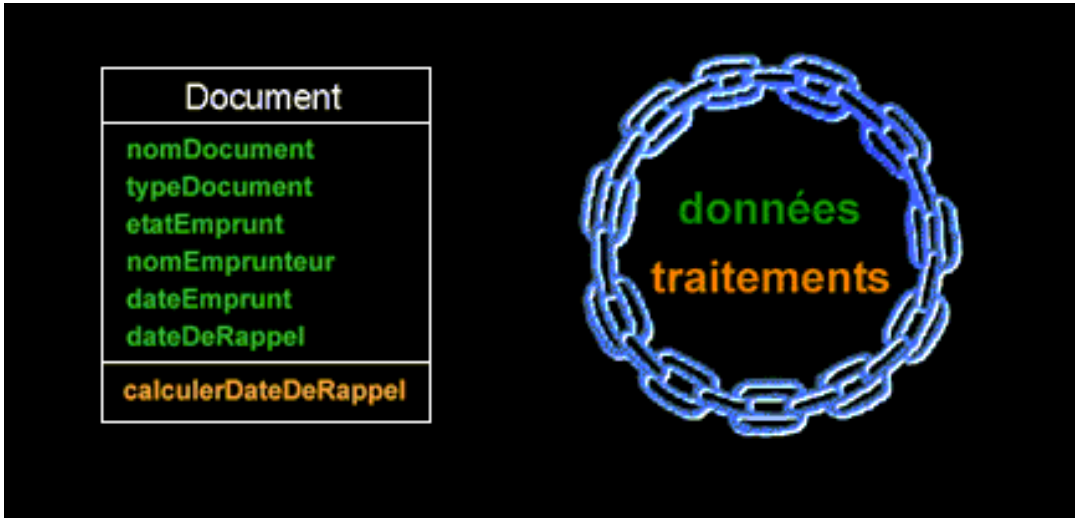
Présentation d’UML

HO Tuong Vinh

Concepts importants de l'approche objet

Objet



Centraliser les données d'un type et les traitements associés, dans une même unité physique, permet de limiter les points de maintenance dans le code et facilite l'accès à l'information en cas d'évolution du logiciel

Concepts importants de l'approche objet

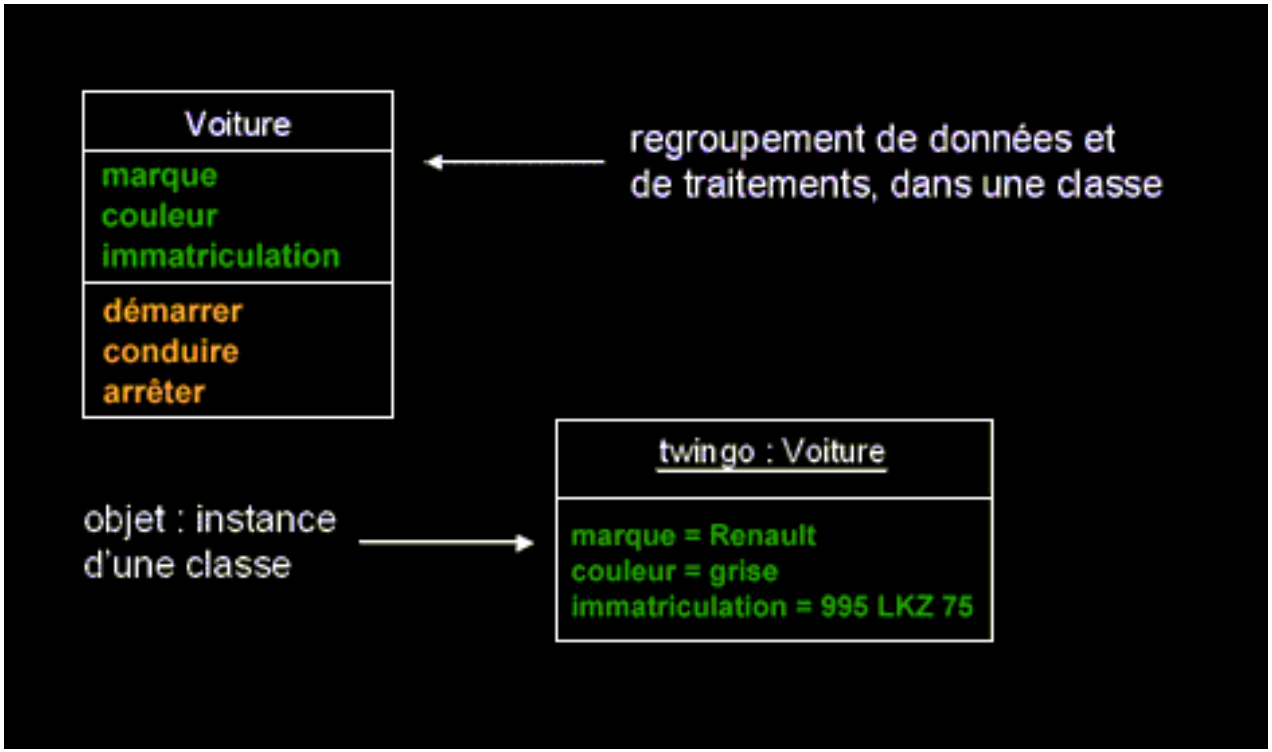
Objet

- Un objet est une entité aux frontières précises qui possède une identité (un nom).
- Un ensemble d'attributs caractérise l'état de l'objet.
- Un ensemble d'opérations (méthodes) en définissent le comportement.
- Un objet est une instance de classe (une occurrence d'un type abstrait).
- Une classe est un type de données abstrait, caractérisé par des propriétés (attributs et méthodes) communes à des objets et permettant de créer des objets possédant ces propriétés.

Concepts importants de l'approche objet

Objet

Exemple



Concepts importants de l'approche objet

Encapsulation

- Consiste à masquer les détails d'implémentation d'un objet, en définissant une interface.
- L'interface est la vue externe d'un objet, elle définit les services accessibles (offerts) aux utilisateurs de l'objet.
- L'encapsulation facilite l'évolution d'une application car elle stabilise l'utilisation des objets : on peut modifier l'implémentation des attributs d'un objet sans modifier son interface.
- L'encapsulation garantit l'intégrité des données, car elle permet d'interdire l'accès direct aux attributs des objets (utilisation d'accesseurs).

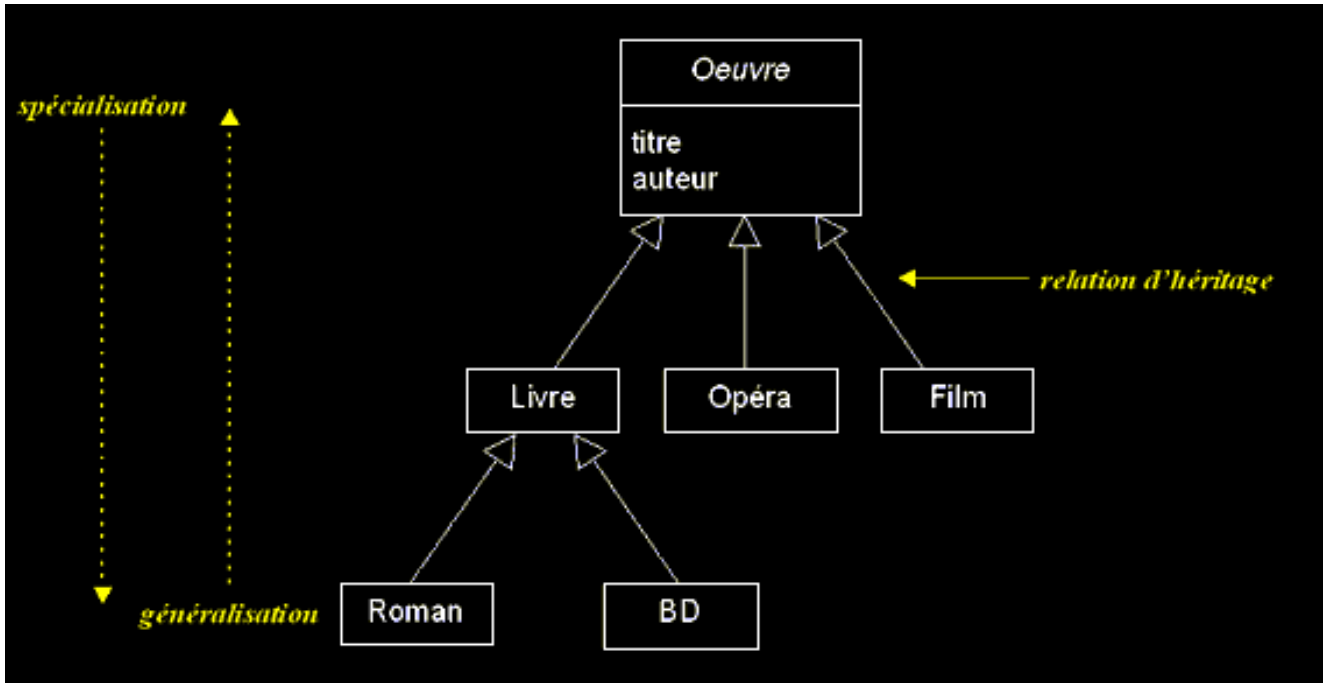
Concepts importants de l'approche objet

Héritage (et polymorphisme)

- L'héritage est un mécanisme de transmission des propriétés d'une classe (ses attributs et méthodes) vers une sous-classe.
- Une classe peut être spécialisée en d'autres classes, afin d'y ajouter des caractéristiques spécifiques ou d'en adapter certaines.
- Plusieurs classes peuvent être généralisées en une classe qui les factorise, afin de regrouper les caractéristiques communes d'un ensemble de classes.
- La spécialisation et la généralisation permettent de construire des hiérarchies de classes.
- L'héritage évite la duplication et encourage la réutilisation.
- Le polymorphisme représente la faculté d'une méthode à pouvoir s'appliquer à des objets de classes différentes.

Concepts importants de l'approche objet

Héritage (et polymorphisme)



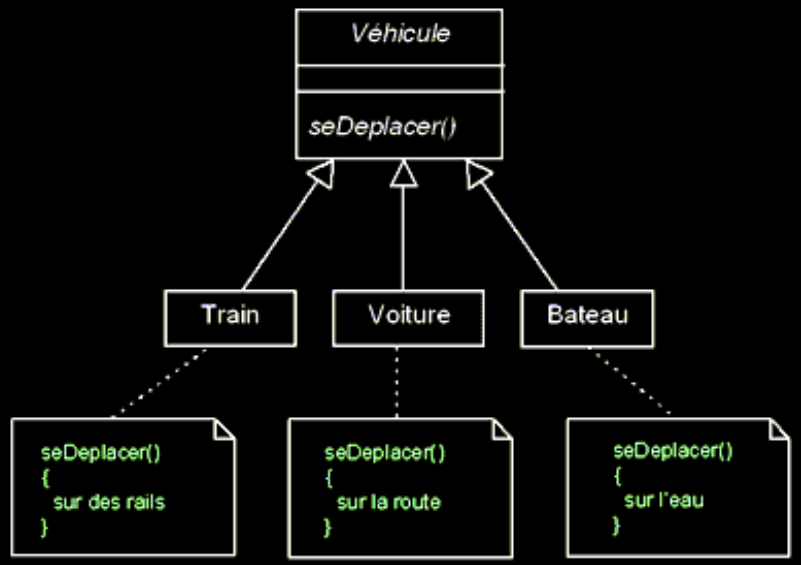
Exemple d'une hiérarchie de classes

Concepts importants de l'approche objet

Héritage (et polymorphisme)

Polymorphisme, exemple :

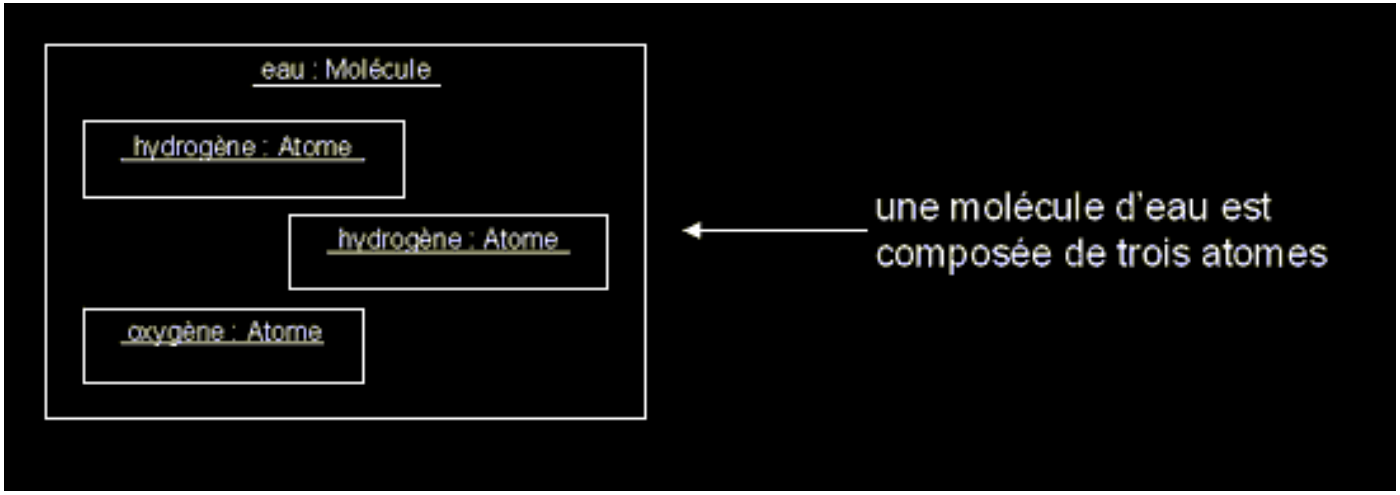
```
Vehicule convoi[3] = {
  Train("TGV"),
  Voiture("twingo"),
  Bateau("Titanic")
};

for (int i = 0; i < 3; i++)
{
  convoi[i].seDeplacer();
}
```



Concepts importants de l'approche objet

Agrégation

- Il s'agit d'une relation entre deux classes, spécifiant que les objets d'une classe sont des composants de l'autre classe.
- Une relation d'agrégation permet donc de définir des objets composés d'autres objets. L'agrégation permet d'assembler des objets de base, afin de construire des objets plus complexes.



Modélisation

Qu'est-ce qu'un modèle ?

Un modèle est une abstraction de la réalité

L'abstraction est un des piliers de l'approche objet.

- Il s'agit d'un processus qui consiste à identifier les caractéristiques intéressantes d'une entité, en vue d'une utilisation précise.
- L'abstraction désigne aussi le résultat de ce processus, c'est-à-dire l'ensemble des caractéristiques essentielles d'une entité, retenues par un observateur.

Modélisation

Qu'est-ce qu'un modèle ?

Un modèle est une vue subjective mais pertinente de la réalité

- Un modèle définit une frontière entre la réalité et la perspective de l'observateur. Ce n'est pas "la réalité", mais une vue très subjective de la réalité.
- Bien qu'un modèle ne représente pas une réalité absolue, un modèle reflète des aspects importants de la réalité, il en donne donc une vue juste et pertinente.

Modélisation

Quelques exemples de modèles

- Modèle météorologique :
  à partir de données d'observation (satellite ...), permet de prévoir les conditions climatiques pour les jours à venir.
- Modèle économique :
  peut par exemple permettre de simuler l'évolution de cours boursiers en fonction d'hypothèses macro-économiques (évolution du chômage, taux de croissance...).
- Modèle démographique :
  définit la composition d'un panel d'une population et son comportement, dans le but de fiabiliser des études statistiques, d'augmenter l'impact de démarches commerciales, etc...

Modélisation

Caractéristiques fondamentales des modèles

Le caractère abstrait d'un modèle doit notamment permettre :

- de faciliter la compréhension du système étudié
  > Un modèle réduit la complexité du système étudié.
- de simuler le système étudié
  > Un modèle représente le système étudié et reproduit ses comportements.

Modélisation

Caractéristiques fondamentales des modèles

Le caractère abstrait d'un modèle doit notamment permettre :

- de faciliter la compréhension du système étudié
  > Un modèle réduit la complexité du système étudié.
- de simuler le système étudié
  > Un modèle représente le système étudié et reproduit ses comportements.

Modélisation orientée objet

- Le cœur d'une résolution orientée objet est la construction d'un modèle.
- Le modèle représente une abstraction des détails essentiels du problème à résoudre.
- Le langage de la modélisation orientée objet le plus connu est l'UML (Unified Modeling Language)

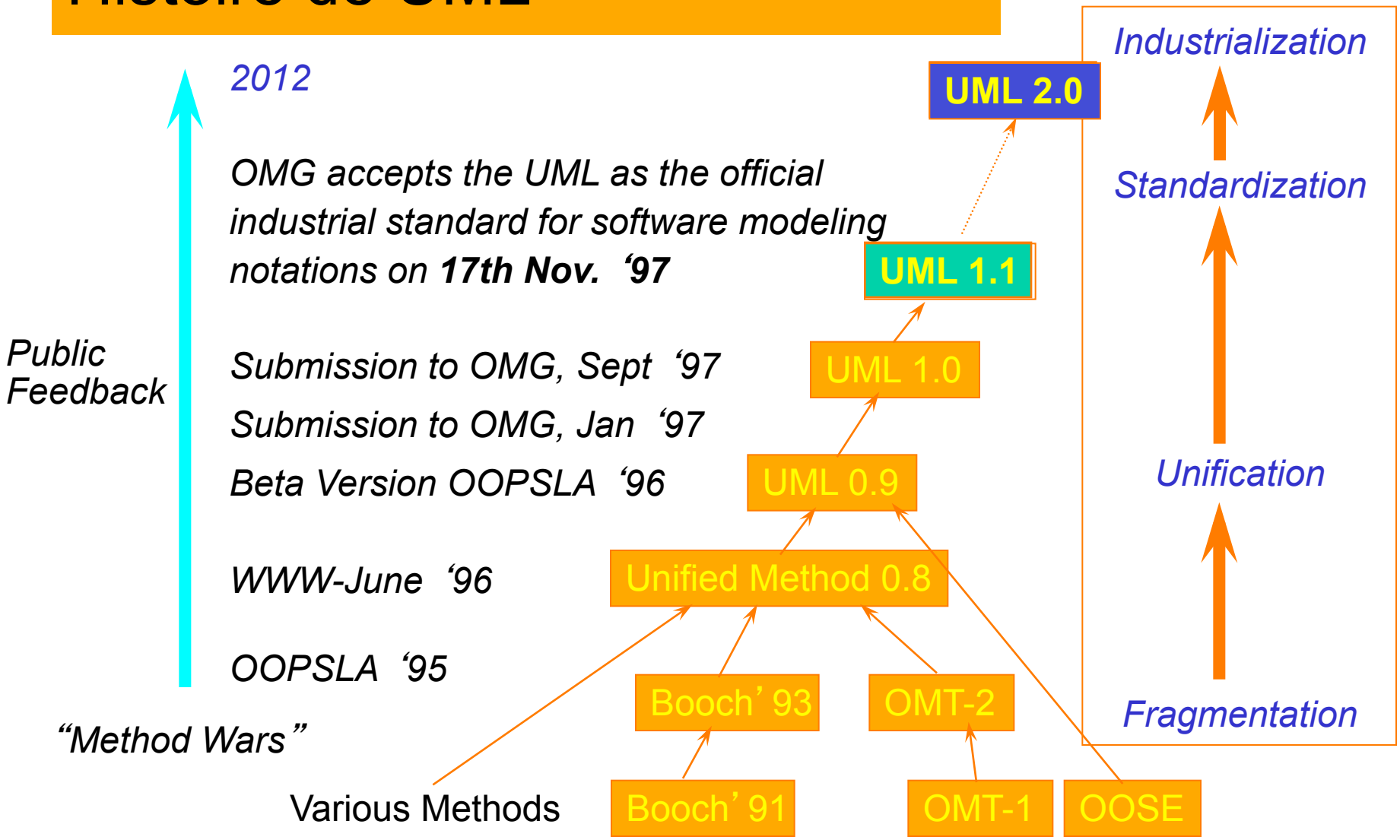
Modélisation orientée objet

- Elle commence avec la construction d'un modèle
- Un modèle est une abstraction du problème à résoudre
- Le domaine est l'environnement (espace) d'où vient le problème
- Les modèles consistent en objets qui interagissent par l'échange des messages
- Penser d'un objet comme "vivant." Les objets ont des choses qu'ils savent (attributs) et des choses qu'ils peuvent faire (comportements ou opérations). Les valeurs des attributs d'un objet déterminent son état
- Les classes sont les "représentation" pour les objets. Une classe enveloppe des attributs (données) et comportements (méthodes ou charges) dans une entité distincte seule. Les objets sont exemples des classes

Modélisation avec UML

- **U**nified **M**odeling **L**anguage
- Une notation de conception standarde pour:
  - ✓Exprimer l'architecture de logiciel
  - ✓Exprimer les comportements de logiciel
  - ✓Documenter le déploiement de logiciel
- OMG standard
- La norme industrielle de-facto aujourd'hui
- UML supporte à la fois le Processus et l' Architecture

Histoire de UML



Modélisation avec UML

Il y a toujours plusieurs manières de voir les choses: la représentation d'un objet est une affaire de point de vue.

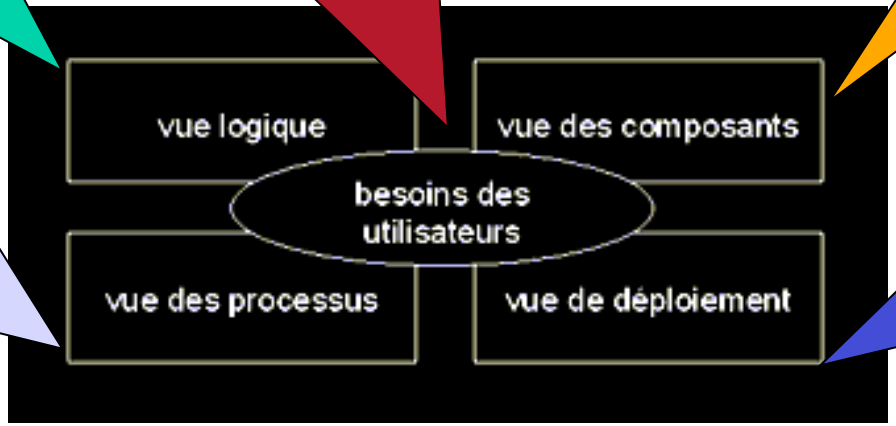
Modélisation avec UML

La vue « 4 + 1 » de UML

Identifier les éléments du domaine, ainsi que les relations et interactions entre ces éléments

Définir les besoins des clients du système

Identifier les modules qui réalisent (physiquement) les classes de la vue logique



Identifier les processus du système, ainsi que les relations et interactions entre ces processus

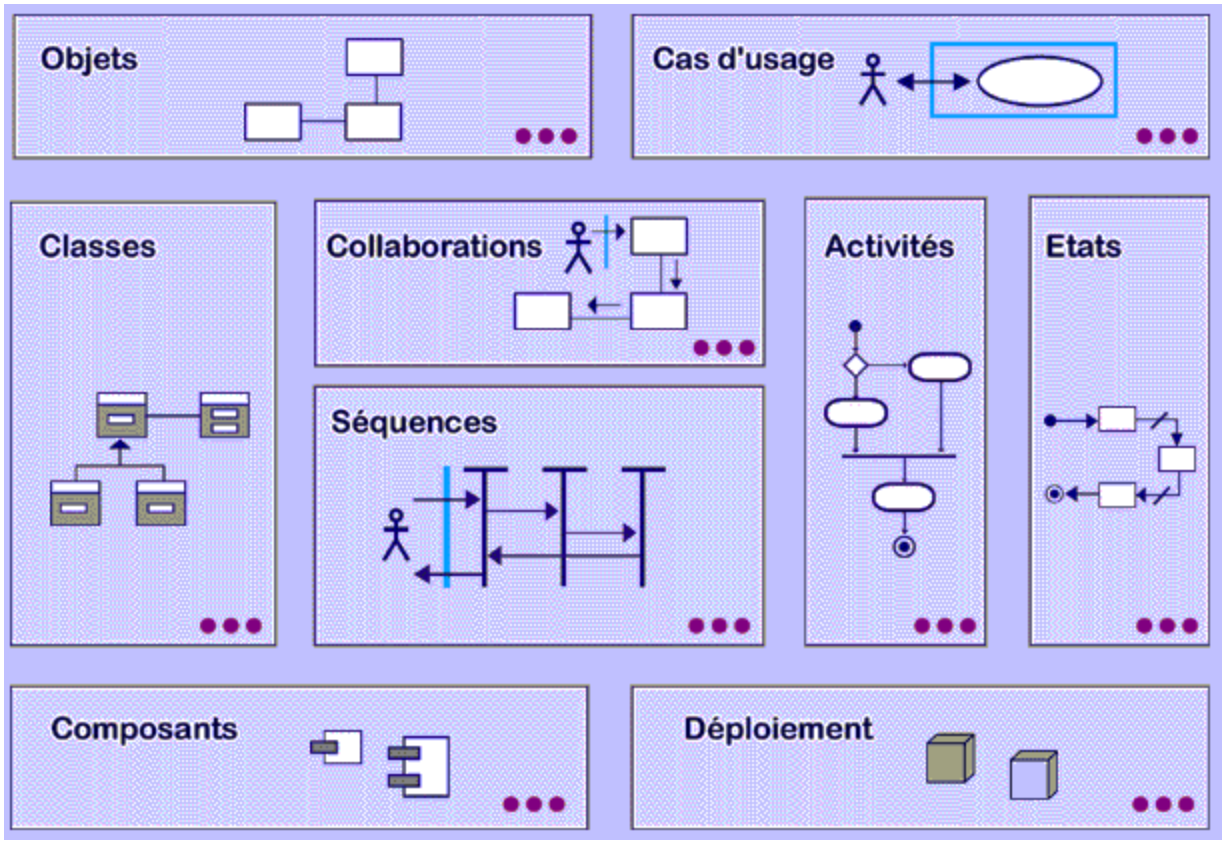
Décrire les ressources matérielles et la répartition du logiciel dans ces ressources

Modélisation avec UML

- UML permet de définir et de visualiser un modèle, à l'aide de diagrammes.
- Un diagramme UML est une représentation graphique, qui s'intéresse à un aspect précis du modèle ; c'est une perspective du modèle, pas "le modèle".
- Chaque type de diagramme UML possède une structure (les types des éléments de modélisation qui le composent sont prédéfinis).
- Un type de diagramme UML véhicule une sémantique précise (un type de diagramme offre toujours la même vue d'un système). Combinés, les différents types de diagrammes UML offrent une vue complète des aspects statiques et dynamiques d'un système.

Les diagrammes de UML

UML fournit neuf diagrammes de base pour modéliser les différents aspects d'un système d'information:



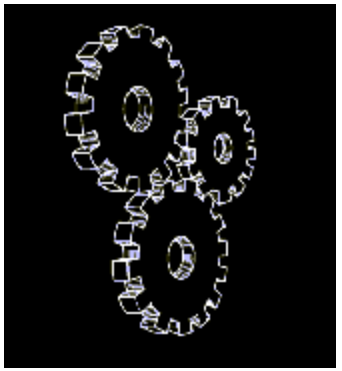
Les diagrammes de UML

Vues statiques du système

- diagrammes de cas d'utilisation
- diagrammes d'objets
- diagrammes de classes
- diagrammes de composants
- diagrammes de déploiement

Vues dynamiques du système

- diagrammes de collaboration
- diagrammes de séquence
- diagrammes d'états-transitions
- diagrammes d'activités

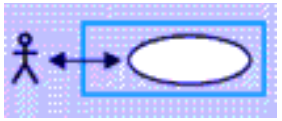


Les diagrammes de UML

Vues statiques du système

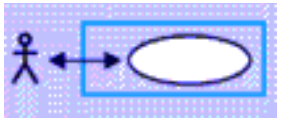
- diagrammes de cas d'utilisation
- diagrammes d'objets
- diagrammes de classes
- diagrammes de composants
- diagrammes de déploiement

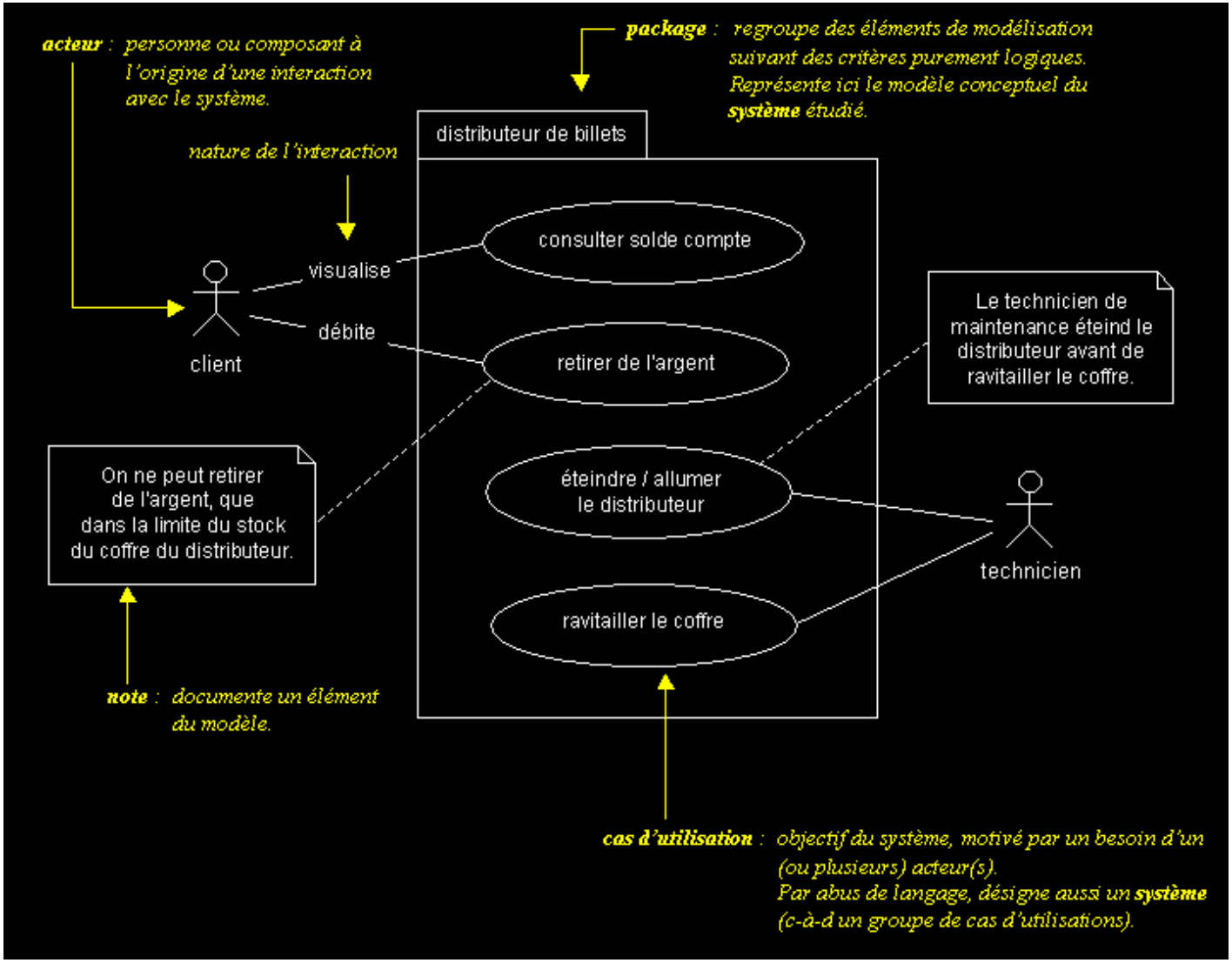
Les cas d’utilisation



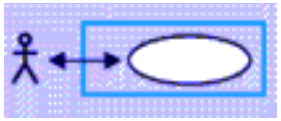
- Les use cases permettent de structurer les besoins des utilisateurs et les objectifs correspondants d'un système.
- Ils centrent l'expression des exigences du système sur ses utilisateurs : ils partent du principe que les objectifs du système sont tous motivés.
- Ils se limitent aux préoccupations "réelles" des utilisateurs ; ils ne présentent pas de solutions d'implémentation et ne forment pas un inventaire fonctionnel du système.
- Ils identifient les utilisateurs du système (acteurs) et leur interaction avec le système.
- Ils permettent de classer les acteurs et structurer les objectifs du système.
- Ils servent de base à la traçabilité des exigences d'un système dans un processus de développement intégrant UML.

Les cas d'utilisation





Les cas d’utilisation



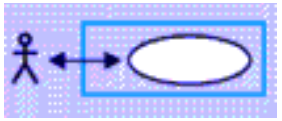
Acteur : entité externe qui agit sur le système (opérateur, autre système...).

- L'acteur peut consulter ou modifier l'état du système.
- En réponse à l'action d'un acteur, le système fournit un service qui correspond à son besoin.
- Les acteurs peuvent être classés.

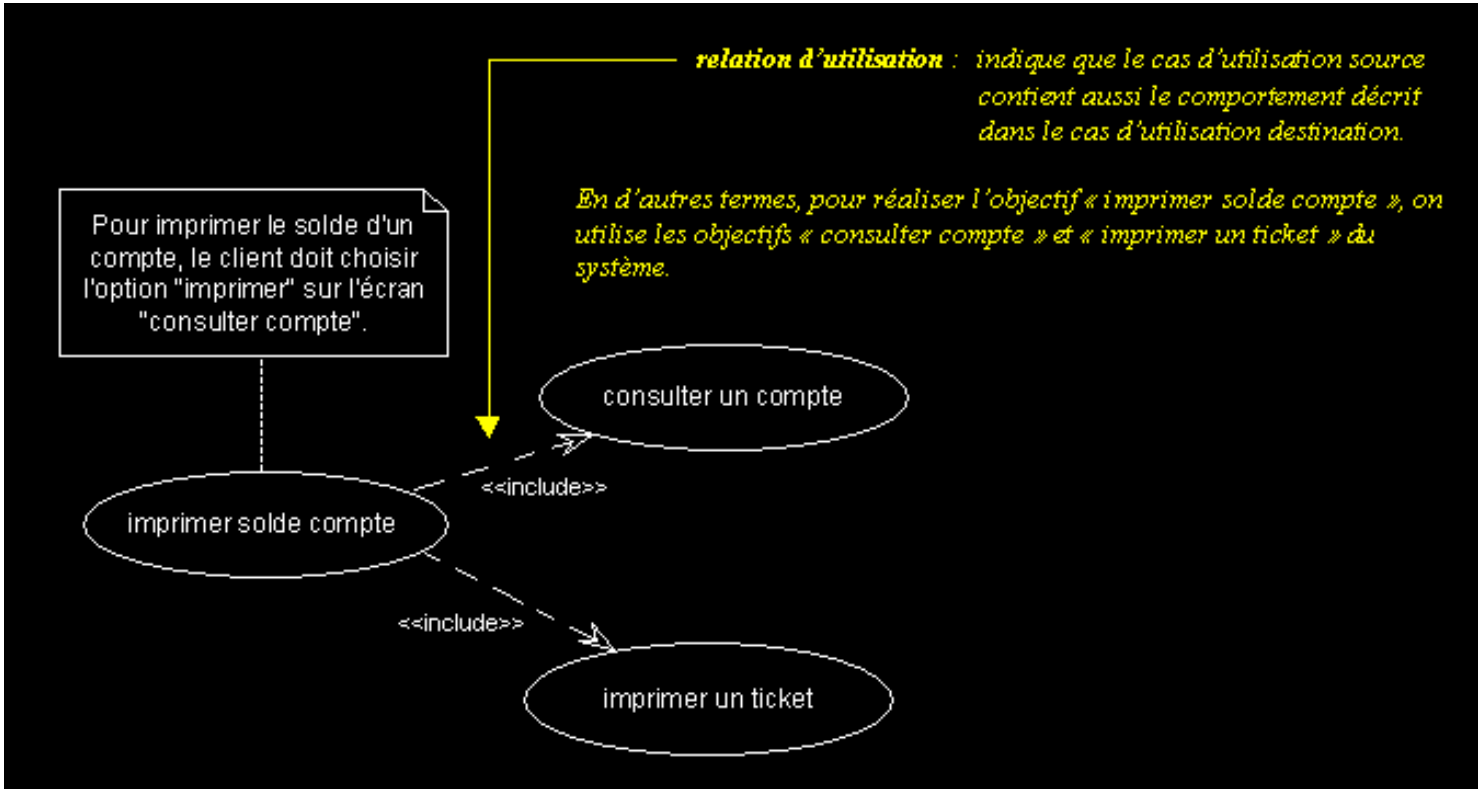
Use case : ensemble d'actions réalisées par le système, en réponse à une action d'un acteur.

- Les uses cases peuvent être structurés.
- Les uses cases peuvent être organisés en paquetages (packages).
- L'ensemble des use cases décrit les objectifs (le but) du système.

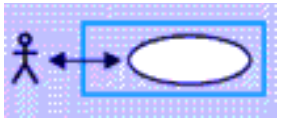
Les cas d'utilisation



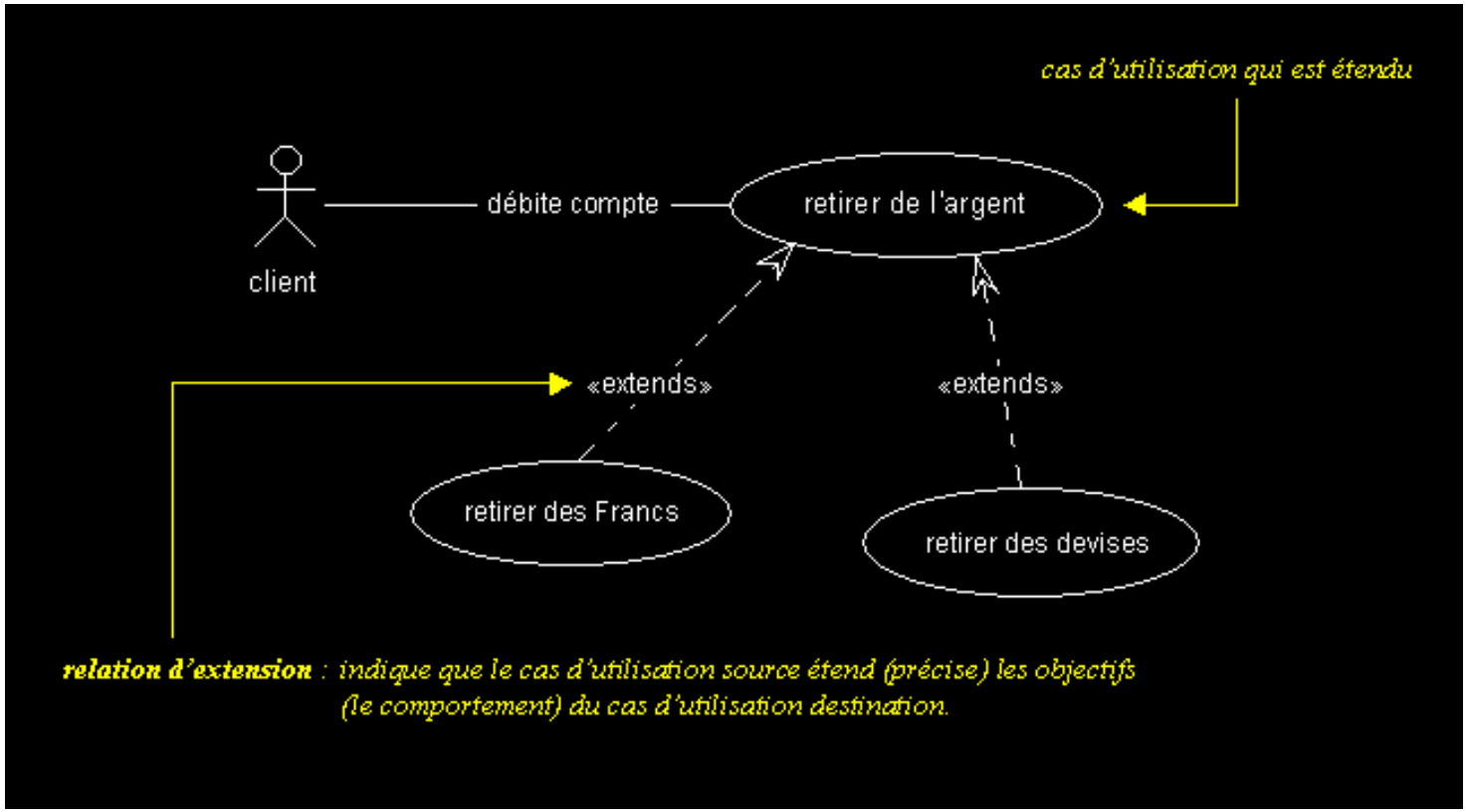
Relation d'utilisation



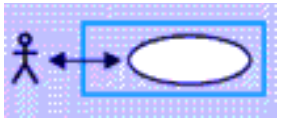
Les cas d'utilisation



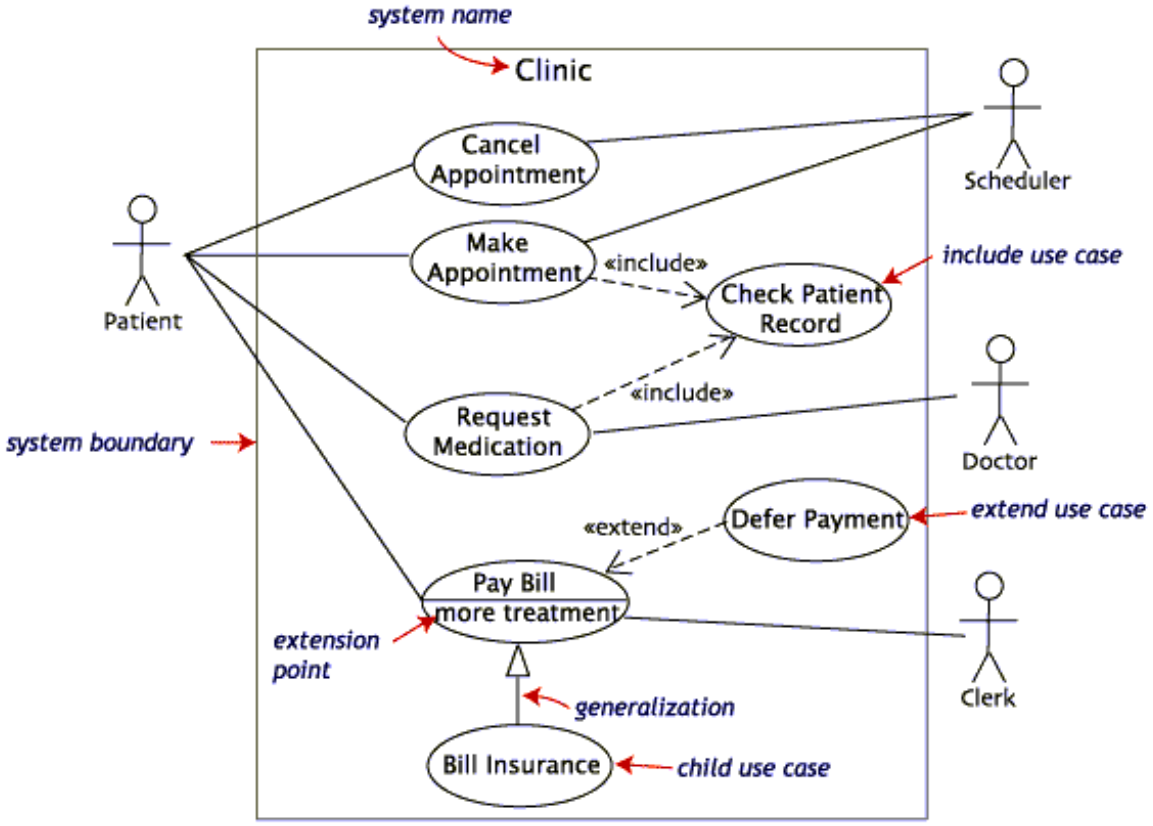
Relation d'extension



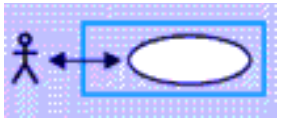
Les cas d'utilisation



Exemple



Les cas d'utilisation



Exemple

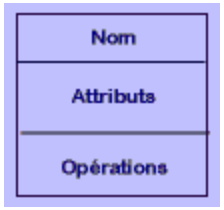
A **system boundary** rectangle separates the clinic system from the external actors.

A use case **generalization** shows that one use case is simply a special kind of another. **Pay Bill** is a parent use case and **Bill Insurance** is the child. A child can be substituted for its parent whenever necessary. Generalization appears as a line with a triangular arrow head toward the parent use case.

Include relationships factor use cases into additional ones. Includes are especially helpful when the same use case can be factored out of two different use cases. Both **Make Appointment** and **Request Medication** include **Check Patient Record** as a subtask. In the diagram, include notation is a dotted line beginning at base use case ending with an arrows pointing to the include use case. The dotted line is labeled <<include>>.

An **extend** relationship indicates that one use case is a variation of another. Extend notation is a dotted line, labeled <<extend>>, and with an arrow toward the base case. The **extension point**, which determines when the extended case is appropriate, is written inside the base case.

Les diagrammes de classe



- Une classe est un type abstrait caractérisé par des propriétés (attributs et méthodes) communes à un ensemble d'objets et permettant de créer des objets ayant ces propriétés.

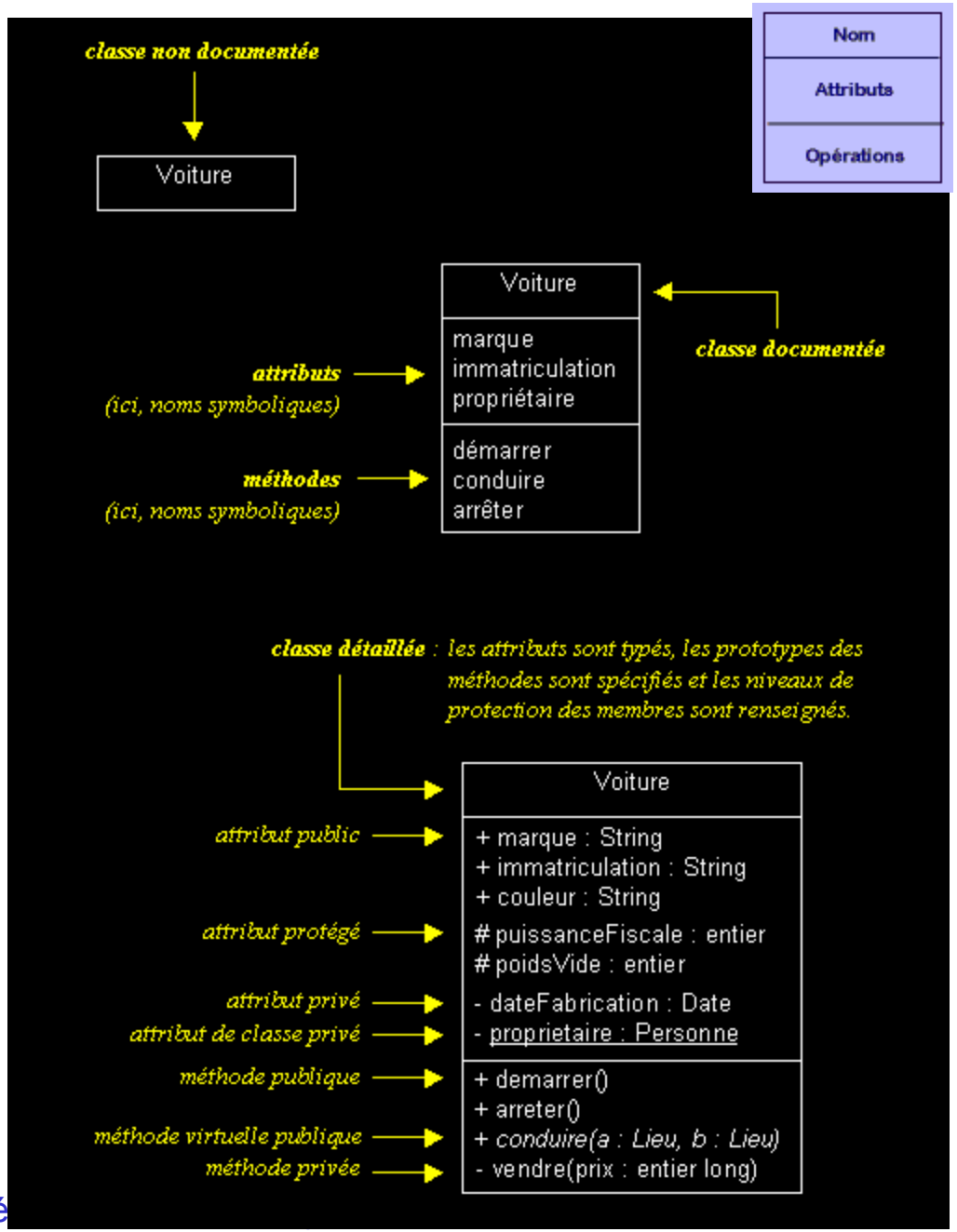
Classe = attributs + méthodes + instanciation

- Ne pas représenter les attributs ou les méthodes d'une classe sur un diagramme, n'indique pas que cette classe n'en contient pas. Il s'agit juste d'un filtre visuel, destiné à donner un certain niveau d'abstraction à son modèle.

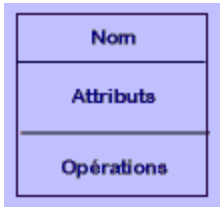
- De même, ne pas spécifier les niveaux de protection des membres d'une classe ne veut pas dire qu'on ne représente que les membres publics. Là aussi, il s'agit d'un filtre visuel.

Les diagrammes de classe

Documentation d'une classe

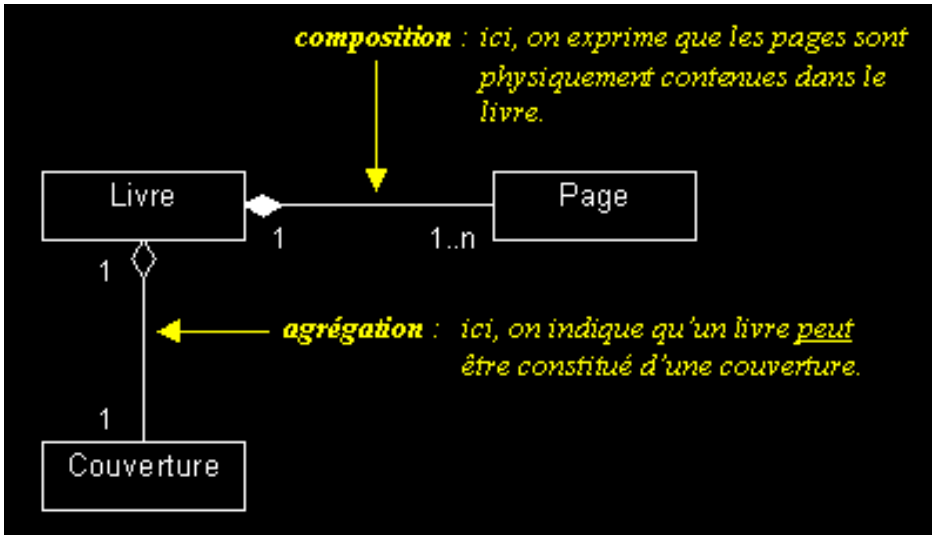


Les diagrammes de classe

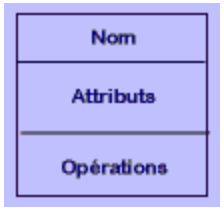


Composition

- La composition est une agrégation forte (agrégation par valeur).
- Les cycles de vies des éléments (les "composants") et de l'agrégat sont liés : si l'agrégat est détruit (ou copié), ses composants le sont aussi.
- A un même moment, une instance de composant ne peut être liée qu'à un seul agrégat. Les "objets composites" sont des instances de classes composées.



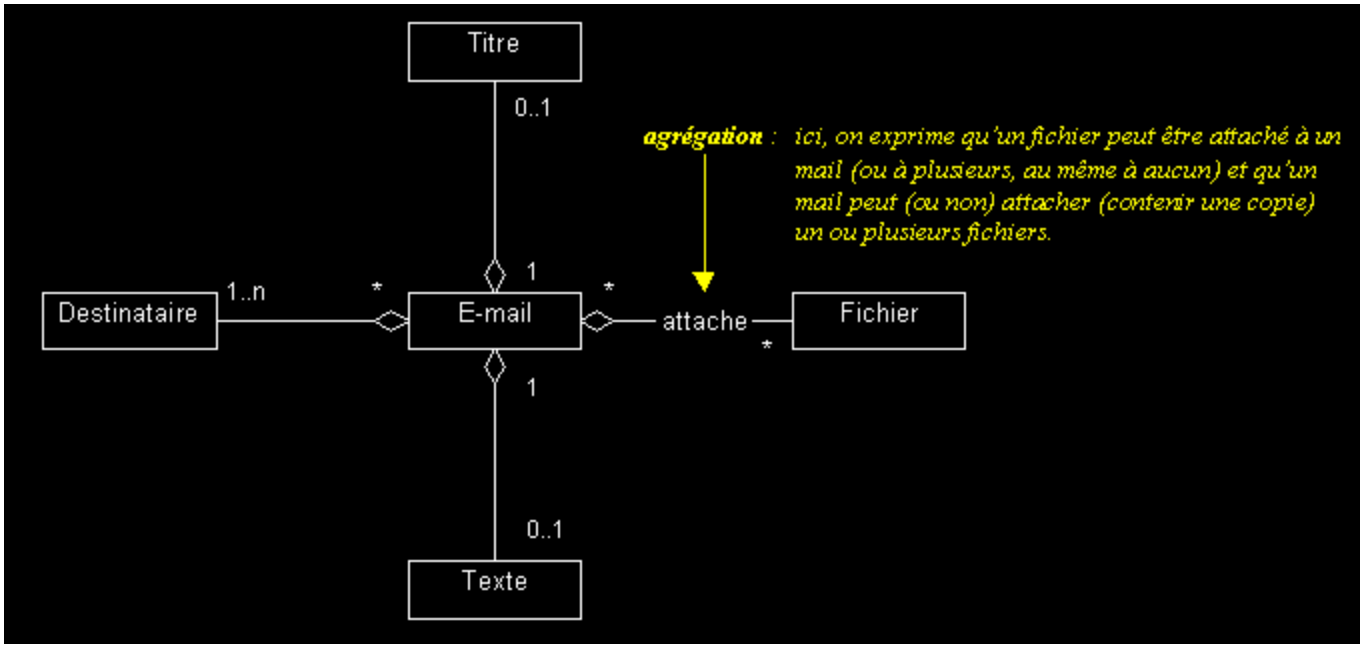
Les diagrammes de classe



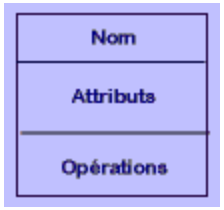
Agrégation

Une agrégation peut notamment (mais pas nécessairement) exprimer :

- qu'une classe (un "élément") fait partie d'une autre ("l'agrégat"),
- qu'un changement d'état d'une classe, entraîne un changement d'état d'une autre,
- qu'une action sur une classe, entraîne une action sur une autre.

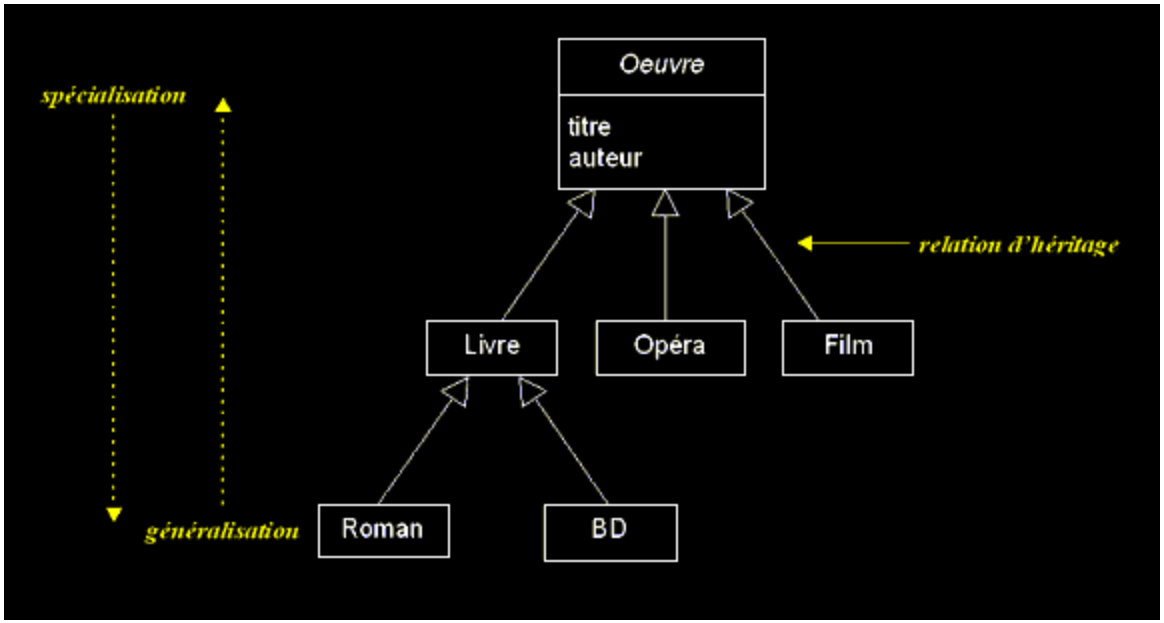


Les diagrammes de classe

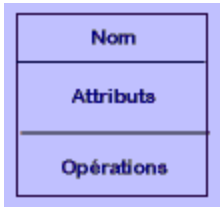


Héritage

L'héritage (spécialisation et généralisation) permet la classification des objets.

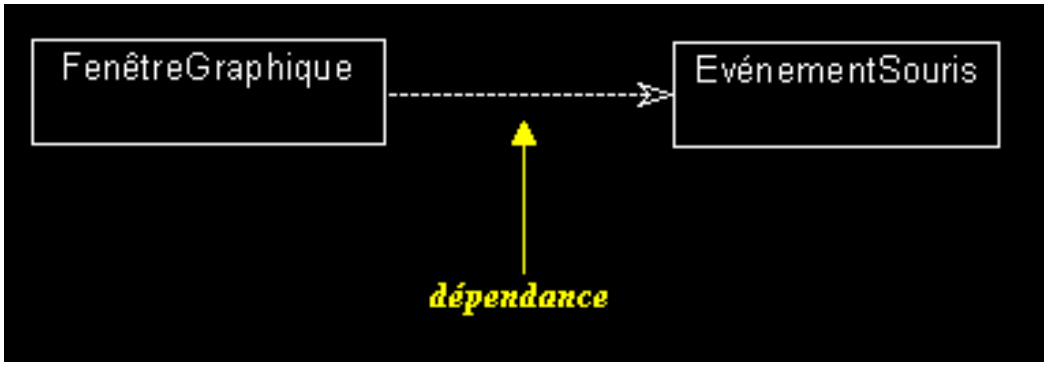


Les diagrammes de classe

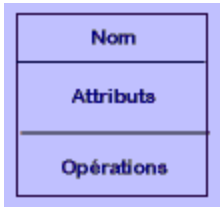


Relation de dépendance

Relation d'utilisation unidirectionnelle et d'obsolescence (une modification de l'élément dont on dépend, peut nécessiter une mise à jour de l'élément dépendant).

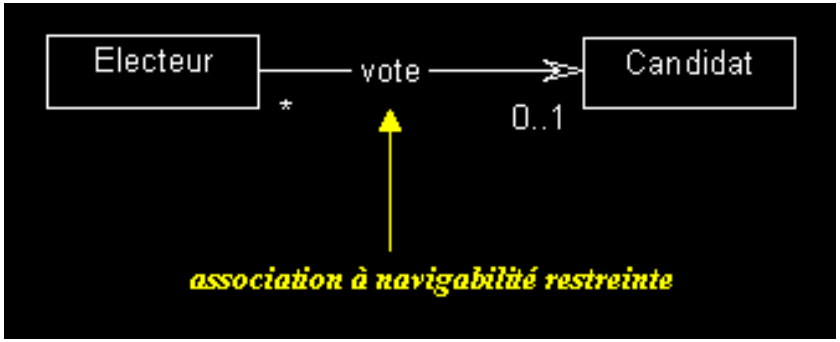


Les diagrammes de classe

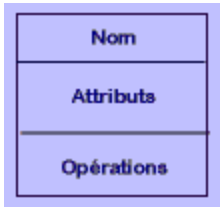


Association à navigabilité restreinte

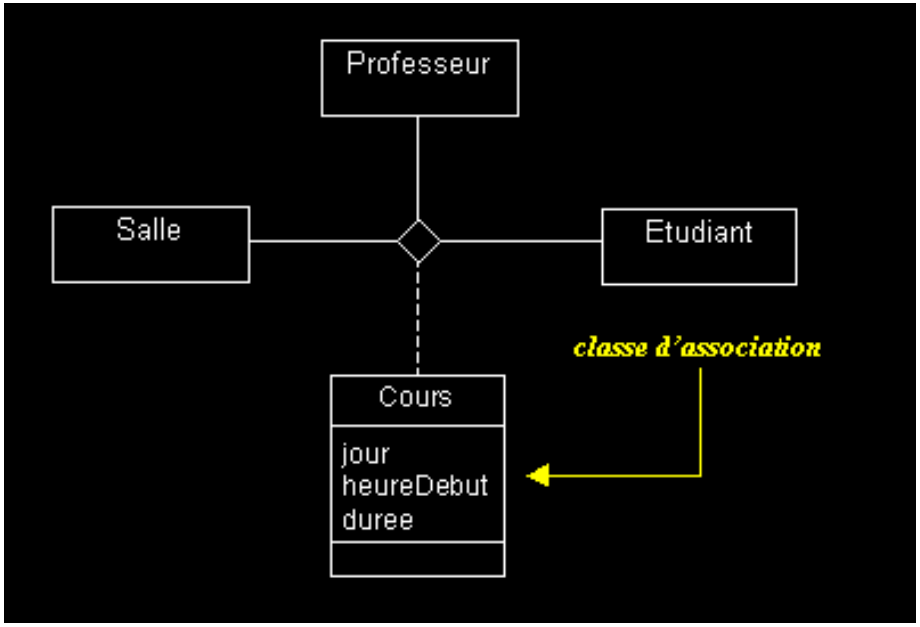
Par défaut, une association est navigable dans les deux sens. La réduction de la portée de l'association est souvent réalisée en phase d'implémentation, mais peut aussi être exprimée dans un modèle pour indiquer que les instances d'une classe ne "connaissent" pas les instances d'une autre.



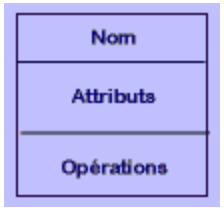
Les diagrammes de classe



Classe d'association : il s'agit d'une classe qui réalise la navigation entre les instances d'autres classes.

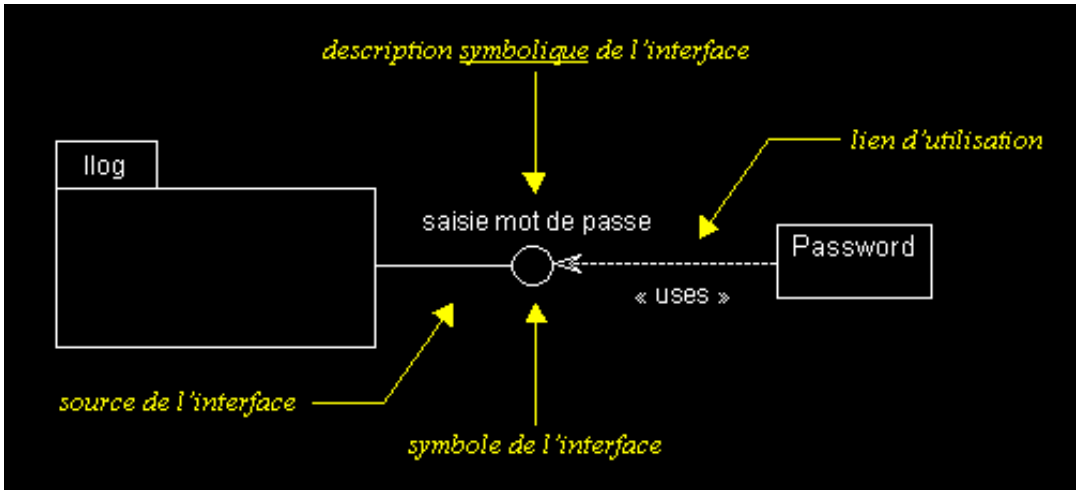


Les diagrammes de classe

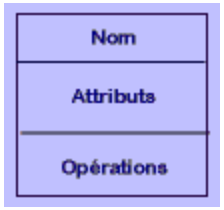


Interface

Une interface fournit une vue totale ou partielle d'un ensemble de services offerts par une classe, un paquetage ou un composant. Les éléments qui utilisent l'interface peuvent exploiter tout ou partie de l'interface.

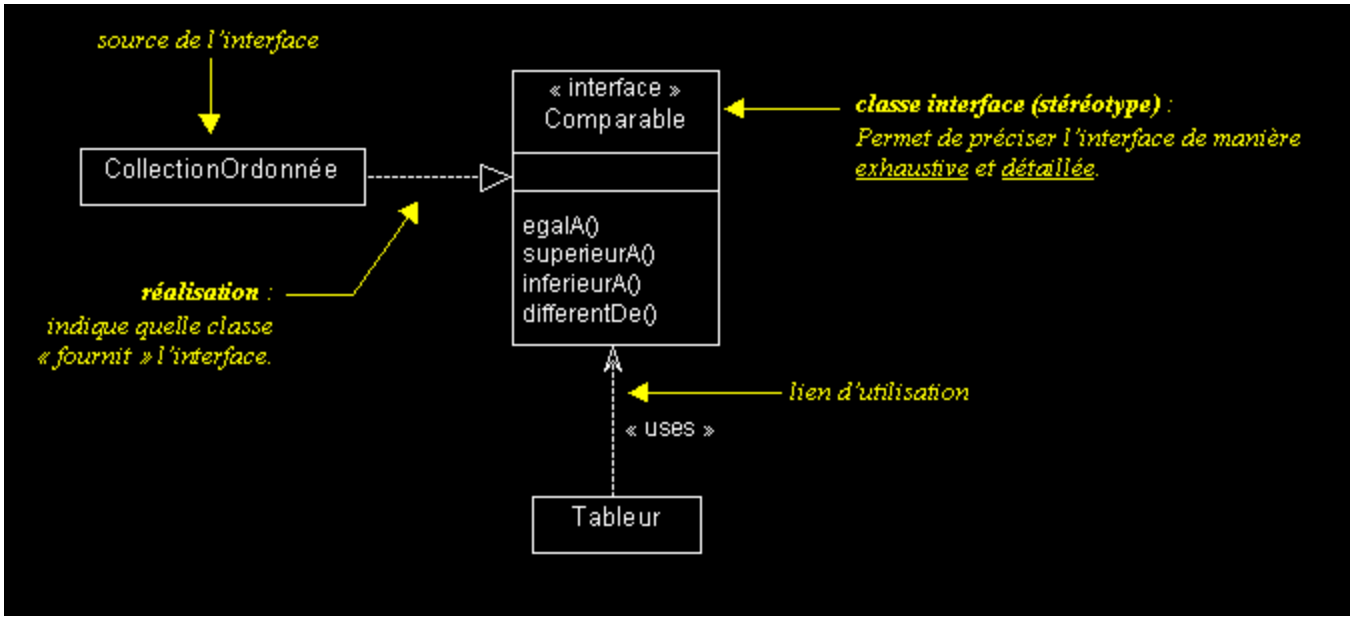


Les diagrammes de classe

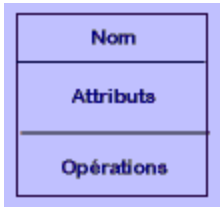


Interface

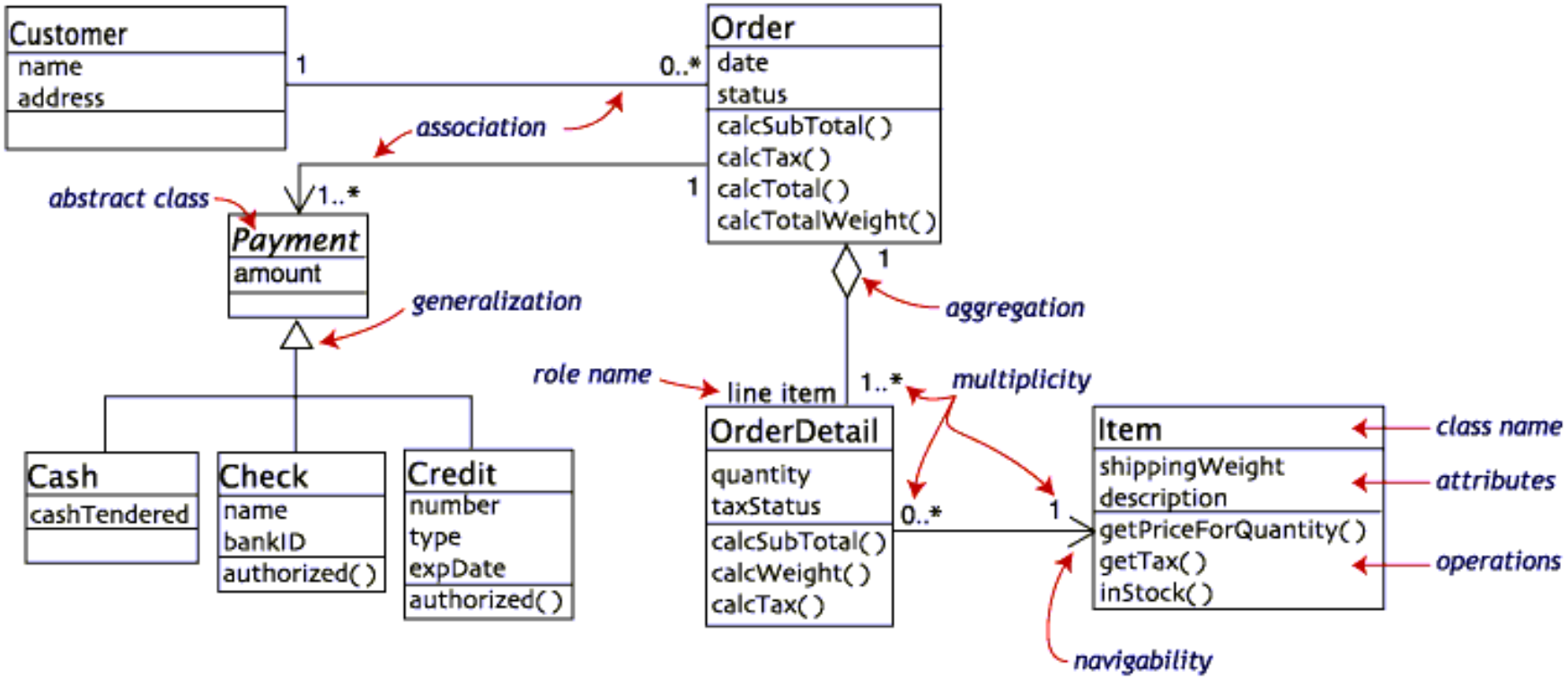
Dans un modèle UML, le symbole "interface" sert à identifier de manière explicite et symbolique les services offerts par un élément et l'utilisation qui en est faite par les autres éléments.



Les diagrammes de classe



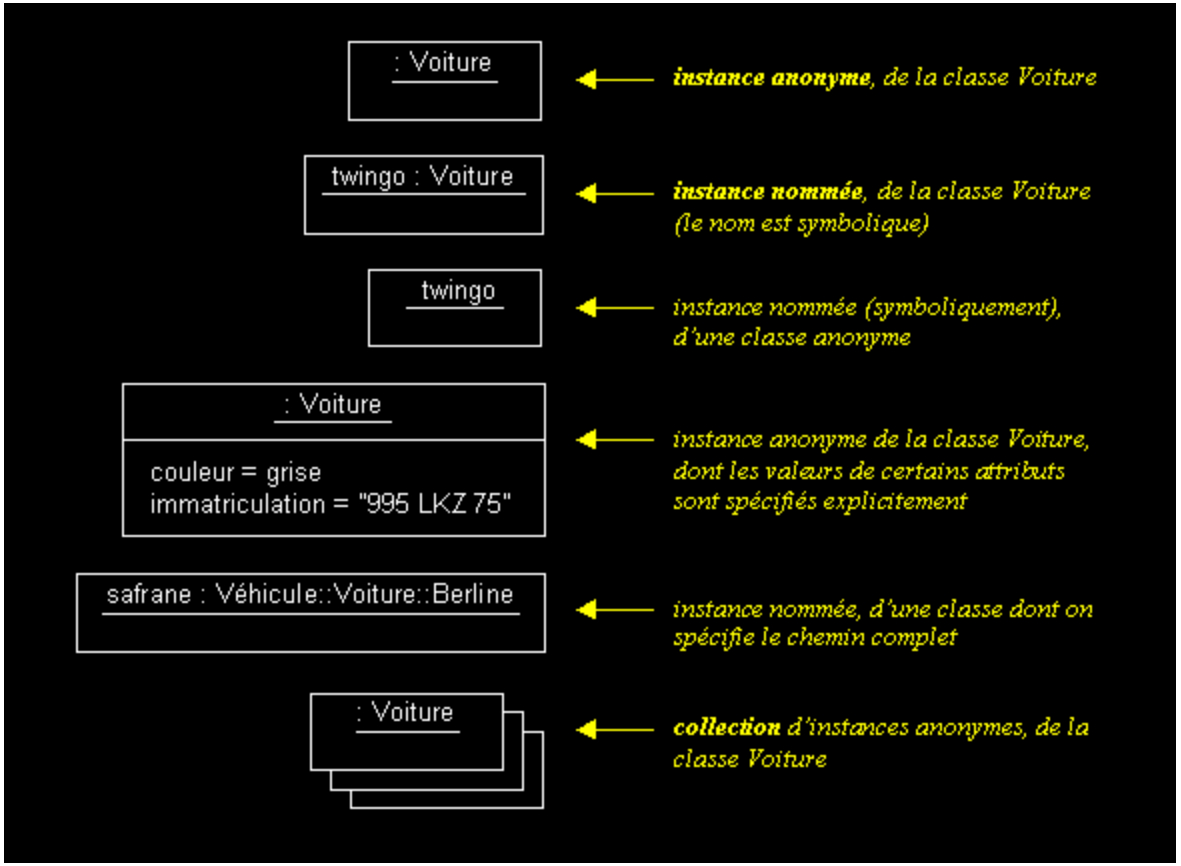
Exemple



The class diagram above models a customer order from a retail catalog. The central class is the **Order**. Associated with it are the **Customer** making the purchase and the **Payment**. A **Payment** is one of three kinds: **Cash**, **Check**, or **Credit**. The order contains **OrderDetails** (line items), each with its associated **Item**.

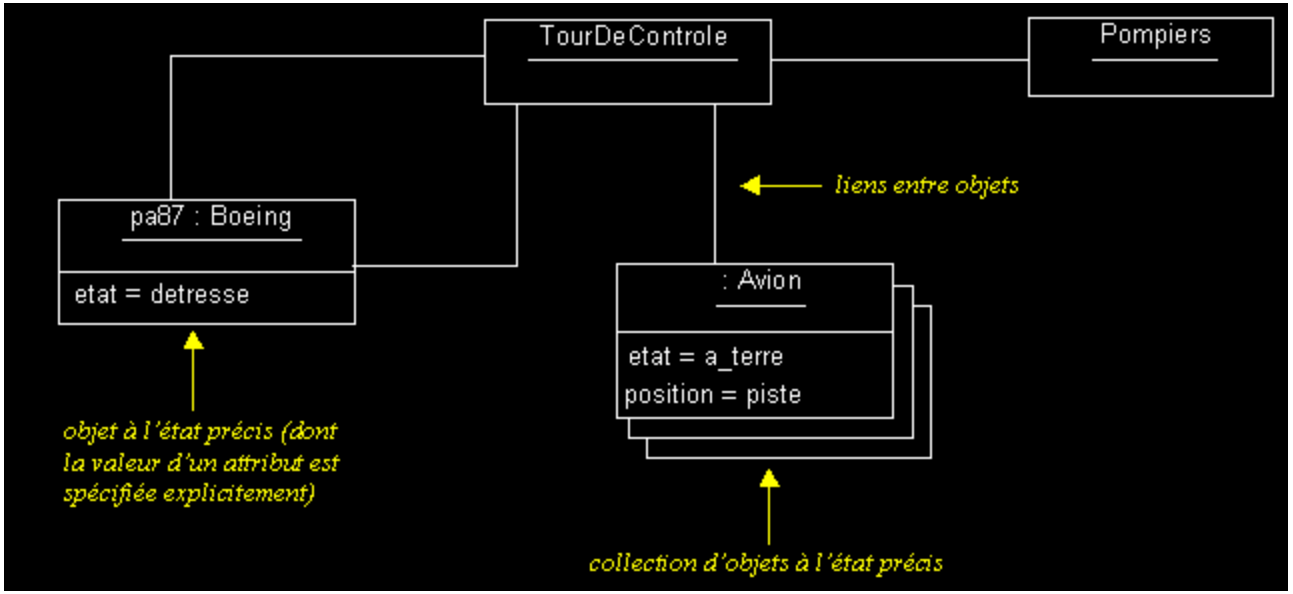
Les diagrammes d'objet

Ce type de diagramme UML montre des objets (instances de classes dans un état particulier) et des liens (relations sémantiques) entre ces objets.



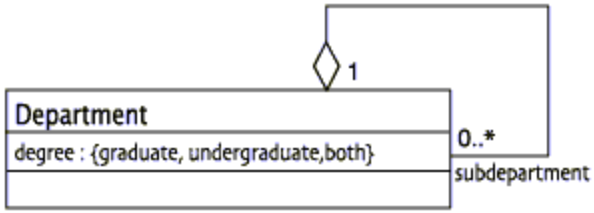
Les diagrammes d’objet

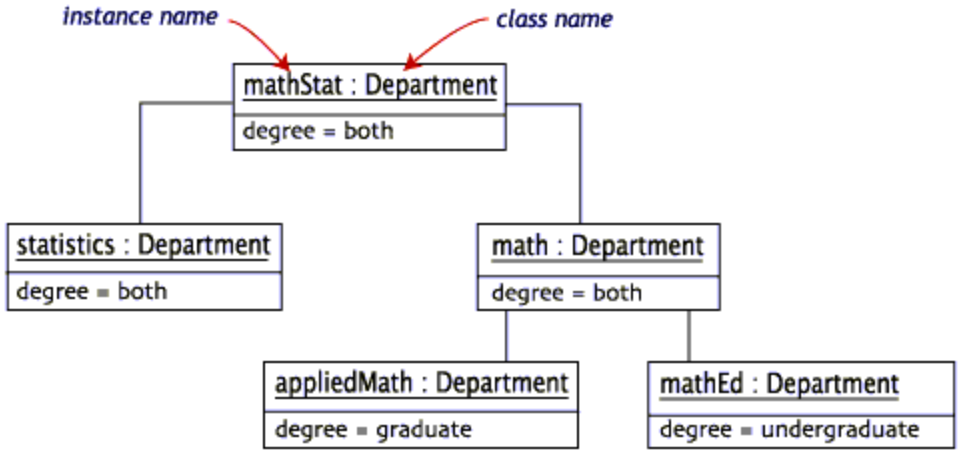
Exemple



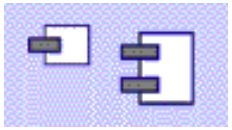
Les diagrammes d’objet

Exemple

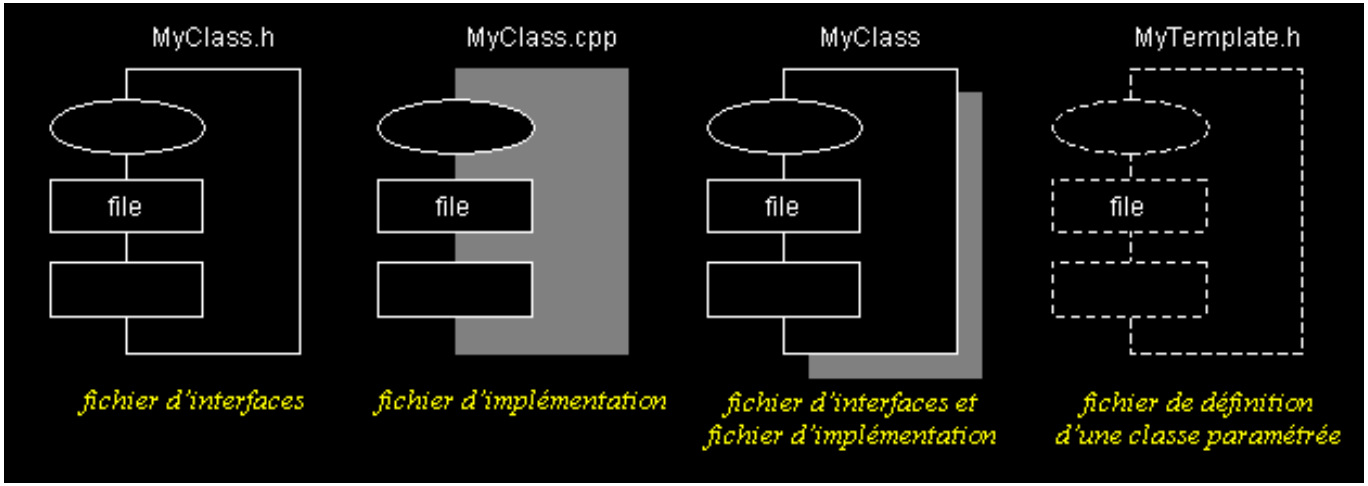




Les diagrammes de composants

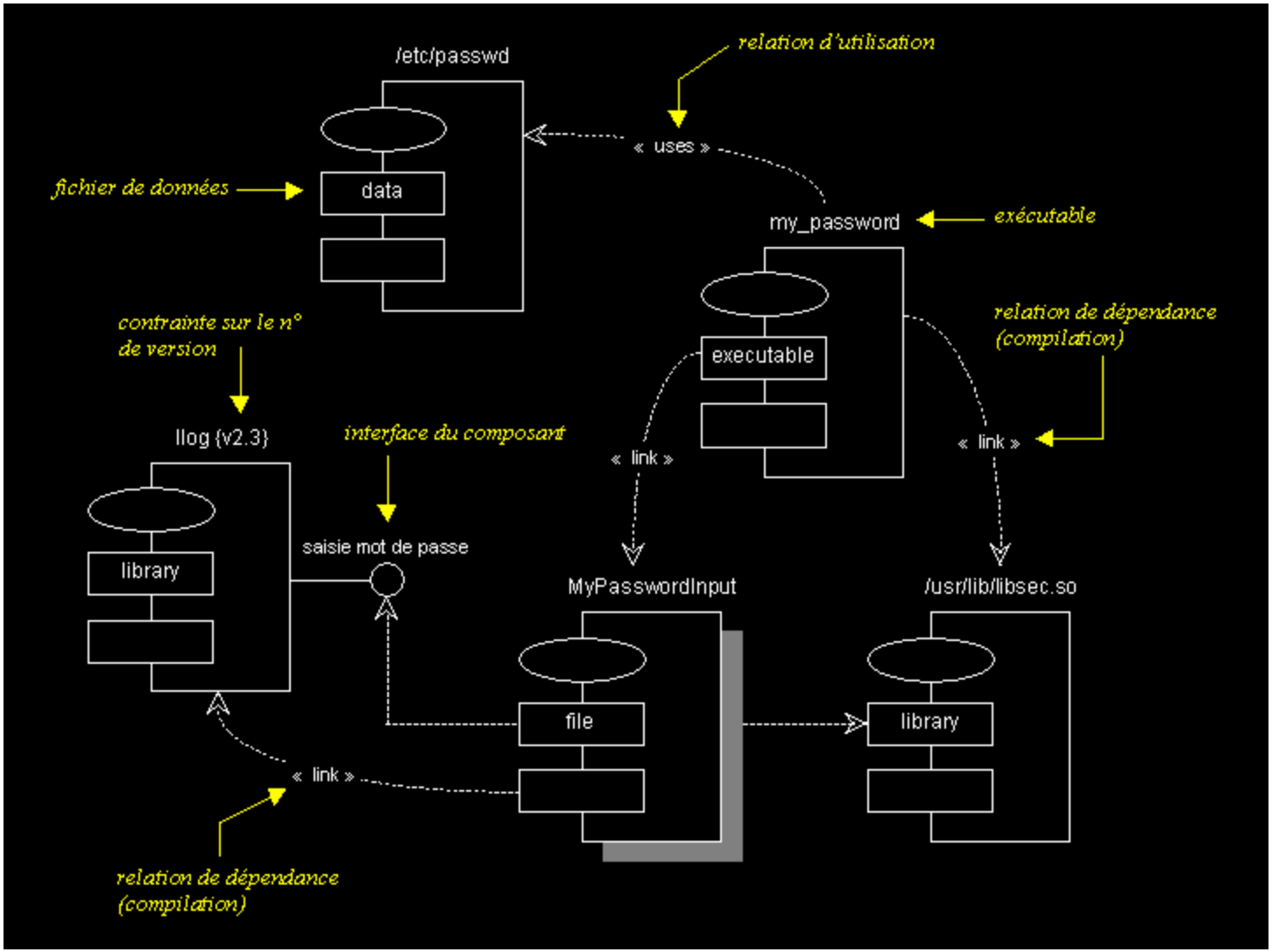


- Les diagrammes de composants permettent de décrire l'architecture physique et statique d'une application en terme de modules : fichiers sources, librairies, exécutables, etc.
- Ils montrent la mise en oeuvre physique des modèles de la vue logique avec l'environnement de développement.

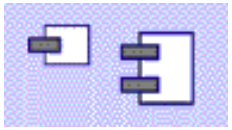


Les diagrammes de composants

Exemple



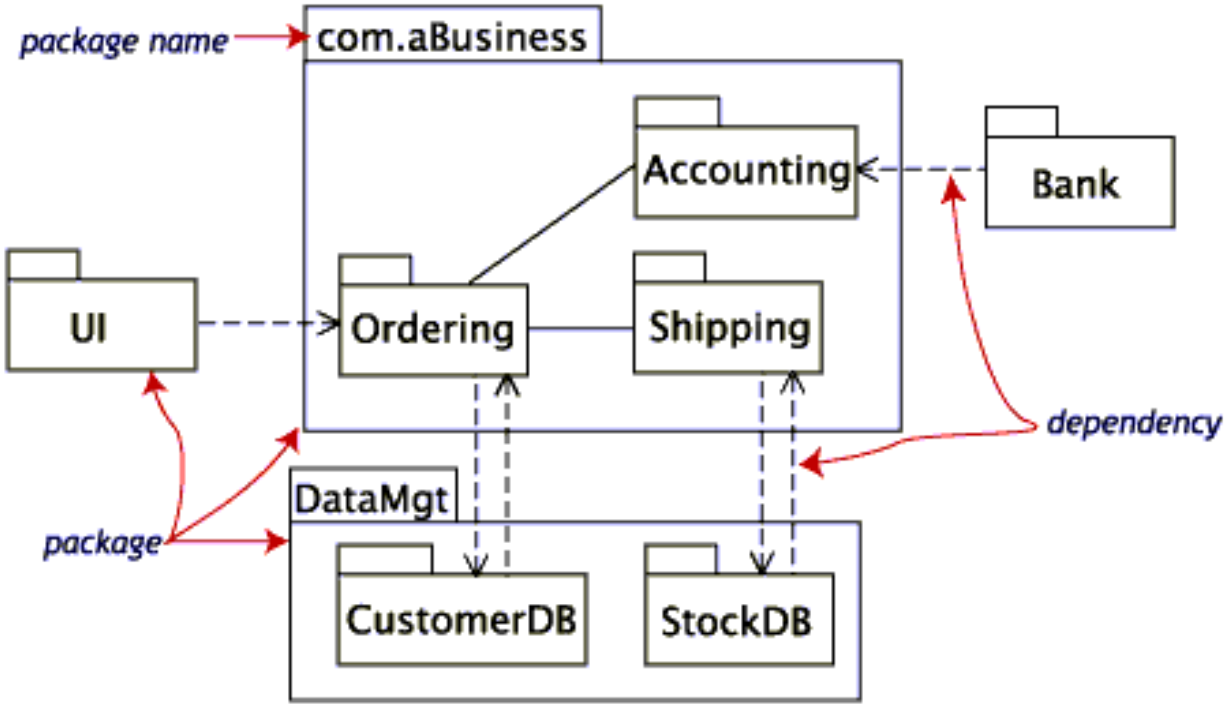
Les diagrammes de composantes



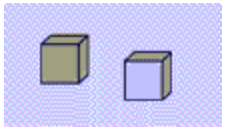
- Les paquetages sont des éléments d'organisation des modèles.
- Ils regroupent des éléments de modélisation, selon des critères purement logiques.
- Ils permettent d'encapsuler des éléments de modélisation (ils possèdent une interface).
- Ils permettent de structurer un système en catégories (vue logique) et sous-systèmes (vue des composants).
- Ils servent de "briques" de base dans la construction d'une architecture.
- Ils représentent le bon niveau de granularité pour la réutilisation.

Les diagrammes de paquetages

Exemple

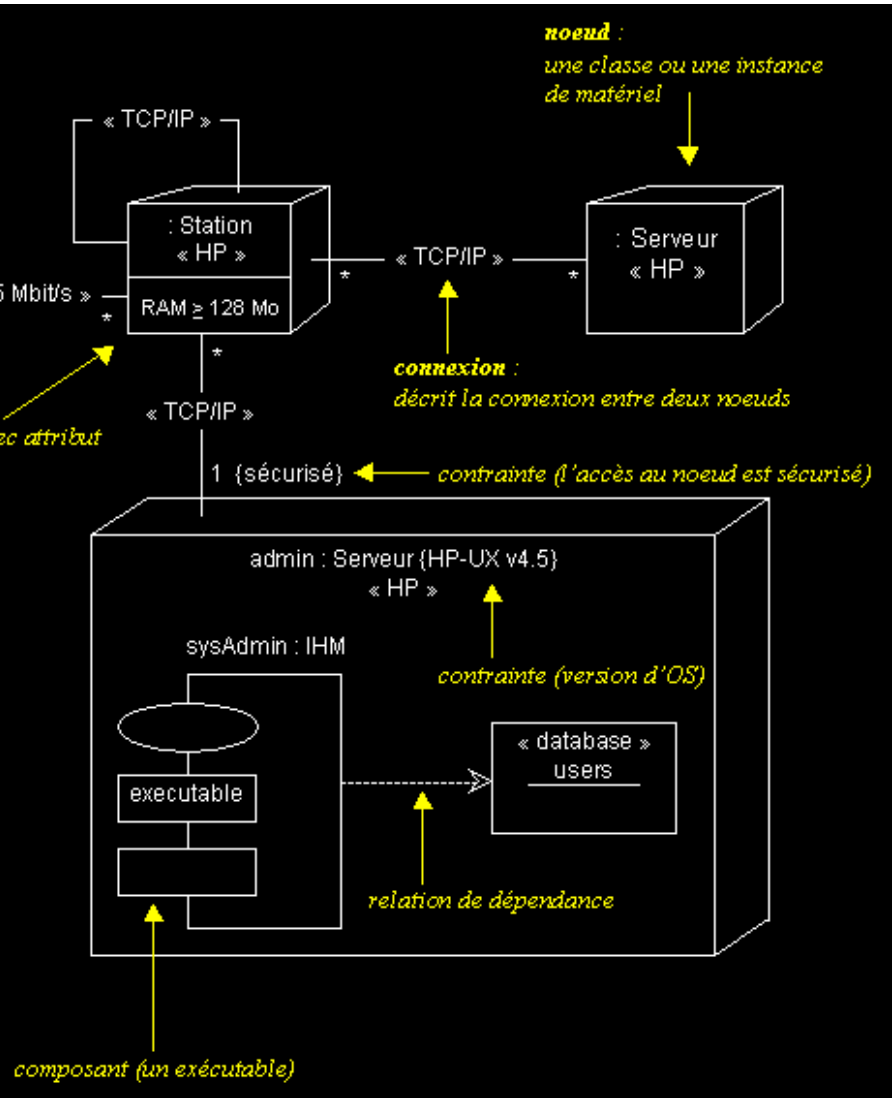


Les diagrammes de déploiement



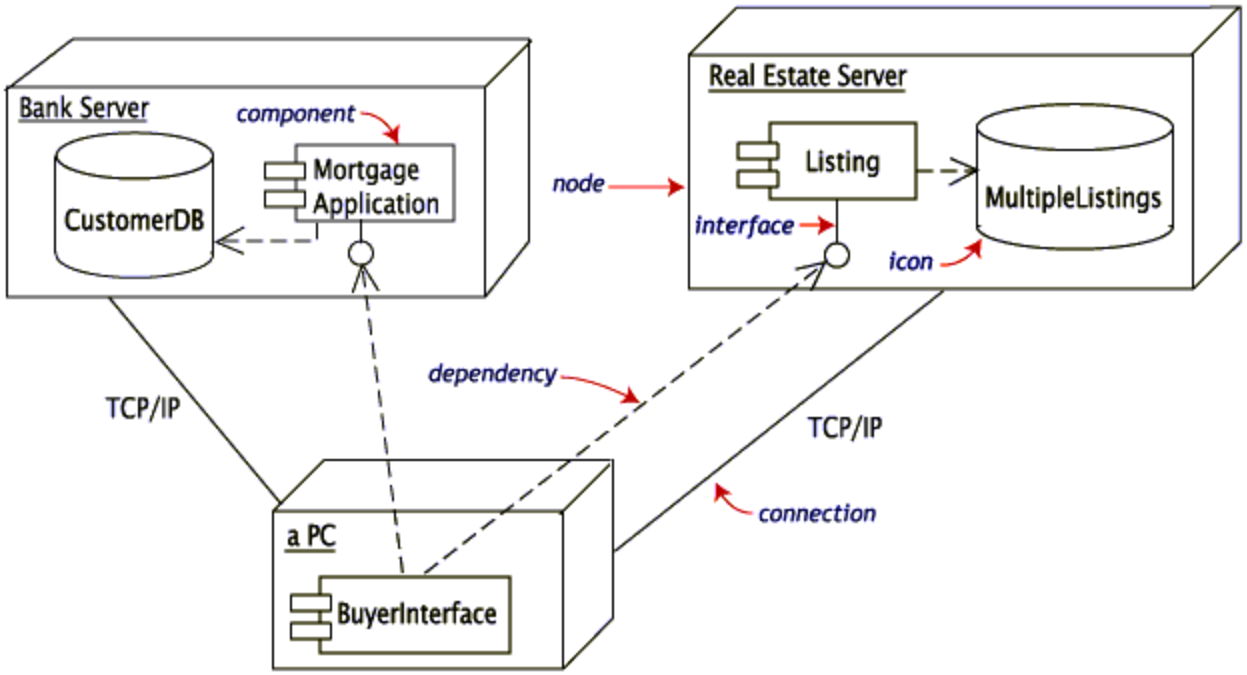
- Les diagrammes de déploiement montrent la disposition physique des matériels qui composent le système et la répartition des composants sur ces matériels.
- Les ressources matérielles sont représentées sous forme de noeuds.
- Les noeuds sont connectés entre eux, à l'aide d'un support de communication. La nature des lignes de communication et leurs caractéristiques peuvent être précisées.

Les diagrammes de déploiement



Les diagrammes de déploiement

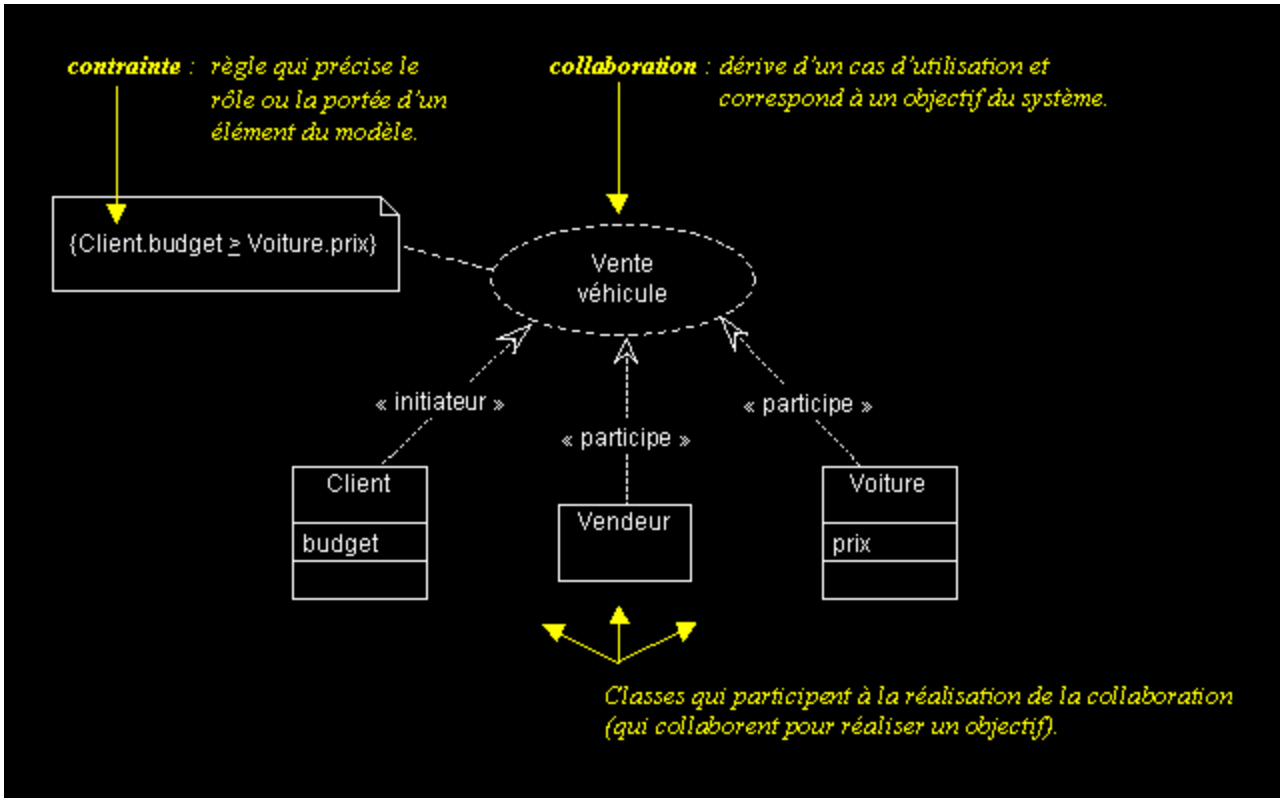
Exemple



The deployment diagram shows the relationships among software and hardware components involved in real estate transactions.

Classes et Cas d’utilisation

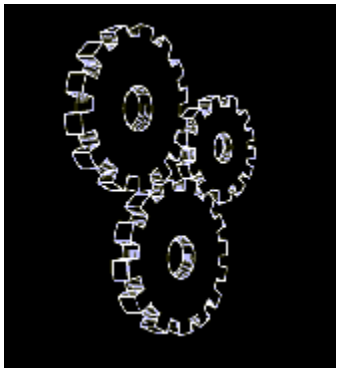
Représenter les classes qui participent à la réalisation d'un cas d'utilisation.



Les diagrammes de UML

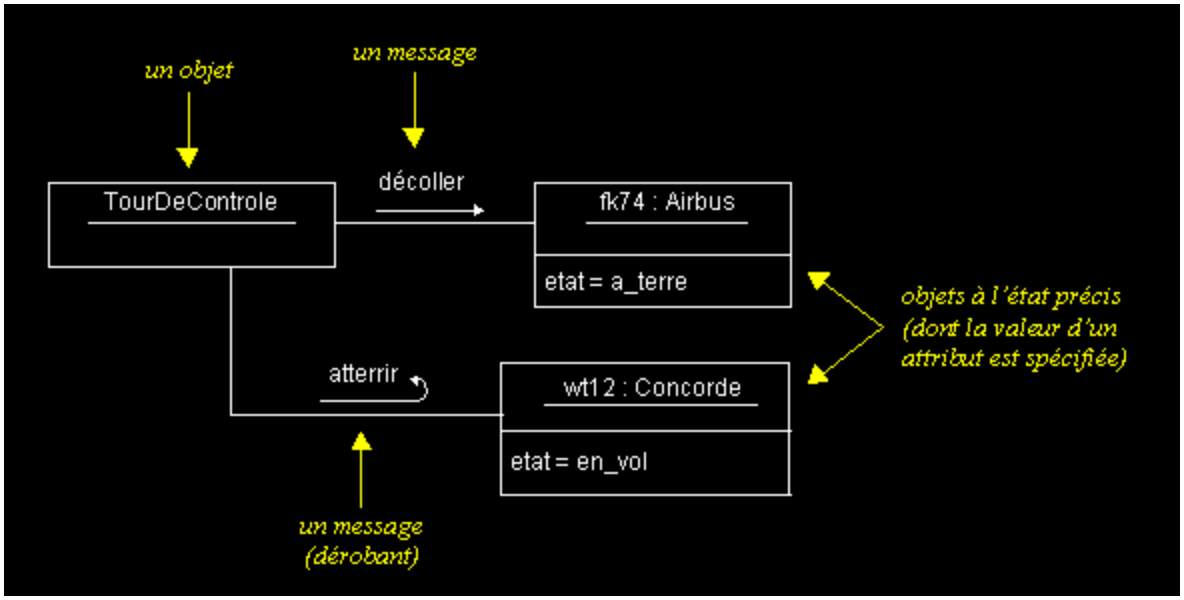
Vues dynamiques du système

- diagrammes de collaboration
- diagrammes de séquence
- diagrammes d'états-transitions
- diagrammes d'activités



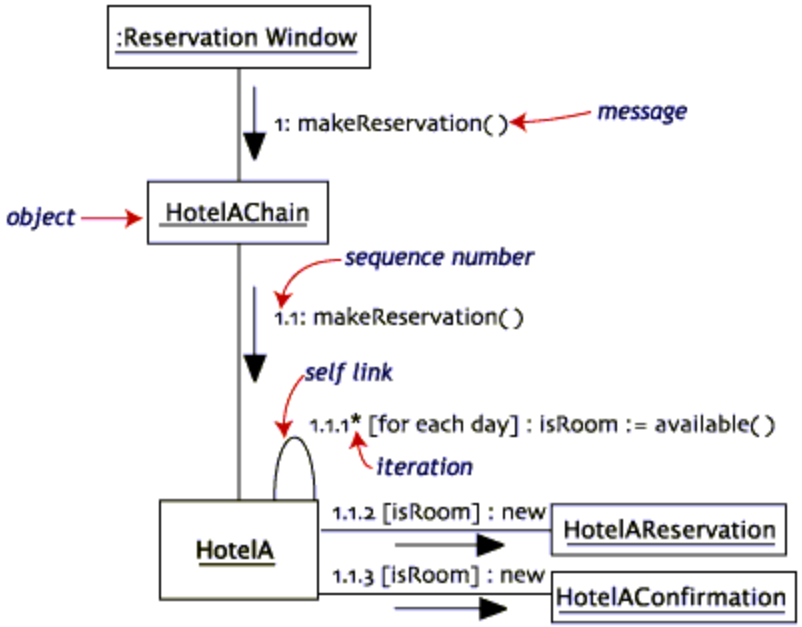
Les diagrammes de collaboration

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.



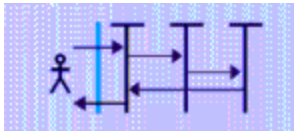
Les diagrammes de collaboration

Exemple



Each message in a collaboration diagram has a **sequence number**. The top-level message is numbered 1. Messages at the same level (sent during the same call) have the same decimal prefix but suffixes of 1, 2, etc. according to when they occur.

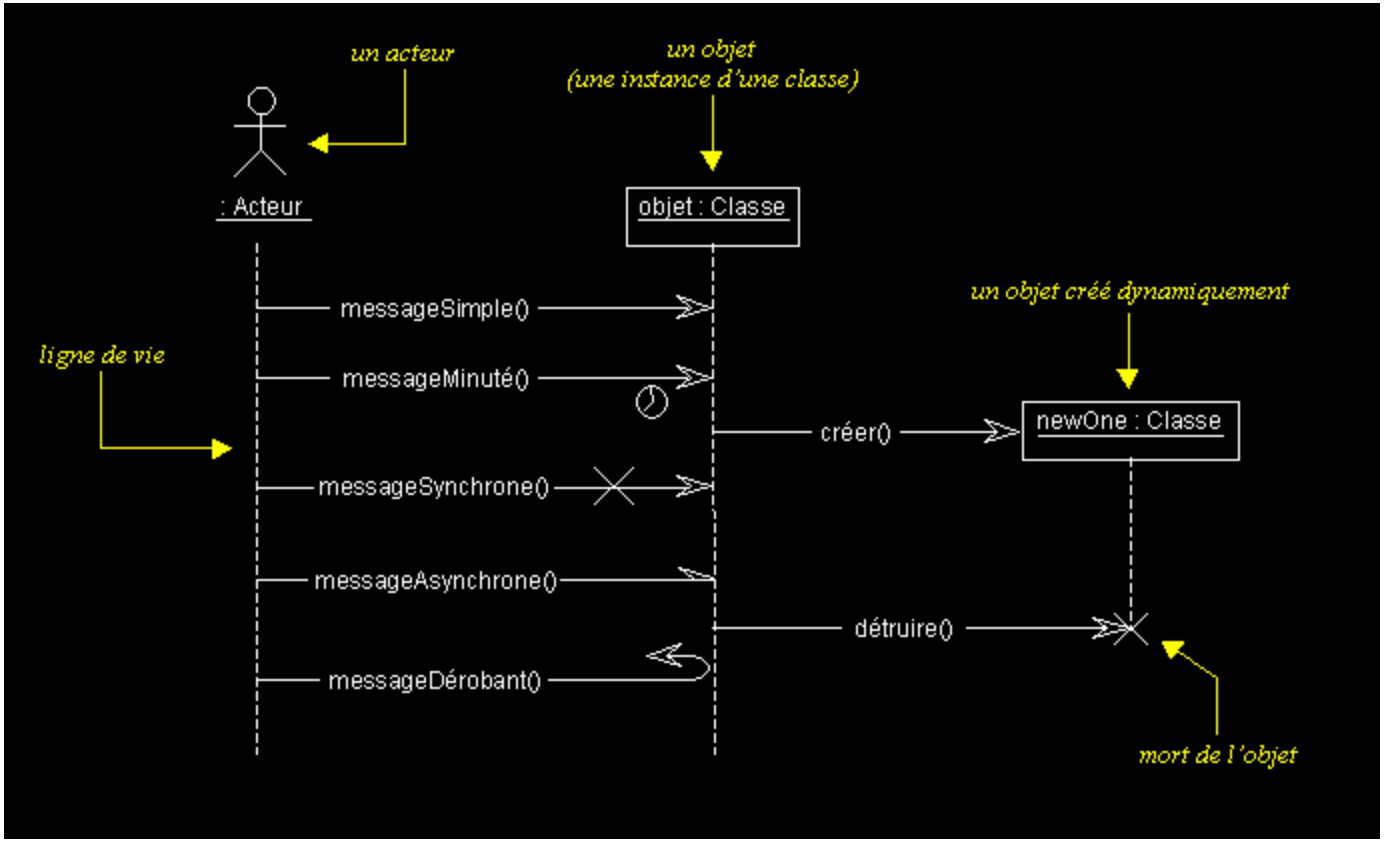
Les diagrammes de séquence

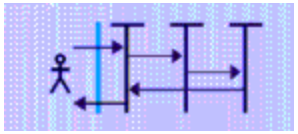


- Les diagrammes de séquences permettent de représenter des collaborations entre objets selon un point de vue temporel, on y met l'accent sur la chronologie des envois de messages.
- Contrairement au diagramme de collaboration, on n'y décrit pas le contexte ou l'état des objets, la représentation se concentre sur l'expression des interactions.
- Les diagrammes de séquences peuvent servir à illustrer un cas d'utilisation.
- L'ordre d'envoi d'un message est déterminé par sa position sur l'axe vertical du diagramme ; le temps s'écoule "de haut en bas" de cet axe.
- La disposition des objets sur l'axe horizontal n'a pas de conséquence pour la sémantique du diagramme.
- Les diagrammes de séquences et les diagrammes d'état-transitions sont les vues dynamiques les plus importantes d'UML.

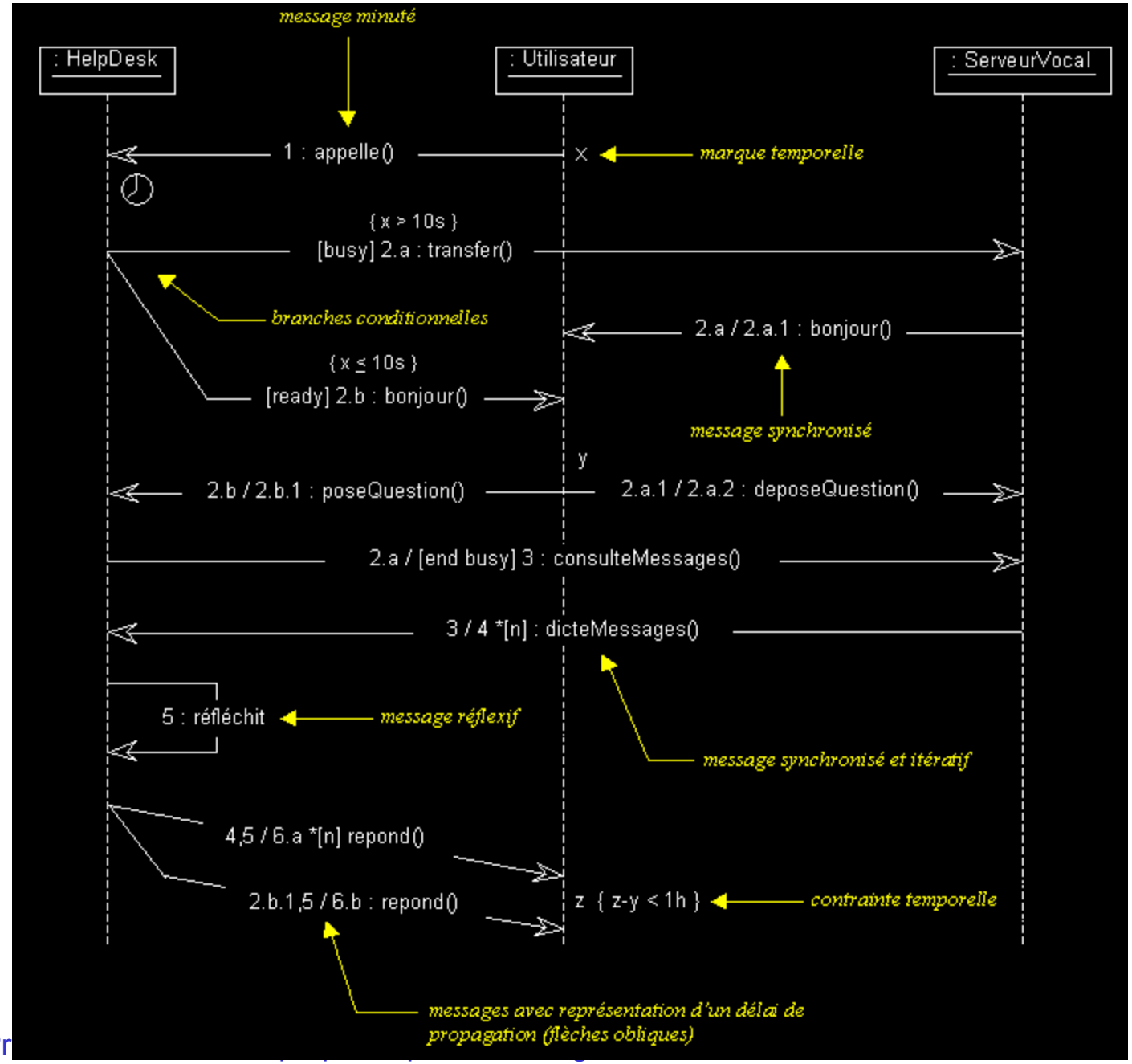
Les diagrammes de séquence

Exemple



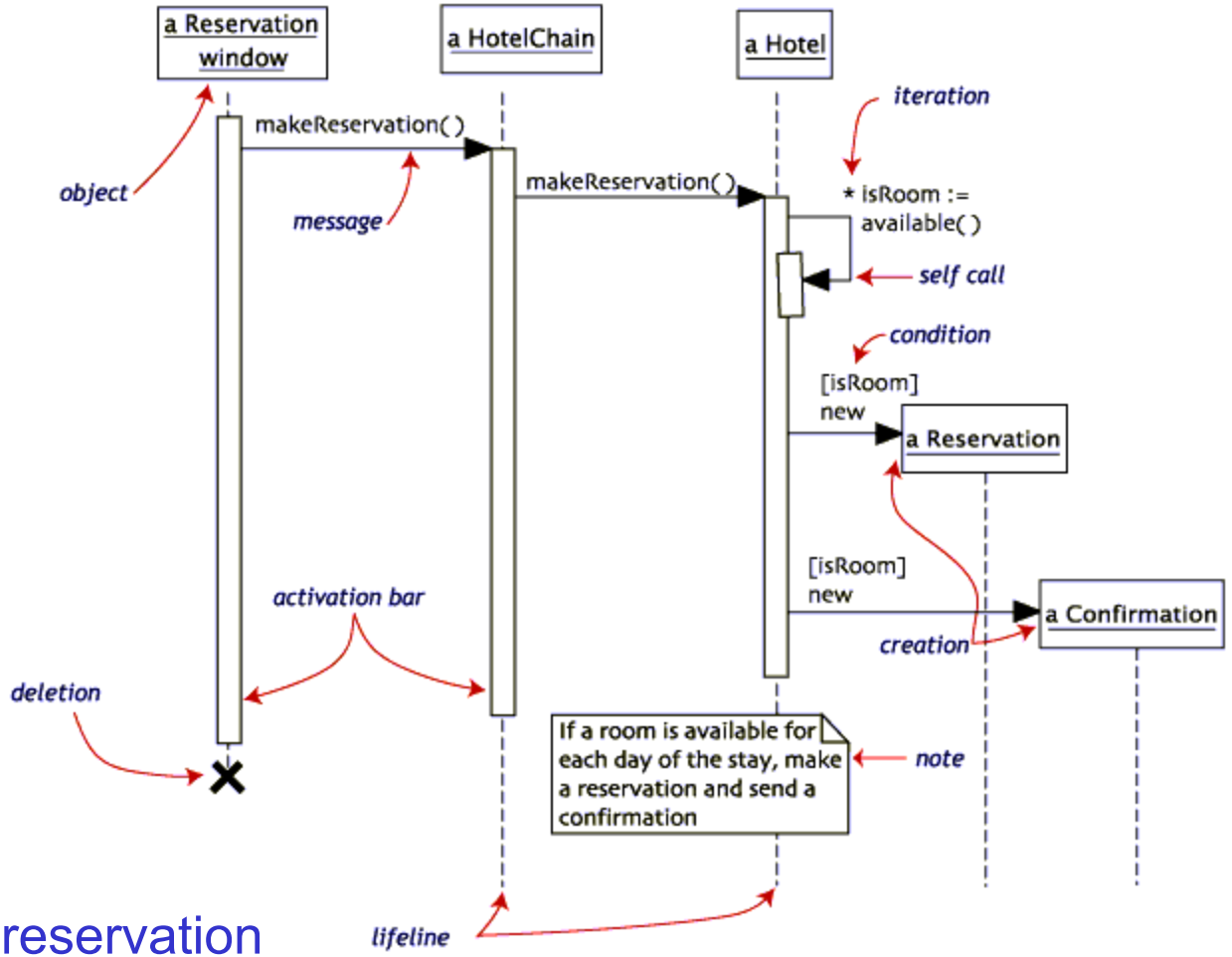


Exemple


message minuté
: HelpDesk
: Utilisateur
: ServeurVocal
1 : appelle()
x
marque temporelle
{ x > 10s }
[busy] 2.a : transfer()
branches conditionnelles
2.a / 2.a.1 : bonjour()
{ x ≤ 10s }
[ready] 2.b : bonjour()
message synchronisé
y
2.b / 2.b.1 : poseQuestion()
2.a.1 / 2.a.2 : deposeQuestion()
2.a / [end busy] 3 : consulteMessages()
3 / 4 *[n] : dicteMessages()
5 : réfléchit
message réflexif
message synchronisé et itératif
4,5 / 6.a *[n] repond()
2.b.1,5 / 6.b : repond()
z { z-y < 1h }
contrainte temporelle
messages avec représentation d'un délai de propagation (flèches obliques)

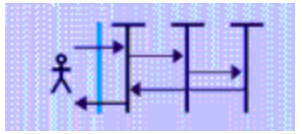
Les diagrammes de séquence

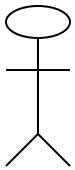
Exemple



Making a hotel reservation

Les diagrammes de séquence

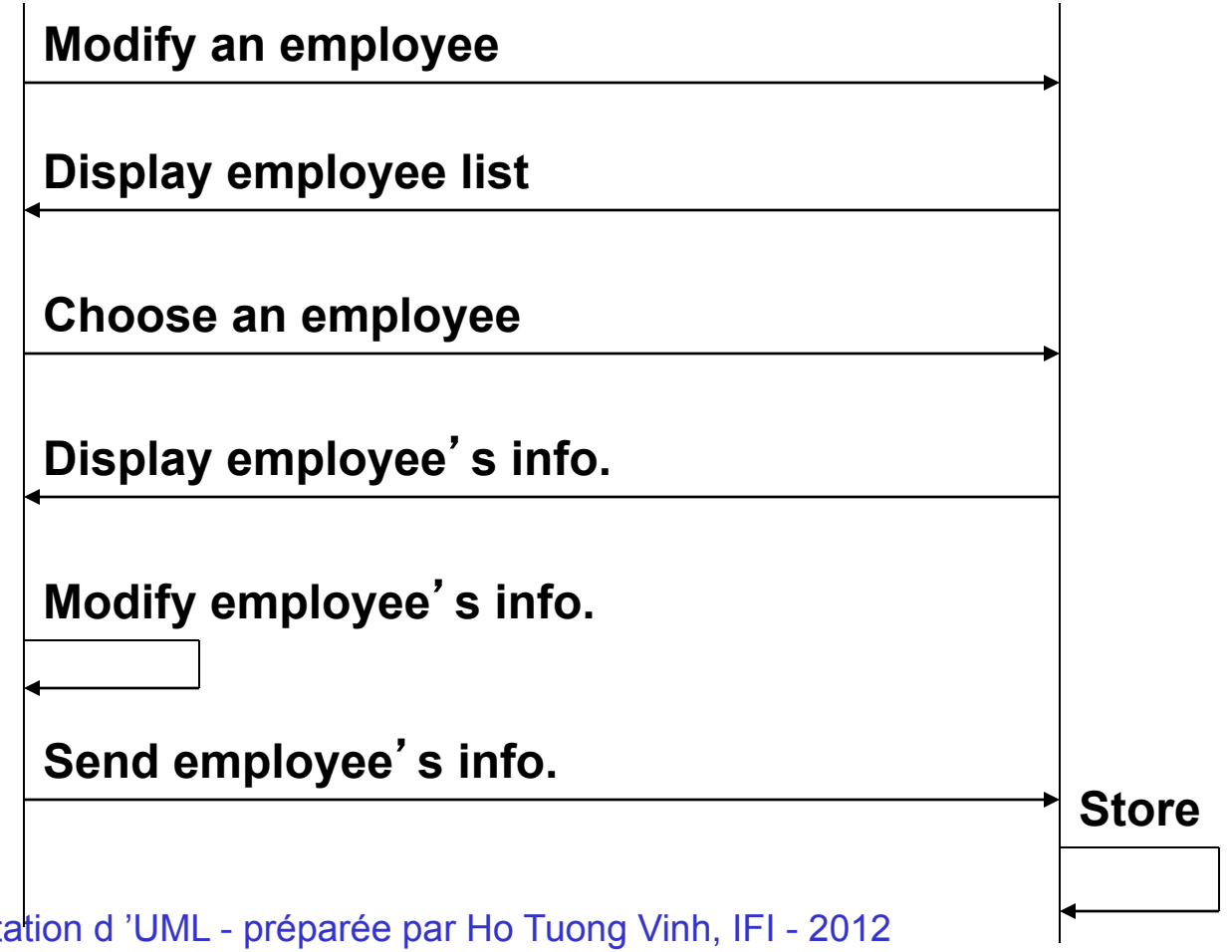




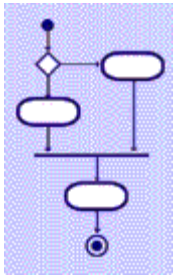
: Supervisor

: System

Exemple

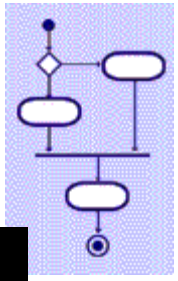


Les diagrammes d’activités

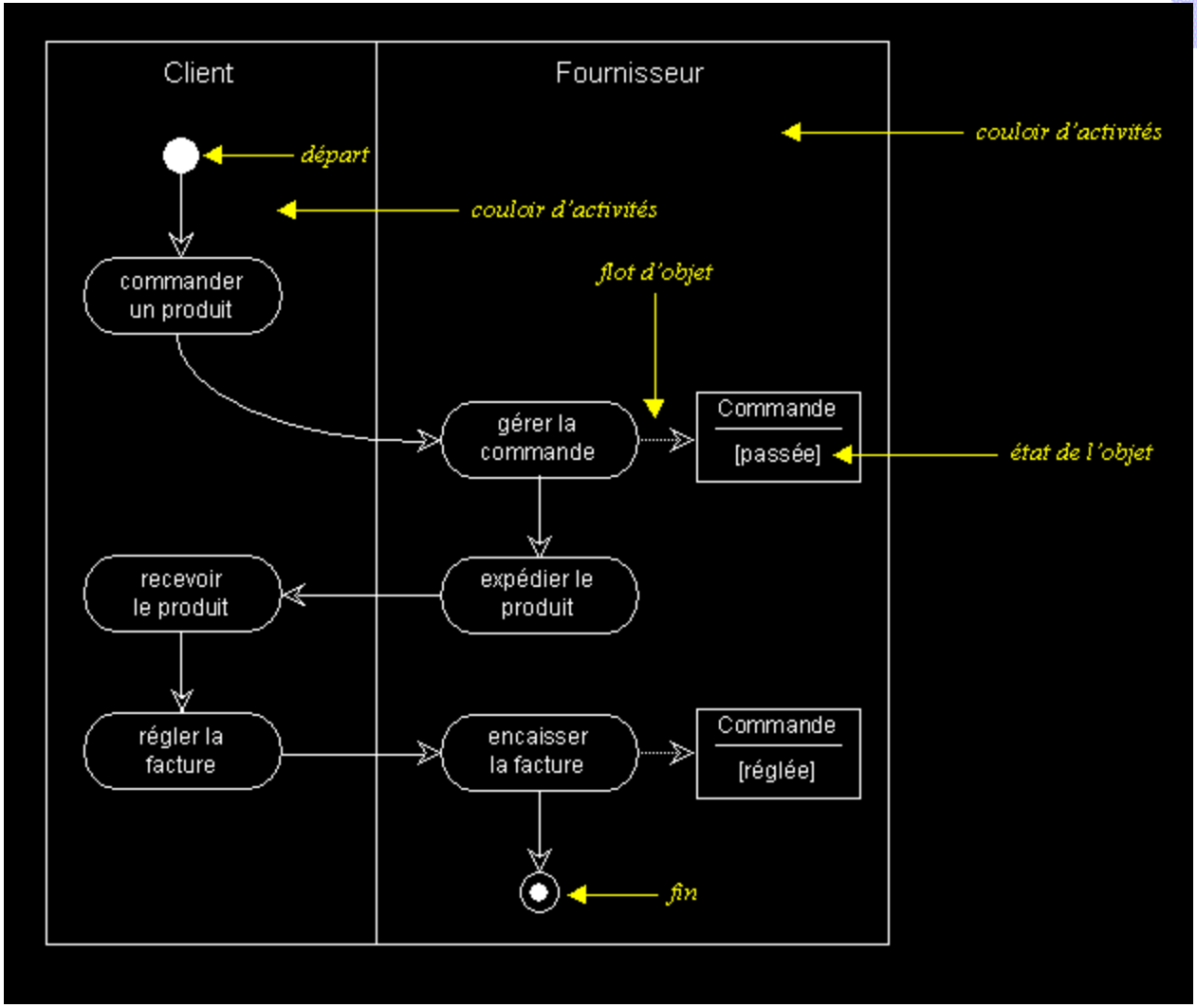


- UML permet de représenter graphiquement le comportement d'une méthode ou le déroulement d'un cas d'utilisation, à l'aide de diagrammes d'activités.
- Une activité représente une exécution d'un mécanisme, un déroulement d'étapes séquentielles.
- Le passage d'une activité vers une autre est matérialisé 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 diagrammes d’activités

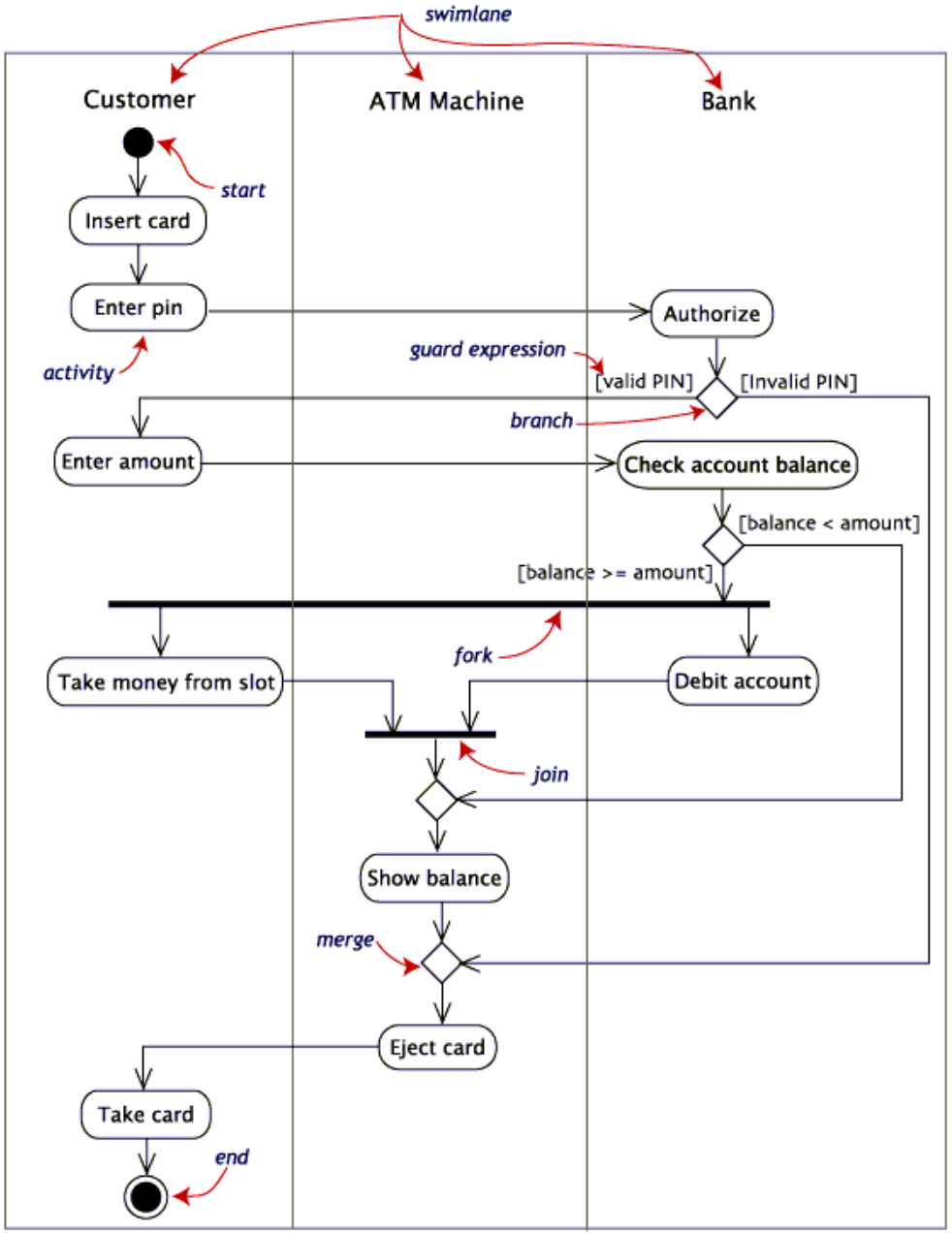


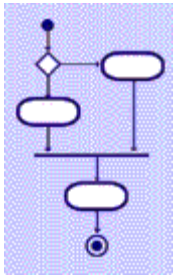
Exemple



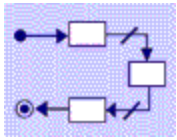
Exemple

Withdraw money from a bank account through an ATM

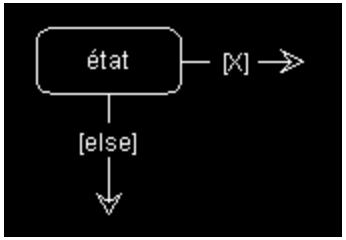




Les diagrammes d’états-transitions

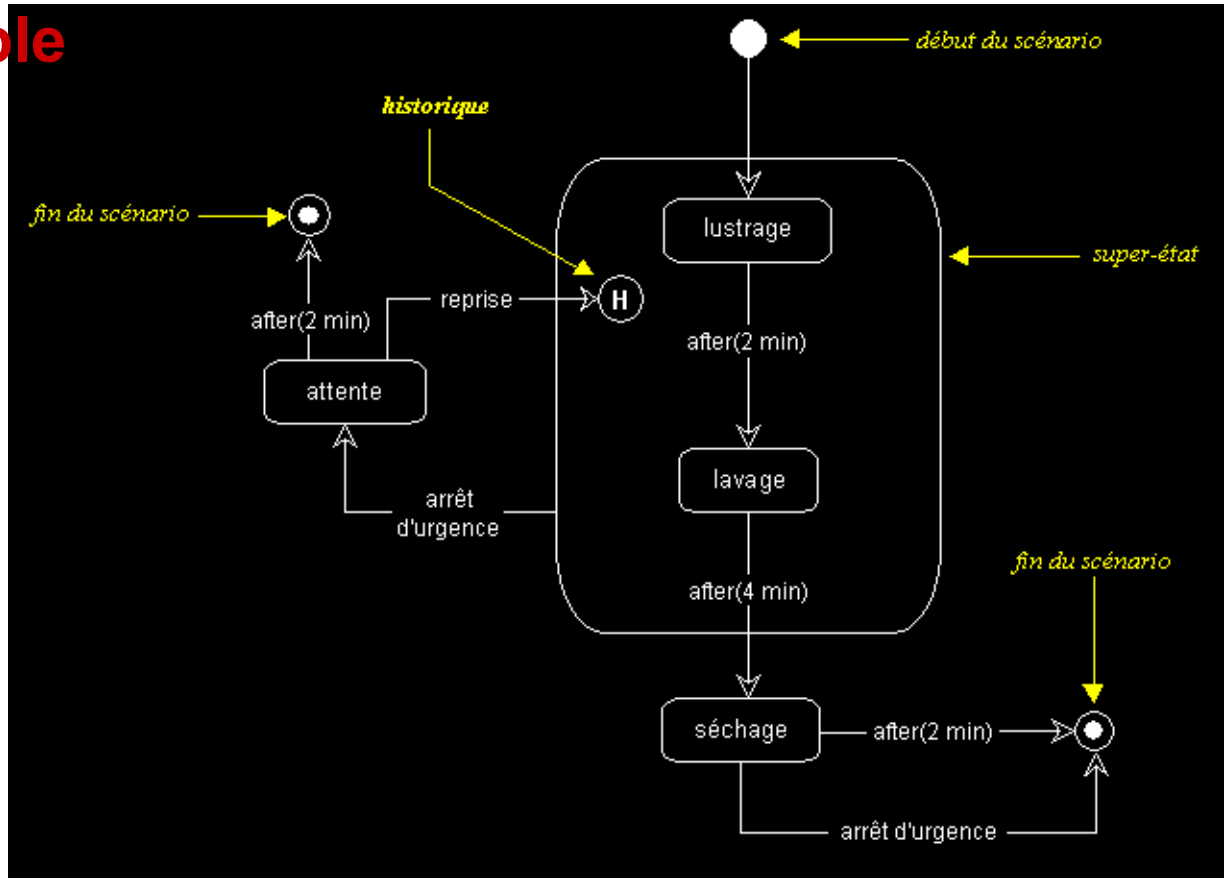


Les diagrammes d'états-transitions permettent de décrire les changements d'états d'un objet ou d'un composant, en réponse aux interactions avec d'autres objets/composants ou avec des acteurs.



Les diagrammes d'états-transitions

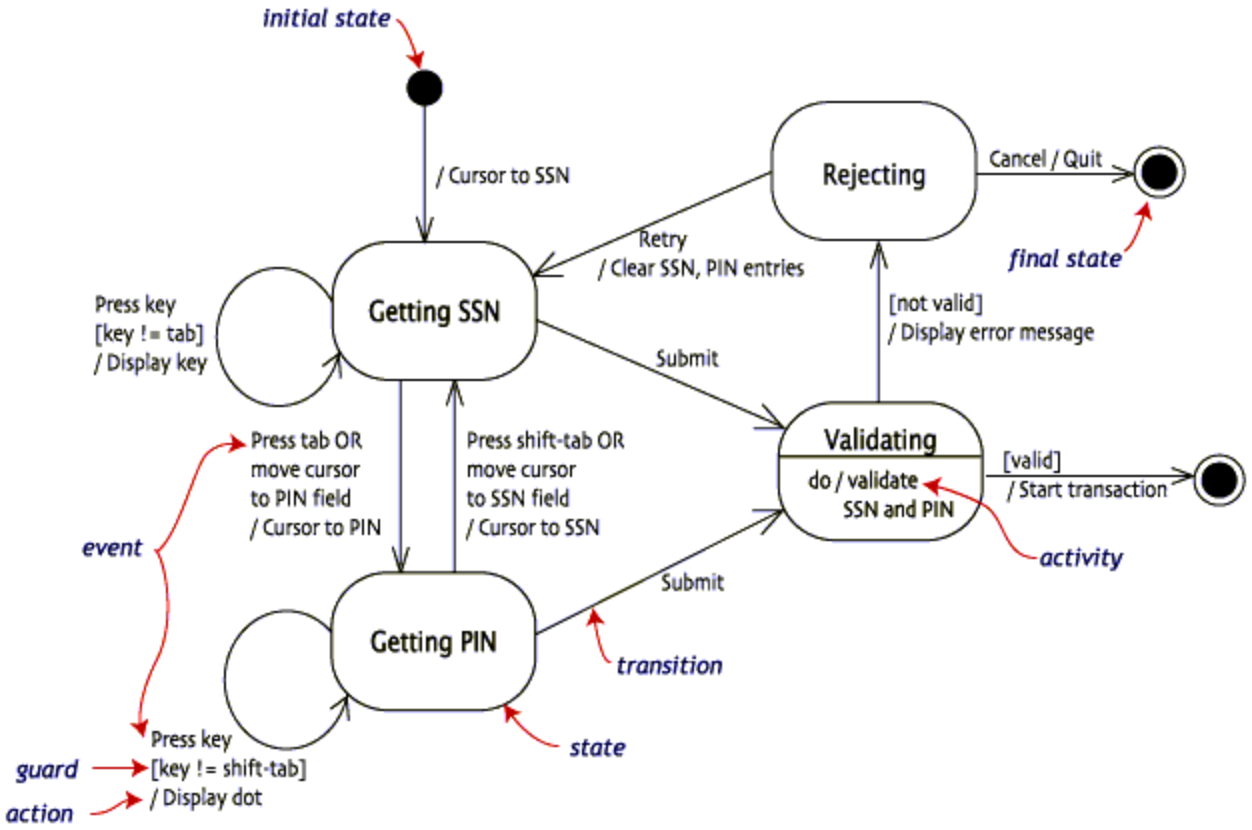
Exemple



En phase de lustrage ou de lavage, le client peut appuyer sur le bouton d'arrêt d'urgence. S'il appuie sur ce bouton, la machine se met en attente. Il a alors deux minutes pour reprendre le lavage ou le lustrage, sans quoi la machine s'arrête. En phase de séchage, le client peut aussi interrompre la machine. Mais dans ce cas, la machine s'arrêtera définitivement.

Les diagrammes d'états-transitions

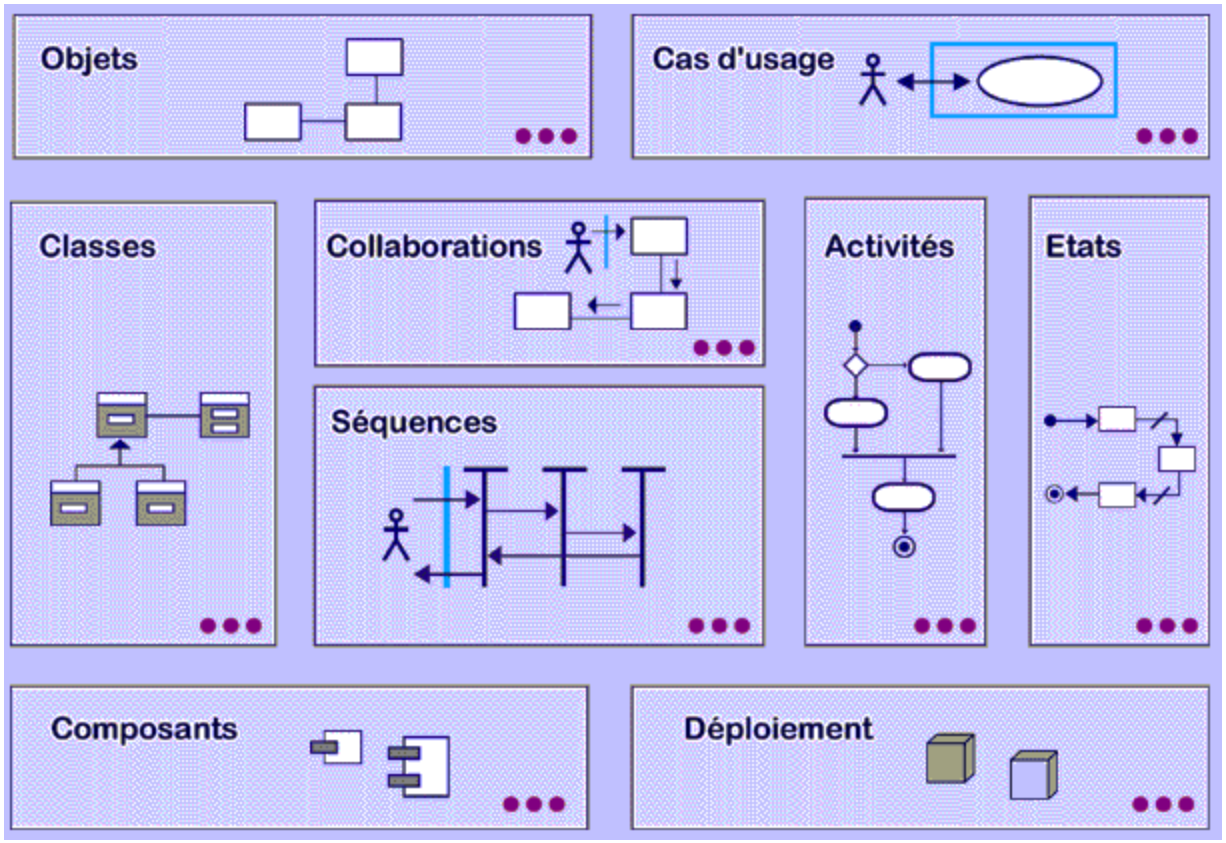
Exemple



This example diagram models the login part of an online banking system. Logging in consists of entering a valid social security number and personal id number, then submitting the information for validation.

Les diagrammes de UML

UML 1.1 fournit neuf diagrammes de base pour modéliser les différents aspects d'un système d'information:



Les diagrammes de UML 1.3

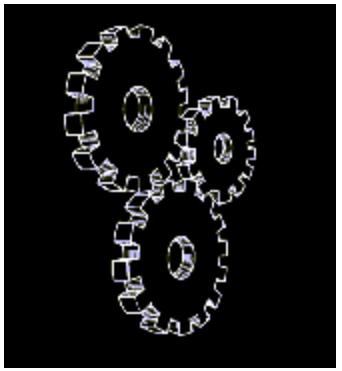
UML 2.0 fournit 13 diagrammes

Vues statiques du système

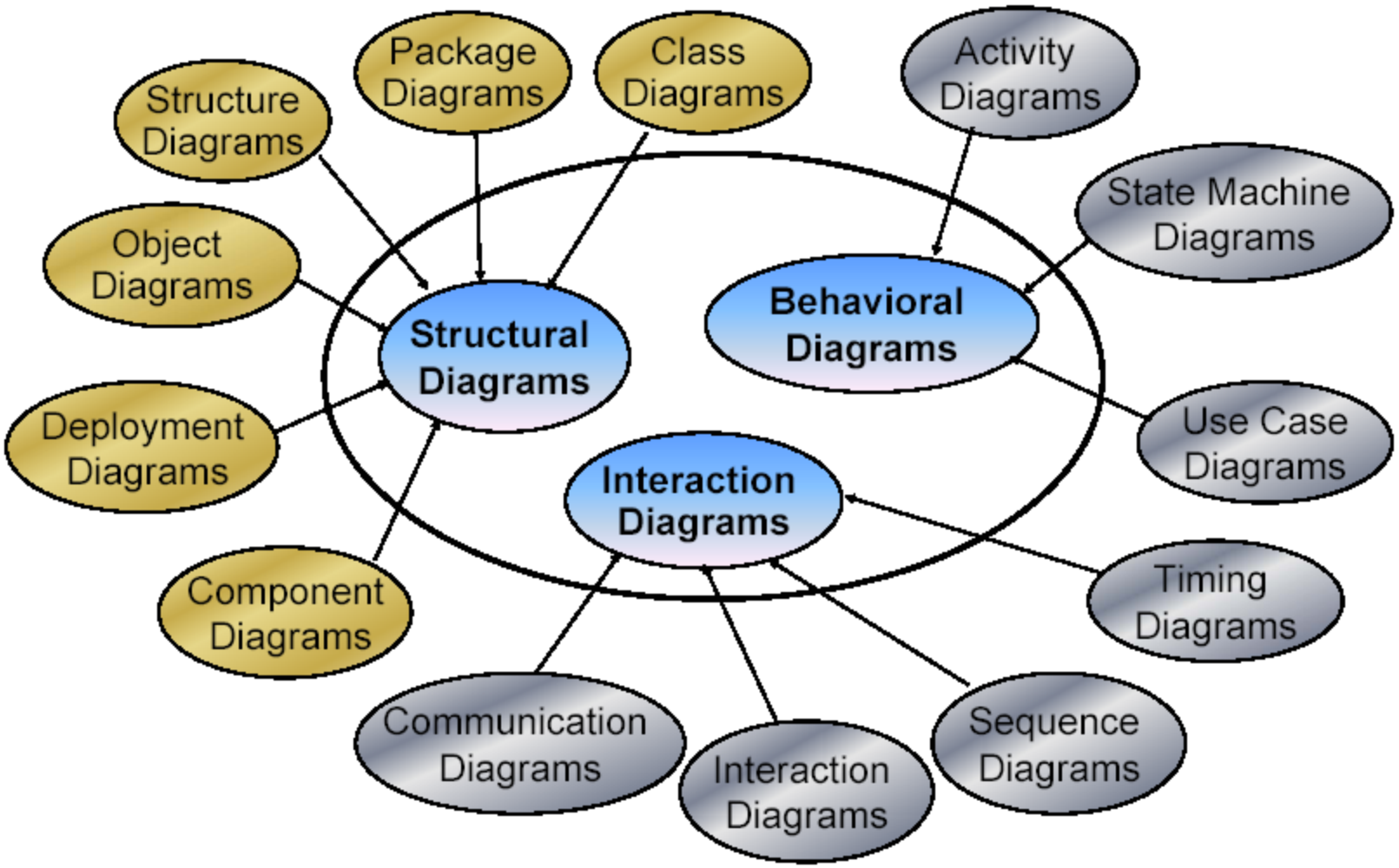
- diagrammes de cas d'utilisation
- diagrammes d'objets
- diagrammes de classes
- diagrammes de composants
- diagrammes de déploiement

Vues dynamiques du système

- diagrammes de collaboration
- diagrammes de séquence
- diagrammes d'états-transitions
- diagrammes d'activités



Les diagrammes de UML 2.0

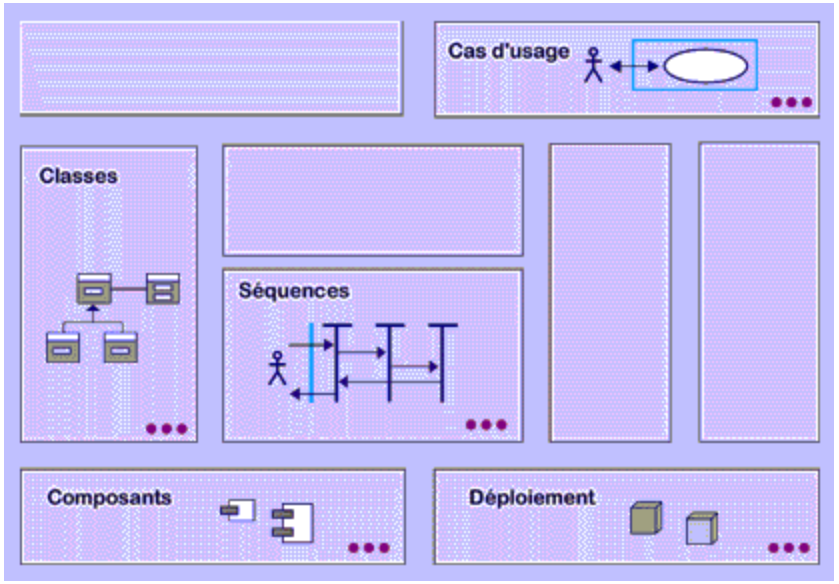


Processus de développement

Configuration minimum (UML "lite")

Parmi les neuf outils fournis par UML, cinq sont incontournables:

- Cas d'utilisation pour les fonctionnalités
- Classes et séquences pour l'analyse et la conception
- Composants et déploiement pour la construction.



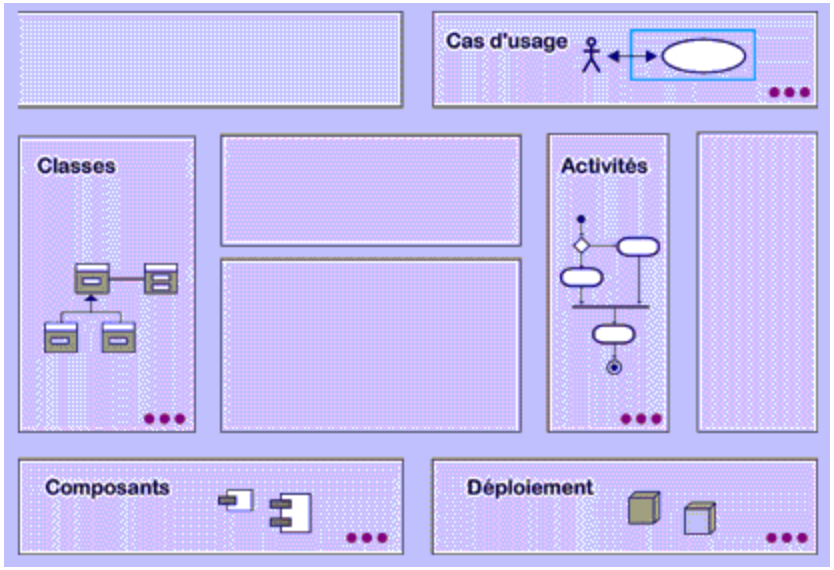
Processus de développement

Développement traditionnel

UML supporte parfaitement les méthodes de développement traditionnelles. Outre le diagramme de cas d'usage (pour les fonctionalités) et les diagrammes de composants et de déploiement (pour la construction), on utilisera:

- Le diagramme de classes, pour spécifier les types de données (pas d'opérations).

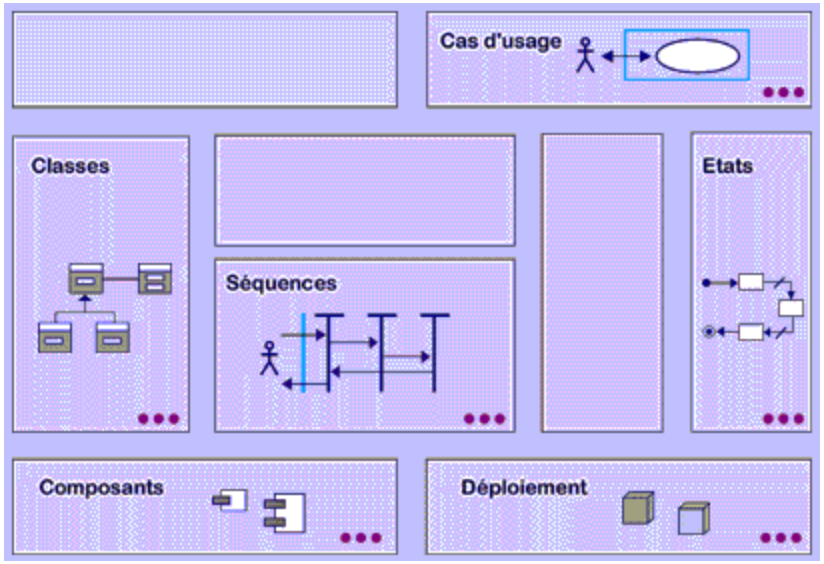
- Le diagramme d'activités, pour modéliser les traitements et la logique de contrôle.



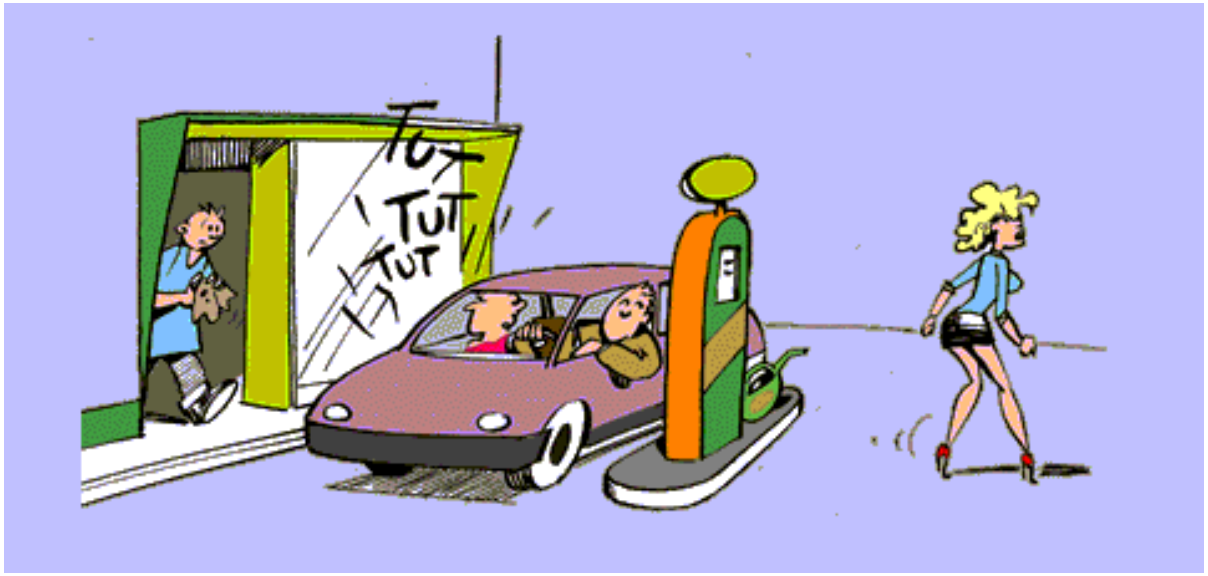
Processus de développement

Développement temps Réel

Aux cinq outils de base il faut ajouter le diagramme d'états, nécessaires pour modéliser la synchronisation des tâches. A noter que le diagramme d'états permet de modéliser les activités.



UML n'est qu'un support de communication !



Références

Cours UML, le langage de modélisation objet unifié

http://uml.free.fr

http://www.agilemodeling.com/essays/umlDiagrams.htm

A Practical Tutorial on UML

http://bdn.borland.com/article/0,1410,31863,00.html

UML-Principes de modélisation, Fannader et Leroux

www.uml.org