiques qui permettent de traduire tout ce qui n'entre pas dans le champ d'UML.

Représentation UML

- Une note peut être :
  - libre : porte sur le diagramme entier :



  - associée à un élément quelconque ; elle est liée par un trait pointillé :



Les types primitifs

UML reconnaît tous les types primitifs des langages et ajoute certains types qui peuvent être représentés par des classes à part entière dans certains langages :
- entier (*int*)
- réel (*float*)
- booléen (*bool*)
- temps (*time*)
- date (date)
- caractère (*char*)
- chaîne de caractères (*string*)
- vide (*void*)

1- Diagramme des cas d'utilisation

Intention

- **Donner une vue externe sur le comportement du système.**
- **Identifier les interactions entre utilisateurs et système de manière à déterminer :**
  - **Quel logiciel développer ?**
  - **Que doit faire le système ?**
  - **Quels sont les utilisateurs ?**
  - **Quelles formes d'interaction ?**
- **Préciser les contours du système : ce qui est à l'intérieur et devra être développé et ce qui est à l'extérieur et devra être utilisé.**

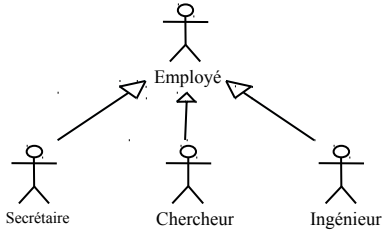
Contexte d'utilisation

À utiliser dès la phase d'analyse des besoins de tout projet (fait partie intégrante du cahier des charges). Il fournit une vision **centrée sur l'utilisateur** du logiciel, c'est-à-dire tournée vers les utilisateurs qui sont la raison d'être du futur logiciel. Il s'agit de décrire les buts et en aucun cas les solutions (ne pas confondre avec le diagramme d'activités).

Éléments UML

1. Acteur

- Représente une entité **extérieure** au système en cours de modélisation.
- S'identifie par le rôle qu'il joue.
- Ce n'est pas forcément une personne humaine (un autre système, un périphérique, …).
- Une même personne physique peut jouer le rôle de plusieurs acteurs.
- Notation UML d'un acteur :..... 

- Hiérarchie d'acteurs ..... 

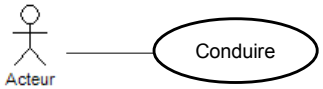
Exemple d'acteurs :
- utilisateurs finaux (rôles),
- systèmes externes (coopérations),
- événements relatifs aux temps (interruptions),
- objets externes passifs (entités).

2. Cas d'utilisation

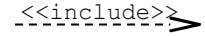
- Représente un ensemble d'actions utilisées pour satisfaire les besoins d'un acteur.
- Spécifié par une phrase intentionnelle (ie., avec verbe).
- Pourra induire à un élément de l'interface graphique (IHM).
- Notation UML d'un cas :..... 

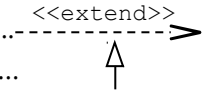
3. Relation entre acteur et cas

- Exprime l'interaction existante entre un acteur et un cas d'utilisation.



4. Relation entre cas

- **Inclusion** : comportement commun : **obligatoire**..... 

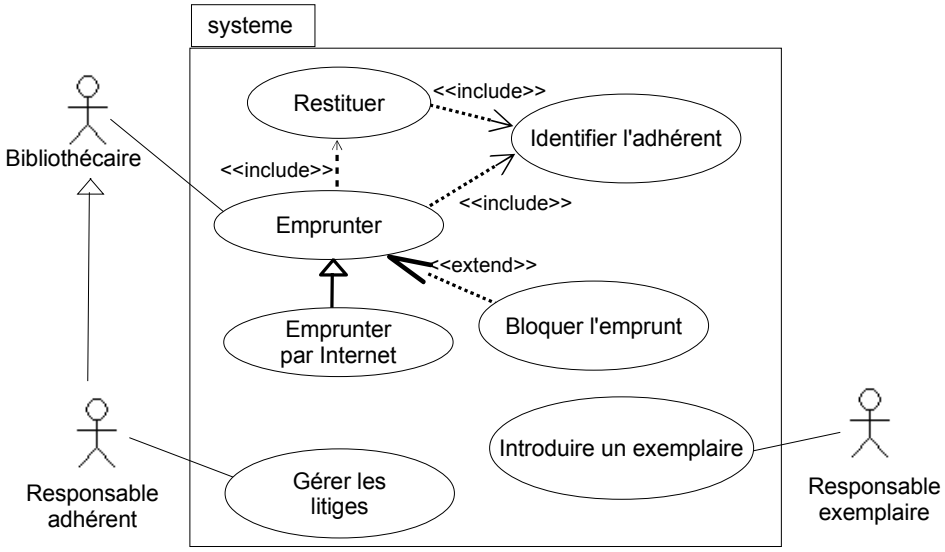
- **Extension** : complément optionnel d'un cas : **facultatif**..... 

- **Spécialisation** : relation de spécialisation d'un cas par un autre cas.....

5. Limite du système

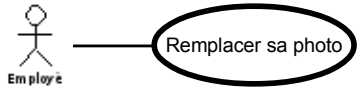
- Seul ce qui est à l'intérieur doit être réalisé.
- Ce qui est à l'extérieur doit pouvoir interagir avec le système via une interface.
- Notation UML : un rectangle qui englobe tous les cas d'utilisation.

6. Exemple : gestion d'une bibliothèque



7. Documentation associée

Le diagramme UML n'explicite pas toutes les informations nécessaires à la description d'un cas d'utilisation. Il est 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. Le nom doit être court et explicite. 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 ?

2- Diagramme de paquets

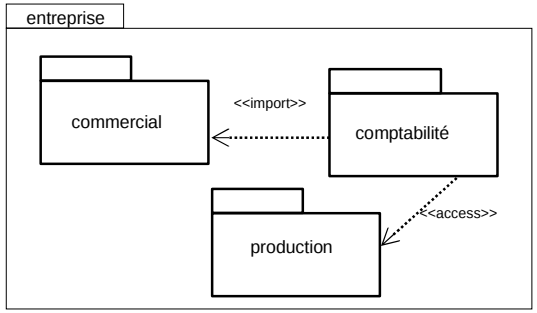
Intention

- **Définir l'organisation globale du logiciel.**
  - **Diviser le projet global en sous-parties.**
  - **Attribuer à des équipes de développement différentes.**
- **Permettre le regroupement des éléments de modélisation.**
  - **Représenté par un dossier étiqueté.**
  - **Nom simple ou nom composé pour les sous-paquets.**
- **Définir un espace de noms.**
  - **Les noms doivent être uniques à l'intérieur d'un paquet.**

Contexte d'utilisation

À utiliser dans tout projet.
Il permet de fournir une **vision logique** de l'organisation du projet.

Représentation UML



Éléments UML

1. Paquet

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

Convention : les noms des paquets sont en minuscule, et peuvent être mis dans l'onglet ou dans le rectangle.

Espace de noms

Les paquets définissent un espace de nom. Les noms des identificateurs sont ainsi locaux aux paquets et n'entrent pas en conflit avec les noms dans d'autres paquets.

Les paquets ont une traduction directe dans le code : **package** en java, **namespace** en C++ et C#.

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, dans la mesure du possible, d'éviter les dépendances circulaires.

a. Une dépendance stéréotypée `<<import>>` : le contenu est ajouté à l'espace de noms (rupture de l'espace de nom).
- La visibilité du paquet est **public**.
- Il n'est pas nécessaire d'utiliser le nom du paquet pour accéder à un élément du paquet importé.
- Les noms doivent être uniques dans chacun des deux paquets.
- C'est une relation transitive en UML, C# et C++ mais pas en Java (si A importe B et B importe C, alors A importe indirectement C).

- Exemple d'utilisation en Java
  - Accès à une méthode statique : import static X; X.methode();
  - Définition d'une instance : import X; X a = new X();

b. Une dépendance stéréotypée `<<access>>` : simple accès (pas de rupture de l'espace de nom).
- La visibilité est **privée**.
- Permet l'utilisation des éléments du paquet sans les ajouter dans l'espace de nom.
- Il est nécessaire d'utiliser le nom du paquet pour accéder à un élément du paquet importé en utilisant la notation '::' en UML ('.' en Java).
- La relation n'est pas transitive.

- Exemple d'utilisation en Java
  - Accès à une méthode : pd.Y.methode();
  - Définition d'une instance : pd.Y a = new pd.Y();

- **Attention** : seuls les éléments **publics** peuvent être importés.

3- Diagramme de classes

Intention

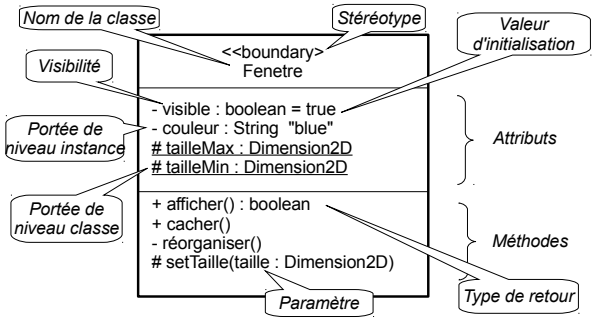
- Exprimer la structure statique du système en termes de classes et de relations entre ces classes.

Contexte d'utilisation

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

Éléments UML

1. Classes

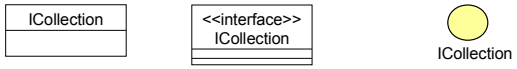


2. Classes abstraites

Notations UML : 

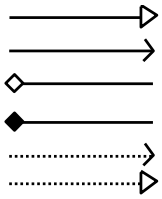
3. Interfaces

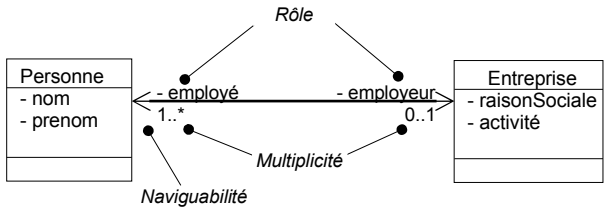
Convention : Les noms des interfaces qui jouent le rôle de type sont suffixés par 'able' (Printable, Comparable) ou préfixés par le caractère I (ICollection).

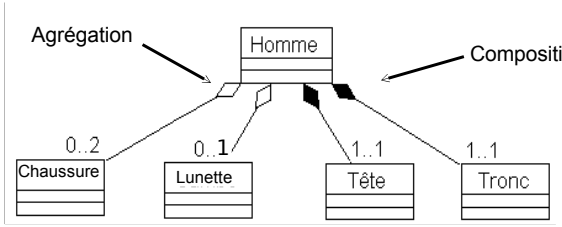
Notations UML : 

4. Relations

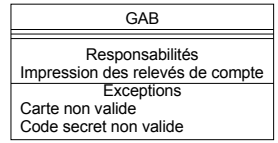
- Héritage
- Association
- Agrégation
- Composition
- Dépendance
- Réalisation







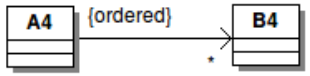
Responsabilités / Exceptions



- Contrat qu'une classe doit respecter.
- Exceptions non prises en compte et renvoyées.

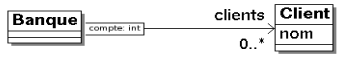
Notation UML : 4e et 5e compartiments dans la classe.

Association ordonnée



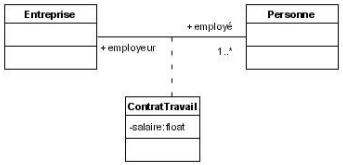
L'association correspond à une liste **ordonnée** d'éléments.

Association qualifiée



Le qualifieur (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.

Classe-association



Utilisée quand un certain nombre d'informations sont attachées au lien entre les instances. Cette représentation est équivalente à un sous-graphe où `Entreprise` est associée à n `ContratTravail`, et `Personne` est aussi associée à n `ContratTravail`.

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 : 

4- Diagramme de séquence

Intention

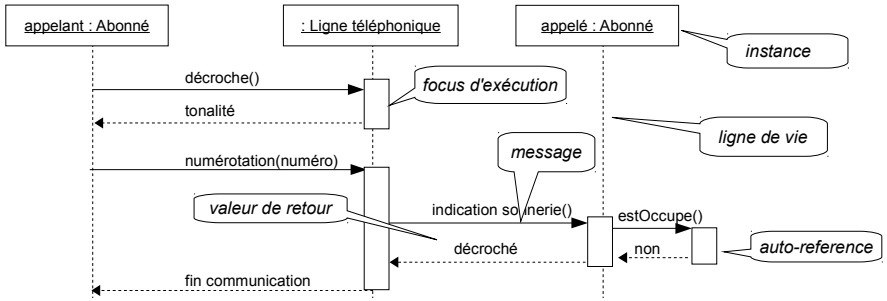
- Décrire une séquence d'interaction entre plusieurs objets, dans un contexte d'exécution donné du système.

- Privilégier le point de vue temporel : lecture du haut vers le bas.

Condition 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 à un cas d'utilisation ou une activité.

Représentation UML



Éléments UML

1. Les instances

Chaque instance est positionnée de la gauche vers la droite.

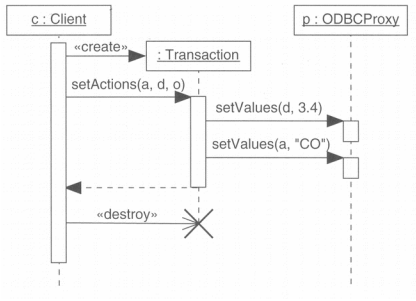
2. La ligne de vie

Représentée par une ligne verticale en pointillée qui prolonge la représentation d'une instance.

3. Les messages

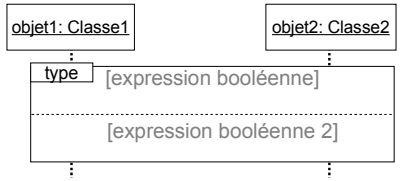
- Catégories de messages :
  - opération de la classe (+ paramètres).
  - envoi de message.
  - destruction/construction d'instance.
- Types de messages :
  - Message de type indéterminé (synchronisé ou non).....
  - Message asynchrone pour les systèmes concurrents.....
  - Message synchrone.....
  - 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 : Interruption de la ligne de vie par une croix ou utiliser le stéréotype `<<destroy>>` sur le message.



Fragments combinés

Un bloc auquel on affecte une structure de contrôle.

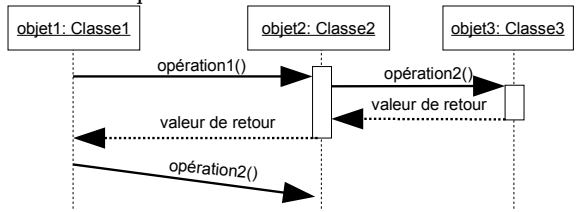


Type :
opt : fragment parcouru si une condition est vérifiée.
loop : répétition du fragment tant que la condition est vérifiée.
alt : une alternative de type si .. alors .. sinon.
critique: fragment exécuté de façon atomique (ininterruptible).
ref : lien vers un autre diagramme pour le détail.

Temporalité

Focus d'exécution :
- Une période d'activation correspond au temps pendant lequel l'objet effectue l'action.
- Représenté par un rectangle plein sur la ligne de vie.
- Peut-être source de nouveaux messages à l'intérieur.

Délai de transmission non négligeable :
- Représenté par une flèche oblique.



5- Diagramme d'activités

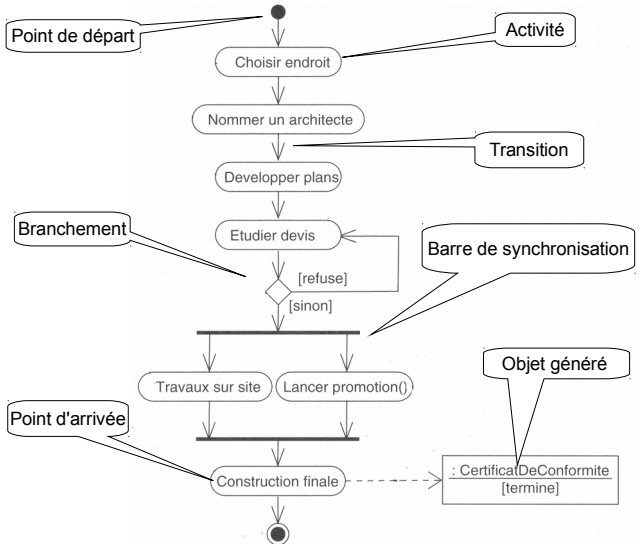
Intention

- Modéliser le comportement par le "flot de contrôle" d'un algorithme.

Condition d'utilisation

- Détailler un **cas d'utilisation** (détailler une fonctionnalité).
- Décrire un **algorithme** (détailler une méthode).
- Modéliser un « **workflow** » (détailler un processus métier).

Représentation UML



Éléments UML

1. Activité

- Étape particulière dans l'exécution
- Opération interruptible qui **dure** un certain temps dans un état particulier.

Notation UML :

- Représentée par un cartouche..... 

- Un point d'entrée..... 
- Un ou plusieurs points de sortie..... 

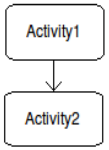
- Une activité peut contenir :
  - l'appel d'une opération.
  - l'envoi d'un signal.
  - la création/la destruction d'un objet.
  - un simple calcul.

2. Transition

Passage automatique d'une activité à une autre.

Notation UML :

- représentée par une ligne fléchée sans nom.
-

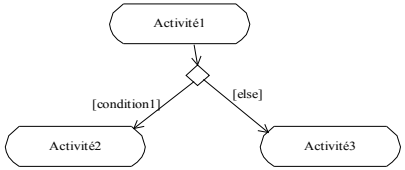


3. Branchement conditionnel

Garde : Condition exprimée une expression booléennes sur la transition d'un branchement.

Syntaxe : expression booléenne.

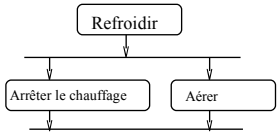
Exemples : [t = 15s] ; [code incorrect & essai > 3]



Synchronisation

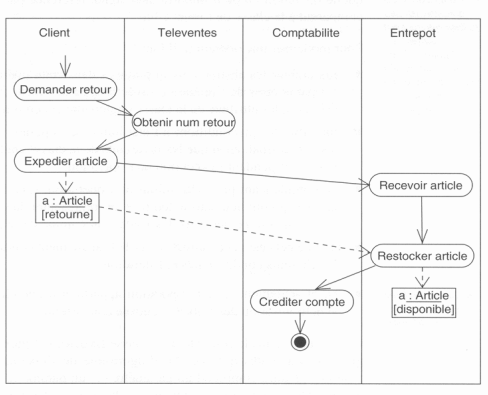
La barre de synchronisation permet d'ouvrir et de fermer des branches parallèles au sein d'un flot d'activités.

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



Les travées (ou couloirs)

- Le modèle est agencé selon des travées verticales correspondant aux classes.
- Exemple : Retour et remboursement d'une marchandise.



6- Diagramme d'états-transitions

Intention

- Décrire les différents états que peuvent prendre une classe en fonction des opérations qui lui sont appliquées.
- Décrire les changements d'états successifs qui sont appliqués à la classe.

Condition d'utilisation

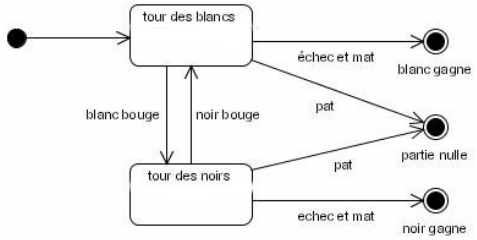
- Un diagramme état-transition est propre à une **classe donnée**.
- Détailler le comportement complexe d'une classe.
- Décrire le comportement d'un objet à travers plusieurs cas d'utilisation.

Éléments UML

1. État

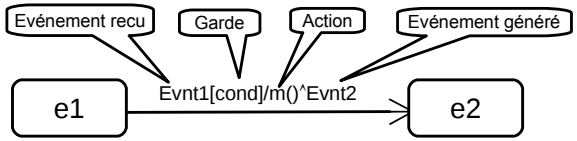
Définit par des valeurs d'attributs de la classe. Deux états différents doivent diverger d'au moins une valeur d'attribut.
Décrit une condition ou une situation qui survient au cours de la vie d'un objet :
- pendant laquelle l'objet satisfait à certaines conditions,
- pendant laquelle l'objet exécute 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.

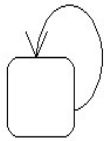


2. Transition

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) : méthode effectuée lors de la transition.
- l'état cible.

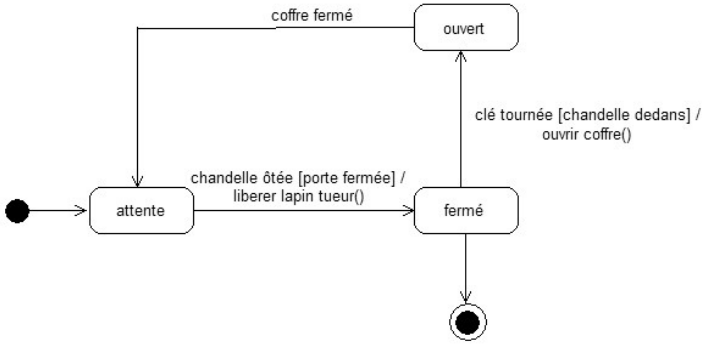


Auto-transition : .....



3. Événement

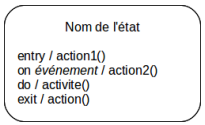
Stimulus qui peut déclencher une transition d'état.
Il se produit à un instant donné.
Il n'a pas de durée.
4 types d'événement :
- un événement de type ***signal*** représente un signal asynchrone envoyé par un composant extérieur.
- un événement de type ***appel*** représente le déclenchement d'une méthode synchrone d'une classe.
- un événement ***temporel*** représente l'écoulement du temps (`after`).
- un événement de ***modification*** représente un changement d'état (`when`).



4. Modélisation d'un état

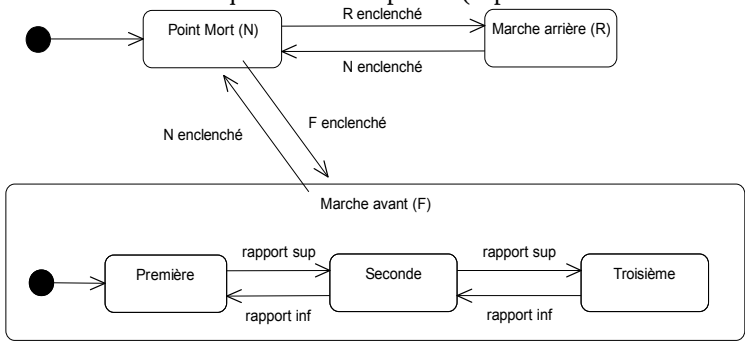
- Action d'entrée : `entry`
  - une **action** correspond typiquement à une opération de la classe mais qui ne peut pas être interrompue durant son exécution.
- Action d'entrée par un événement donné : `on evenement`
- Action de sortie : `exit`
- Activité : `do`
  - Une **activité** est une opération de classe qui peut être interrompue durant son exécution.

- Le séquencement est possible `do / operation1(); operation2(); operation3().`



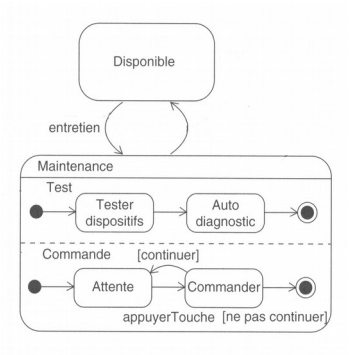
Sous-état

Permet une modélisation hiérarchique d'états complexes (super-état raffiné en sous-états).



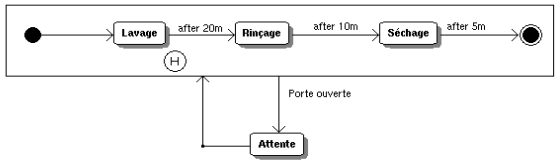
1. Sous-états concurrents

Permet d'invoquer le parallélisme :



2. Avec historique

Permet de revenir à l'état quitté précédemment.



Extensions de la notation UML

Extension par les prototypes

- Son utilisation permet d'étendre la définition des éléments d'UML.
- On les représente :

- par un mot entre double chevrons..... 

- par un icône ..... 

1. Exemples applicables aux classes

- `<<interface>>` classe sans implémentation.
- `<<abstract>>` classe ou opération abstraite.
- `<<exception>>` fonctionnement identique aux exceptions des langages de programmation.
- `<<signal>>` identique aux signaux des OS (ou signaux/slots de Qt).
- `<<enumeration>>` type énuméré.
- `<<utility>>` classe dont les opérations sont toutes statiques (p.ex. La classe Math en Java).
- `<<process>>` … processus lourd.
- `<<thread>>` … processus léger.
- `<<boundary>>` classe d'interface avec l'utilisateur (IHM graphique ou texte)..... 
- `<<control>>` classe qui joue le rôle de contrôleur (p. ex. croupier)..... 
- `<<entity>>` une classe qui détient des données (p. ex. base de données)..... 
- `<<actor>>` modélisation du comportement d'un agent extérieur au système..... 

2. Exemples applicables aux dépendances

- `<<access>>, <<import>>` entre packages.
- `<<subsystem>>` sous-système du système à modéliser.
- `<<send>>` envoi de signal.
- `<<instance-of>>` et `<<instantiate>>` relations instance-classe.
- `<<friend>>` donne accès à certains éléments privés, cf C++.

3. Exemples applicables aux messages ou événements

- `<<create>>`, `<<destroy>>` création/destruction de l'instance cible.
- … et tous ceux que l'on souhaite créer pour une application particulière.

Extension par les étiquettes (tag ou tag-value)

- s'exprime entre accolades.
  - soit comme un simple littéral : `{ constante }`
  - soit comme un couple : `{ étiquette = valeur }`.
- s'applique à la plupart des éléments.

Extension par les contraintes - OCL (Object Constraint Language)

Elle s'exprime entre accolades.

1. Contrainte temporelle

```
{ toutes les minutes }
{ temps < 10 ms }
```

2. Contrainte sur les instances

`{ transient }` : non sérialisable.

3. Contrainte sur les associations

`{ or }` : soit l'une soit l'autre des associations mais pas les deux.
`{ ordered by surname }` : les classes sont ordonnées par leur nom.

4. Contrainte sur les associations (généralisation)

`{ uncomplete }` : les classes dérivées ne couvrent pas toute la classe de base.
`{ complete }` : les classes dérivées couvrent toute la classe de base.
`{ disjoint }` : les classes dérivées sont disjointes donc pas d'héritage multiple.
`{ overlapping }` : l'héritage multiple est possible.

Quand utiliser les diagrammes ?