

Modélisation UML

**Mohamed DERKAOUI
Christopher LORENT**

**clorent@dawan.fr
01/2019**

Plus d'informations sur <http://www.dawan.fr>
Contactez notre service commercial au **0810.001.917** (prix d'un appel local)

DAWAN Paris, 11, rue Antoine Bourdelle, 75015 PARIS

DAWAN Nantes, 32 Boulevard Vincent Gâche, 44200 NANTES

DAWAN Lyon, Bâtiment de la banque Rhône Alpes, 2ème étage, montée B - 235, cours Lafayette, 69006 LYON

DAWAN Lille, 16, place du Générale de Gaulle, 6ème étage, 59800 LILLE

formation@dawan.fr

Objectifs

- Maîtriser les principes de l'UML : quand, où, pourquoi réaliser des diagrammes
- Connaître les conditions d'utilisation des différents diagrammes UML
- Être capable de lire et écrire les différents types de diagrammes UML

Bibliographie

UML 2 Infrastructure et Superstructure 2.4.1 – OMG 2011

UML 2 - Analyse et conception - Mise en oeuvre guidée avec études de cas, Joseph Gabay et David Gabay – Dunod, Juin 2008



UML 2 par la pratique - Etude de cas et exercices corrigés, Pascal Roques - Eyrolles Septembre 2009 (7ème édition)



UML 2 en action - De l'analyse des besoins à la conception, Pascal Roques et Franck Vallée - Eyrolles, Mars 2007



Plan

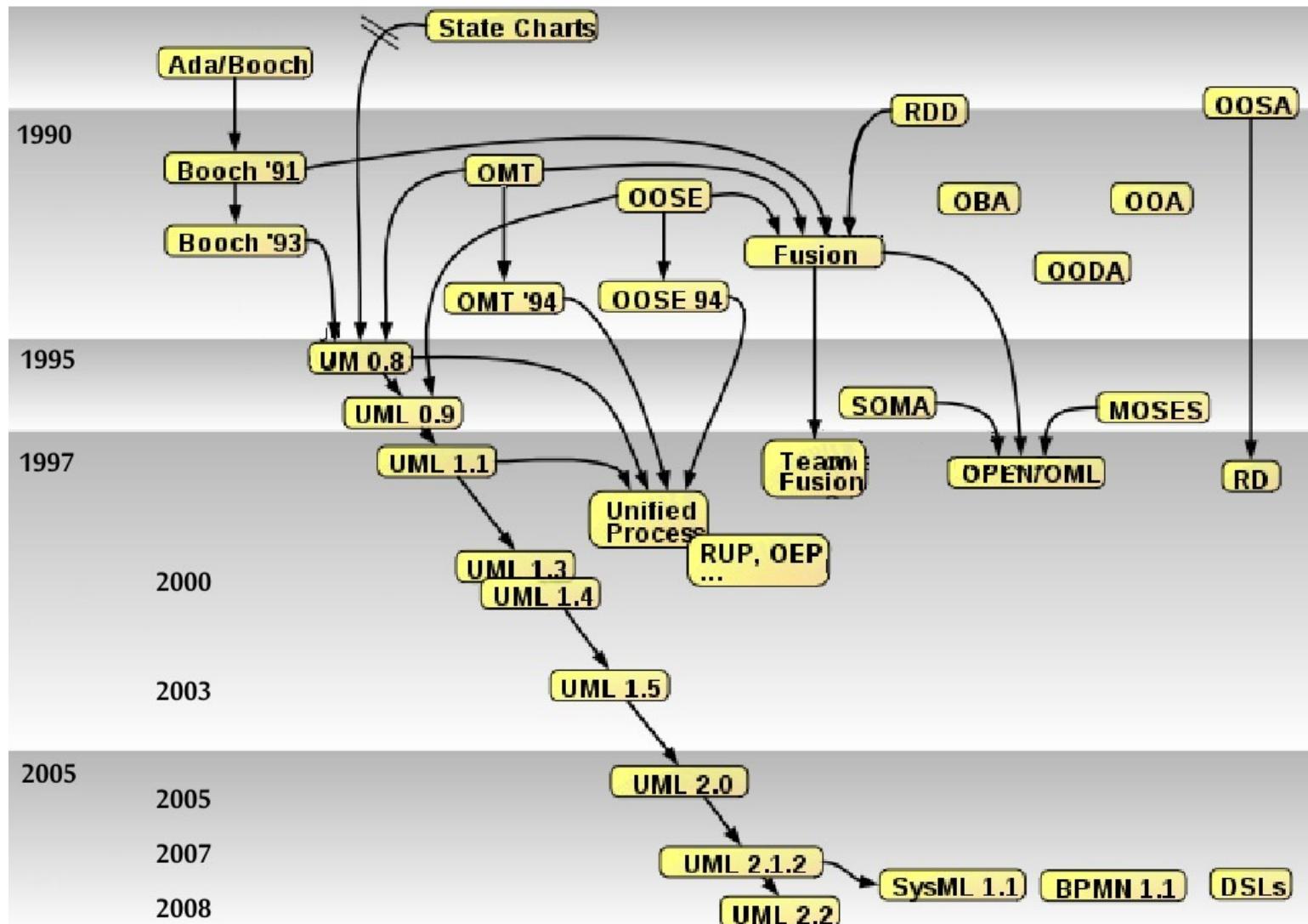
- Présentation
- Terminologie générale de l'UML
- Diagrammes structurels (statiques)
- Diagrammes comportementaux
- Diagrammes d'interactions (dynamiques)

Présentation

Introduction

- 1970 : Arrivée des méthodes d'analyse
- 1980 : Modélisation des données et modélisation des traitements (Merise, Axial, ...)
- 1990 : Arrivée des méthodes objet (OMT, OOD, OOSE...)
- 1997 : fusion des 3 méthodes (OMT, OOD et OOSE) :
Ces 3 méthodes ont donc été unifiée sous la bannière UML qui signifie **Unified Modeling Language**.
- 2005 : UML 2.0
- 2013 : UML 2.5 (bêta)

Historique



Source : wikipedia

Unified Modeling Language

UML = Langage pour la modélisation des classes, des objets, des interactions etc...

- UML n'est pas une méthode de conception contrairement au processus de développement logiciel RUP (*Rational Unified Process*), 2TUP, XP ...
- UML est défini dans des documents de référence issus de l'OMG (Object Management Group)
- L'OMG est un consortium international (IBM, Sun Microsystem, Apple, ...)

Méthodes associées à UML

- Les méthodes associées au projet informatique sont nombreuses, l'UML vient en support des diverses phases préconisées par la méthode
- Exemple avec cycle en V :
 - Analyse des besoins : diag. de cas d'utilisation
 - Spécifications : diag. de cas d'utilisation, de communication
 - Conception architecturale : diag. de composants, de structure composite, de séquence, global d'interaction, de paquetages
 - Conception détaillée : diag. de classes, d'objets, d'activité, d'états-transitions

UML et Merise

- MERISE : méthode d'analyse et de conception (prédominante en France)
- Inclus la gestion du projet, l'analyse, etc.
- Présente des modèles de représentation des structures de données et de leurs relations
- Ne présente pas la structure de l'application, son fonctionnement ni son déploiement
- UML et MERISE peuvent être utilisés ensemble, par les mêmes acteurs du projet, pour communiquer

UML dans la réalité

- Qui utilise ? Les acteurs du projet, généralement informatique (spécificateur, architecte, analyste, développeur, chef)
- Sous quelle forme ? Feuilles de papier ou outils (voire outils de génération UML <-> code)
- En quelle quantité ? De 1 diagramme de cas d'utilisation pour tout le projet à 1 diagramme / jour*homme
- Pourquoi pas ? Apprentissage, temps de réalisation, domaine très simple

Terminologie générale de l'UML

Les Vues

- Vue : ensemble d'observations du système modélisé (utilisant une « unité de langage »)
- Vue des cas d'utilisation : modèle vu par les acteurs du système (Qui fait Quoi ?)
- Vue logique : définition du système vu de l'intérieur. Comment satisfaire les besoins des acteurs ?
- Vue d'implémentation : définition des dépendances entre modules
- Vue des processus : vue temporelle et technique (tâches, contrôle, synchronisation)
- Vue de déploiement : architecture physique des éléments du système (Où ?)

Les modèles

- Contiennent trois catégories d'éléments (répartis en unités de langage) :
 - Classificateurs : ensemble d'objets
 - Objet : chose individuelle, avec un état et des relations avec d'autres objets
 - Evénements : ensemble d'occurrences possibles
 - Occurrence : Ce qui arrive (et a des conséquences sur le système)
 - Comportements : ensemble d'exécutions possibles
 - Exécution : déroulement d'un algorithme (en fonction de conditions précises)

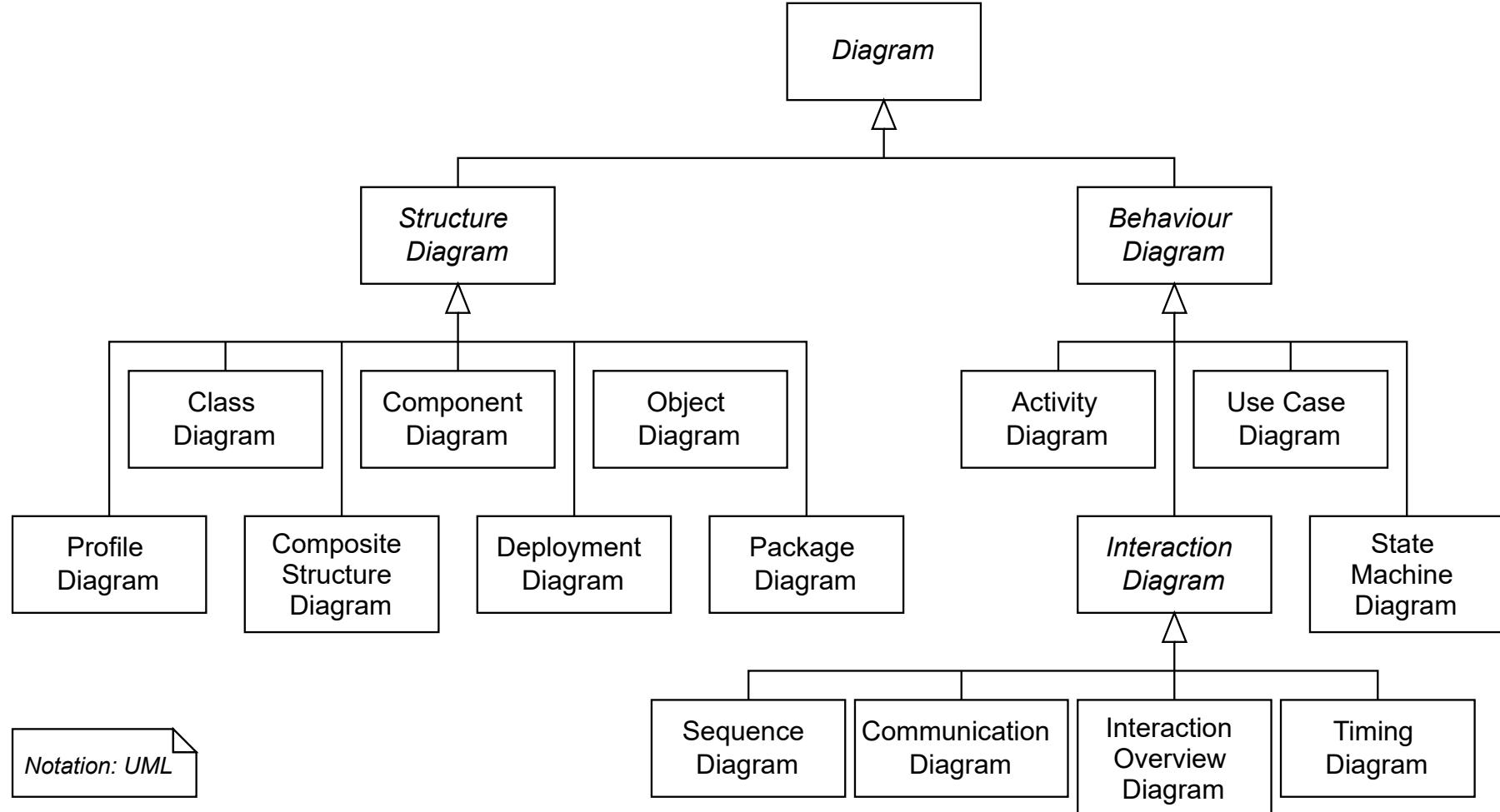
Les diagrammes UML

UML 2.3 comporte ainsi 14 types de diagrammes représentant autant de vues distinctes pour représenter des concepts particuliers du système d'information, grâce à des éléments.

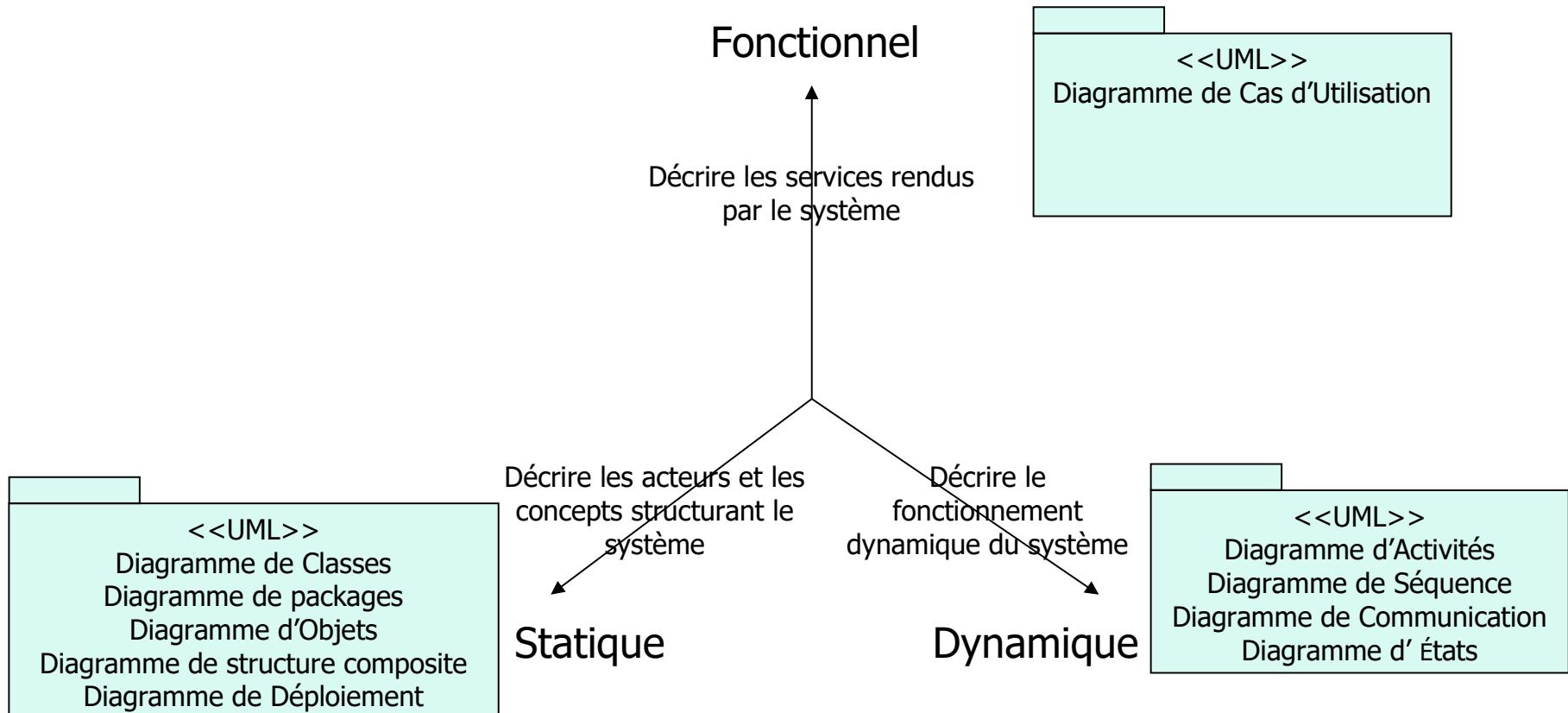
On peut les répartir en plusieurs groupes :

- Diagrammes structurels (statiques)
- Diagrammes comportementaux
- Diagrammes d'interactions (dynamiques)

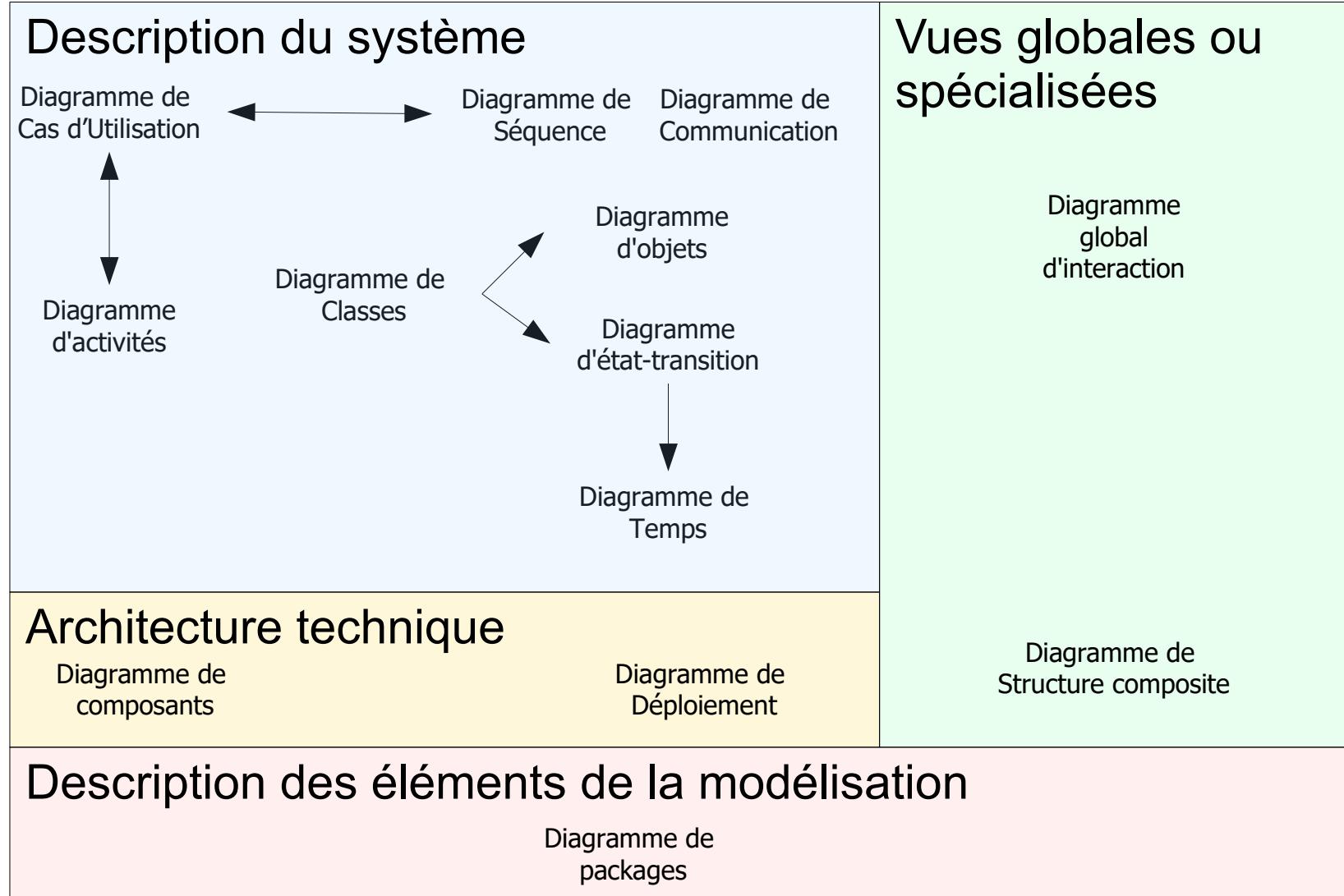
Les diagrammes UML



3 axes de modélisation



Regrouper les diagrammes



Le diagramme

Le diagramme est :

- Un ensemble d'éléments choisis et disposés par le créateur humain (ou une machine)
- Afin de communiquer des informations à un autre humain (ou une machine)
- En utilisant une (ou plusieurs) unités de langage

Le créateur fait des choix, le diagramme :

- Discute d'une vue particulière
- N'est pas nécessairement complet
- Peut être coupé en morceaux, jusqu'aux éléments eux-même
- Peut être associé à du texte, qui l'explique ou qu'il complète

Programmation Orientée Objet

Définition

L'orienté-objet = approche de résolution algorithmique de problèmes permettant de produire des programmes modulaires de qualité..

Objectifs :

- développer une partie d'un programme sans qu'il soit nécessaire de connaître les détails internes aux autres parties;
- Apporter des modifications locales à un module, sans que cela affecte le reste du programme;
- Réutiliser des fragments de code développés dans un cadre différent.

Qu'est ce qu'un objet ?

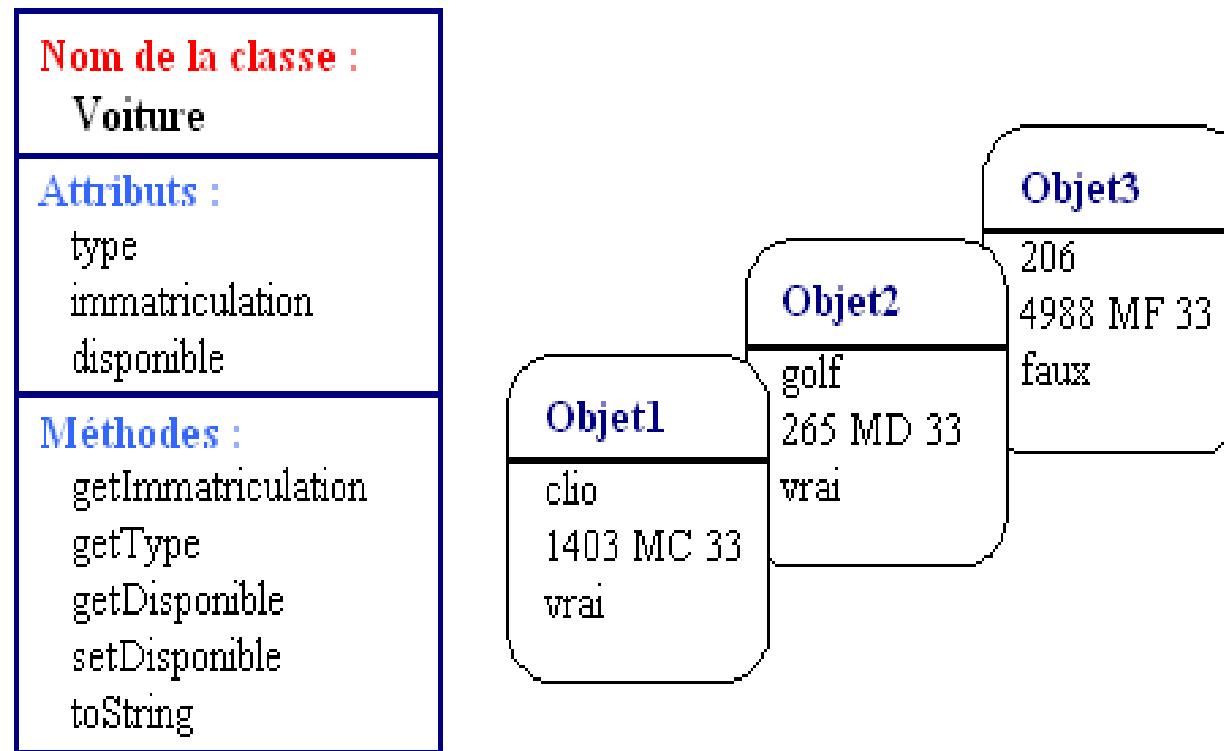
**Objet = élément identifiable du monde réel,
soit concret (voiture, stylo,...), soit
abstrait (entreprise, temps,...)**

Un objet est caractérisé par :

- son état (les données de l'objet)
- son comportement (opérations : ce qu'il sait faire)

Qu'est-ce qu'une Classe ?

- Une classe est un type de structure ayant des attributs et des opérations.
- On peut construire plusieurs **instances** d'une classe.



Packages ou Espaces de noms



Package = groupement de classes qui traitent un même problème pour former des « bibliothèques de classes ».

Une classe appartient à un package s'il existe une ligne au début renseignant cette option :

package nompackage; (*syntaxe java*)

Pour utiliser une classe depuis une autre classe (au choix) :

- Être dans le même package
- Préfixer par le package (à chaque utilisation)
- Au début du fichier, importer la classe, ou le package entier

import nompackage.*; (*syntaxe java*)

Attributs

Définissent l'état de l'objet :

- On les appelle également «variables de classes» (*ou propriété, ou champ*)
- La valeur d'un attribut est propre à chaque instance.

Certains attributs sont rattachés à la classe et non à ses instances. Sa valeur est alors partagée par l'ensemble des instances de la classe (par exemple une constante). On parle de variables de classes.

Opérations

Aussi appelées méthodes (*ou fonction*).

Bloc d'instructions définissant un comportement d'une instance.

- déclarées à l'intérieur d'une classe.
- peuvent être surchargées dans la plupart des langages (même nom, différents paramètres,...)

Comme pour les attributs, certaines opérations sont rattachées à la classe. Il s'agit d'un comportement global ou un service particulier (créer une instance, ...). On parle de méthodes de classe.

Agrégation et Accessibilité

- **Agrégation** : associer un/plusieurs objet avec un autre
- **Composition** : aggrégation avec une contrainte temporelle
- **Accessibilité** : utilisation de facteurs de visibilité
 - public**:
 - Accessible par toutes les classes
 - protected**:
 - Accessible par toutes les sous-classes et les classes du même package
 - "nothing" ou default**:
 - Accessible seulement par les classes du même package.
 - private**:
 - Accessible seulement dans la classe elle-même

Encapsulation

Encapsulation =

- Regroupement de code et de données.
- Masquage d'information par l'utilisation d'accesseurs (**getters et les setters**) afin d'ajouter du contrôle .

L'encapsulation permet de restreindre les accès aux membres d'un objet, obligeant ainsi l'utilisation des membres exposés.

Héritage

Mécanisme mis en place lorsque plusieurs classes partagent des attributs et/ou des méthodes.

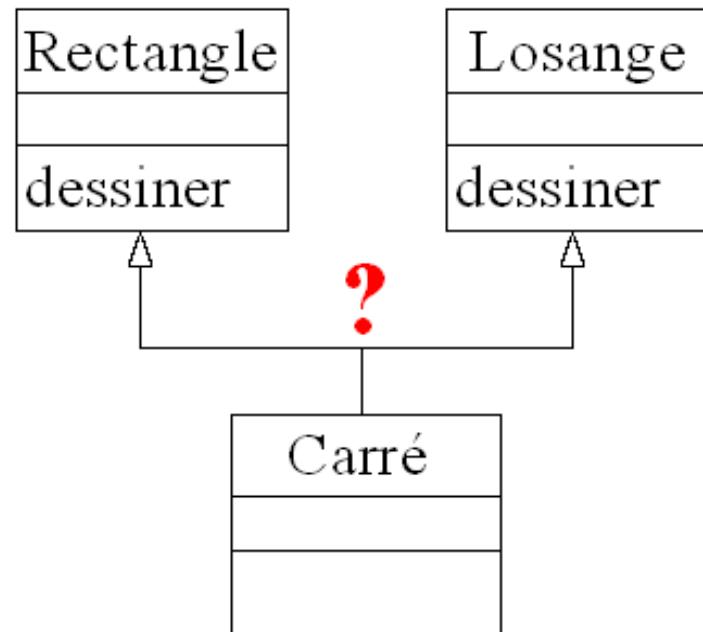
- Généralisation : extraction d'une super-classe à partir de plusieurs classes.
- Spécialisation : création d'une sous-classes à partir d'une super-classe.

Une sous-classe dispose de toutes les méthodes et attributs de la classe dont elle hérite.

Héritage multiple

L'héritage multiple permet à une classe d'hériter simultanément de plusieurs autres classes.

Problème :



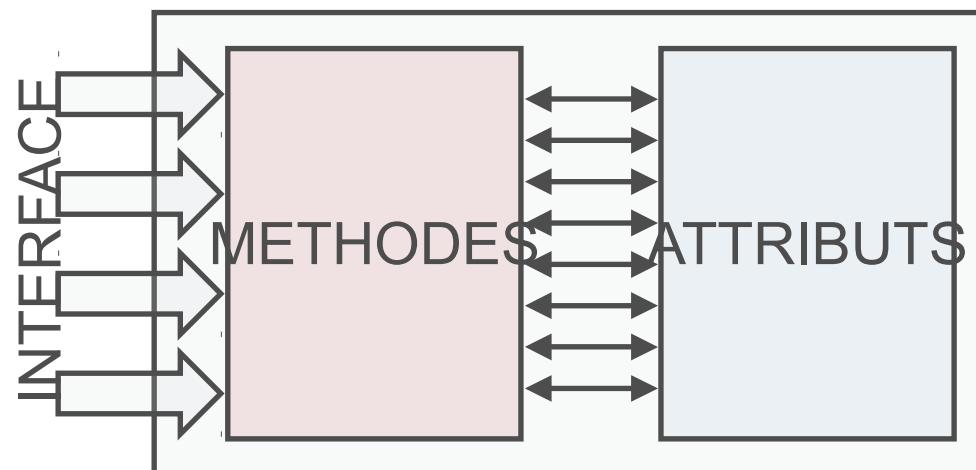
Supporté par peu de langages (C++, Python, Eiffel).

L'héritage multiple peut être partiellement comblé par une approche d'héritage en cascade des classes.

Classe Abstraite/Interface

Classe abstraite : classe qui ne peut être instanciée. Elle définit un squelette pour les classes qui en hériteront.

Interface : vue externe d'une classe. Elle contient la liste des méthodes accessibles par les autres classes. Elle ne contient (en principe) pas d'attributs.



Polymorphisme

Le polymorphisme est la propriété d'une entité de pouvoir se présenter sous diverses formes. Ce mécanisme permet de faire collaborer des objets entre eux sans que ces derniers aient déclarés leur type exact.

Exemples :

- On peut avoir une voiture prioritaire avec le type Voiture
- On peut créer un tableau de Voitures et placer à l'intérieur des objets de type Voiture et d'autres de type VoiturePrioritaire

Avantages de la POO

- Abstraction du monde réel
- Forte évolutivité
- Faible couplage entre les composants
- Réutilisation
- Stabilité et robustesse
- Simplicité du modèle : 5 fondements :
 - ◆ Objet
 - ◆ Classe et instantiation
 - ◆ Message et encapsulation
 - ◆ Héritage
 - ◆ Polymorphisme

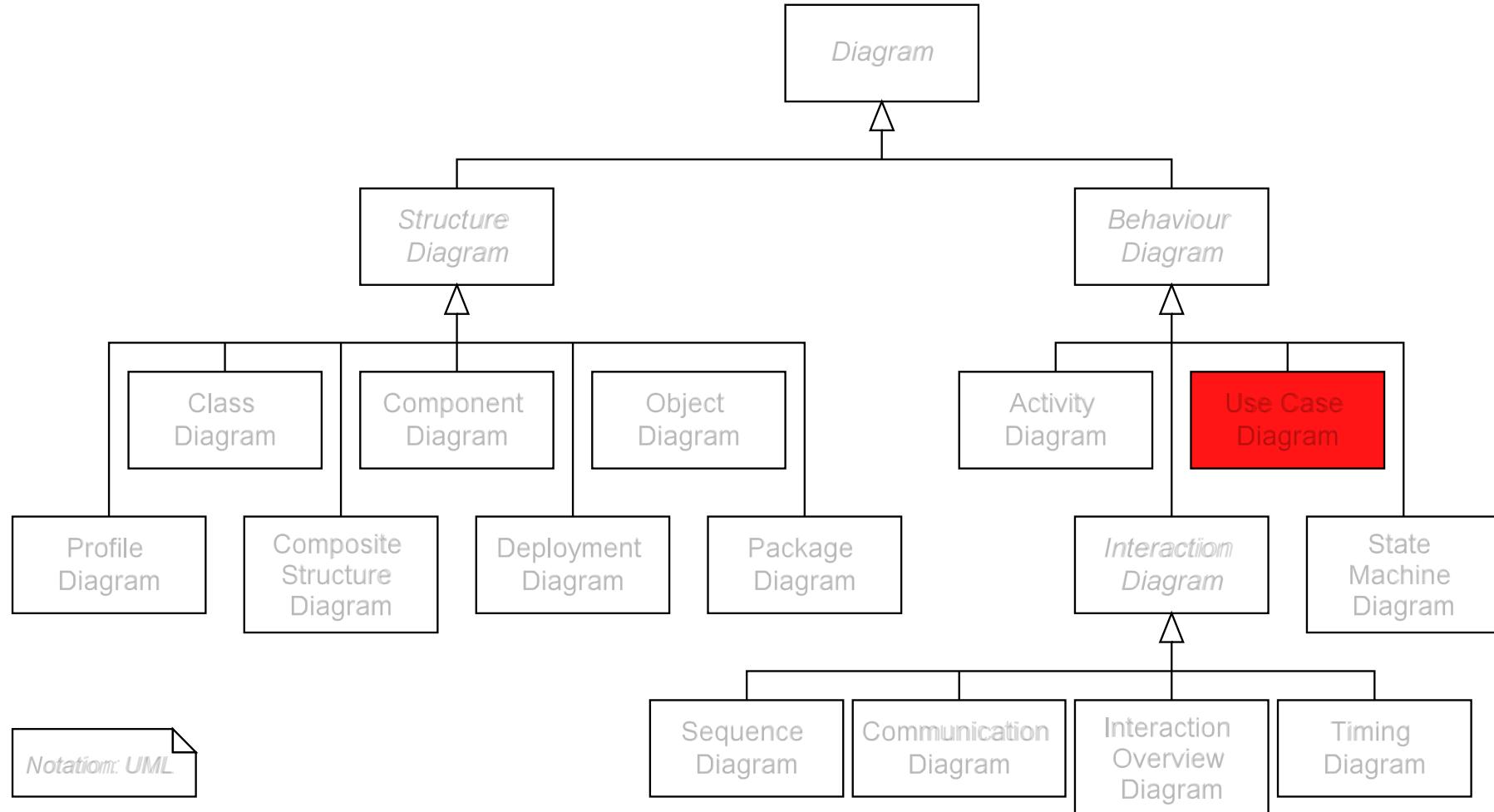
Avantages de la POO (2)

- Gestion de la complexité : il faut diviser pour comprendre et réunir pour construire
- L'approche objet repose sur une démarche systémique
 - ◆ Le système est un tout organisé dont les éléments sont solidaires et définis les uns par rapport aux autres
 - ◆ Met en valeur ce que le système fait mais surtout de quoi il est fait

Recueil et analyse des besoins

Diagramme des cas d'utilisation

Les diagrammes UML

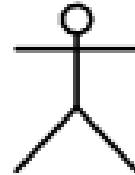


Description

- Modélisation des interactions entre les utilisateurs (acteurs) et le système.
- Définition des comportements et des contraintes (scripts ou scénarios)
- Utilisation :
 - * Détermination des besoins
 - * Communication avec les clients
 - * Génération de cas de tests

Eléments du diagramme

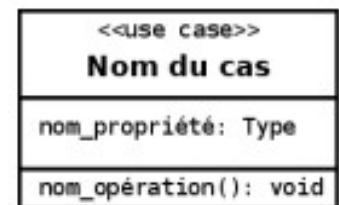
- Acteur ou



Actor



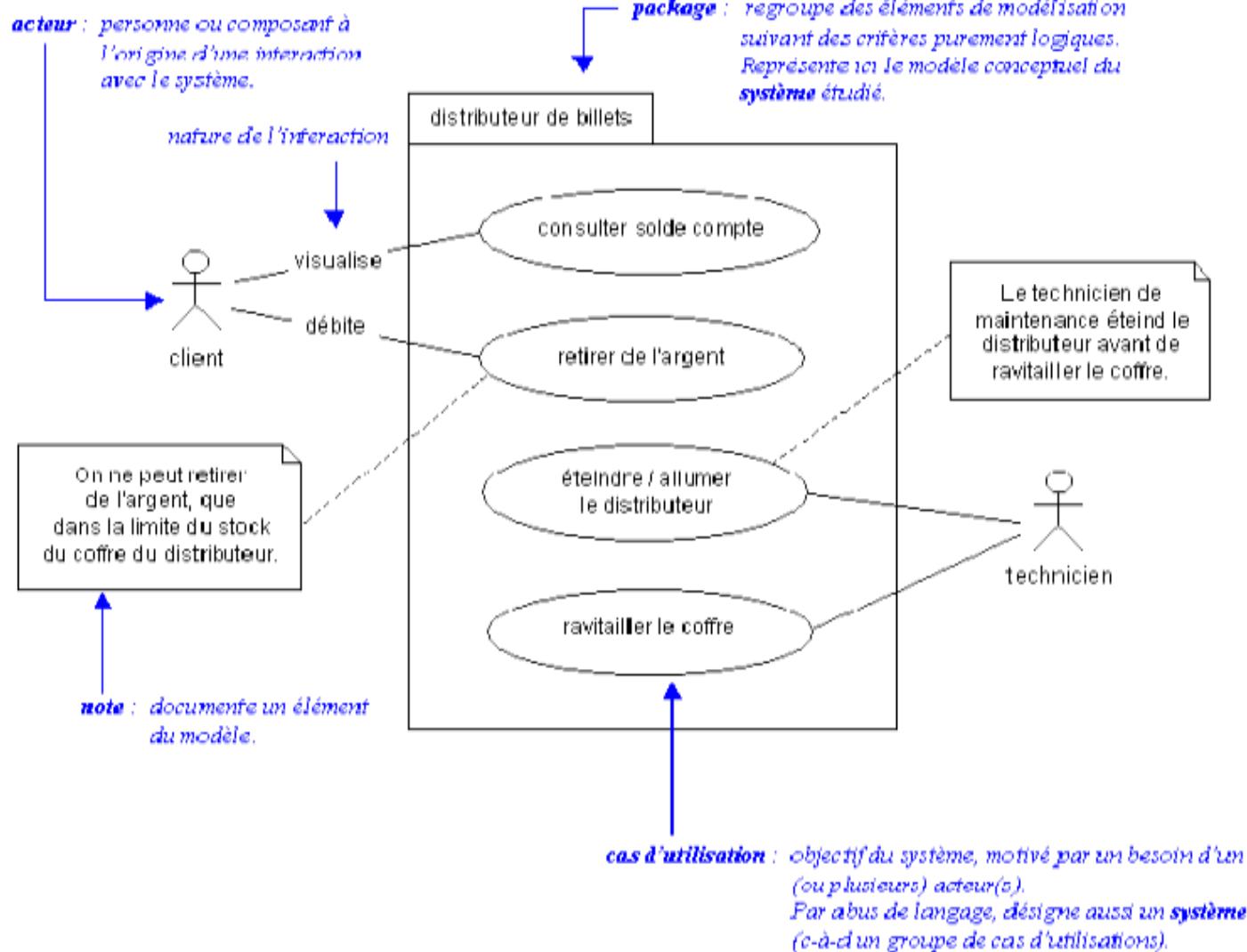
- Cas d'utilisation ou



- Association

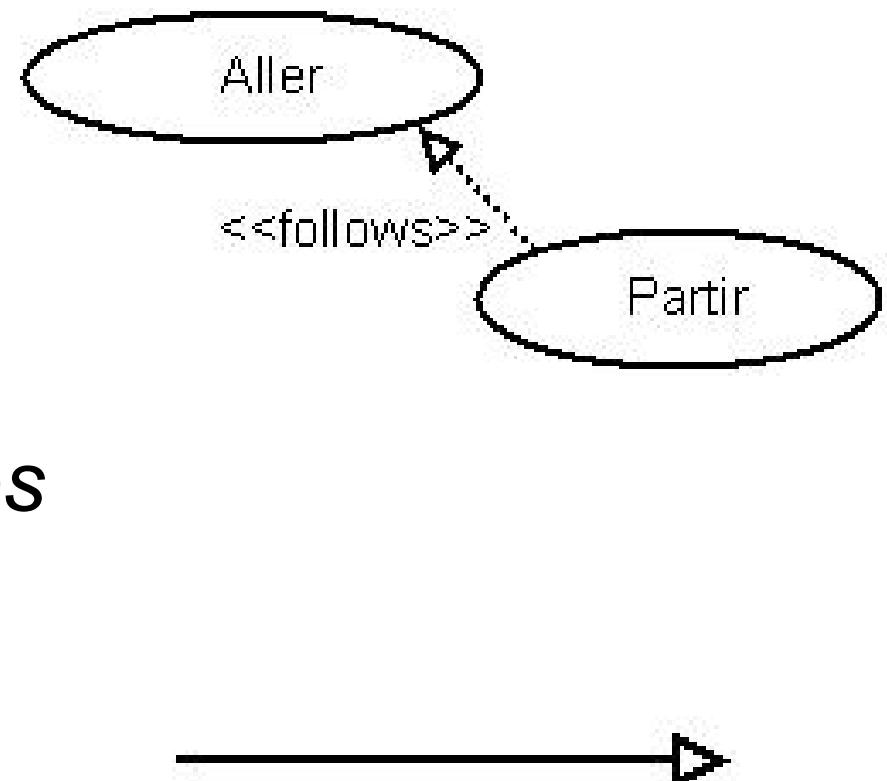
<<stéréotype>>

Exemple

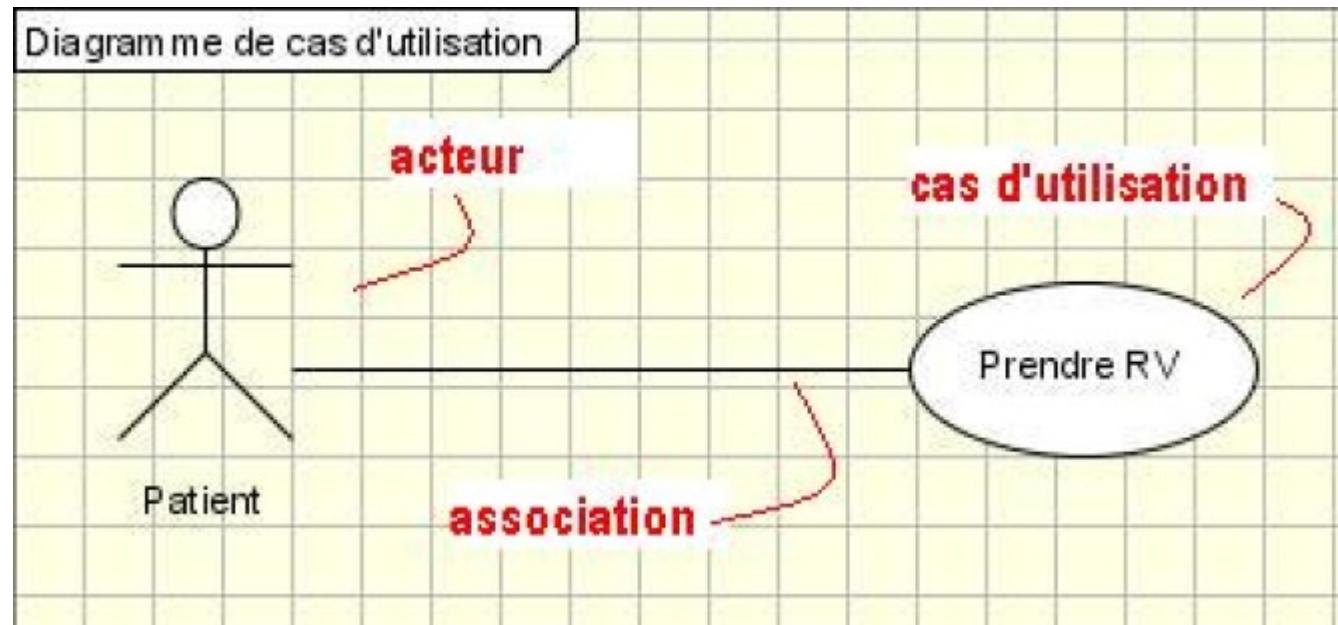


Relations

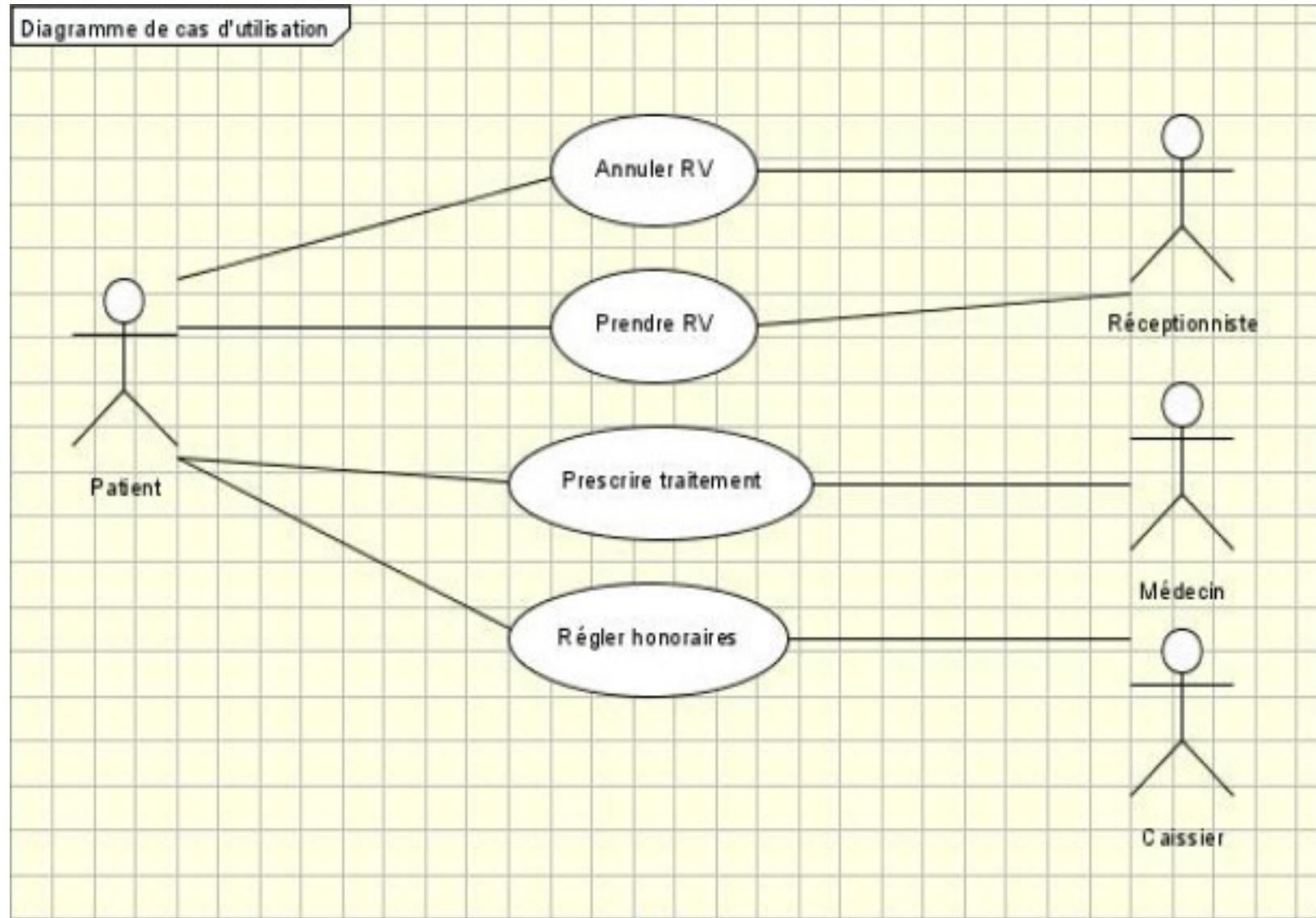
- Relations entre différents cas d'utilisation
- Habituellement écrit en anglais, entre « »
- Principales : *includes*, *extends*, *follows*, *requires*
- Utilisable aussi : généralisation (flèche)



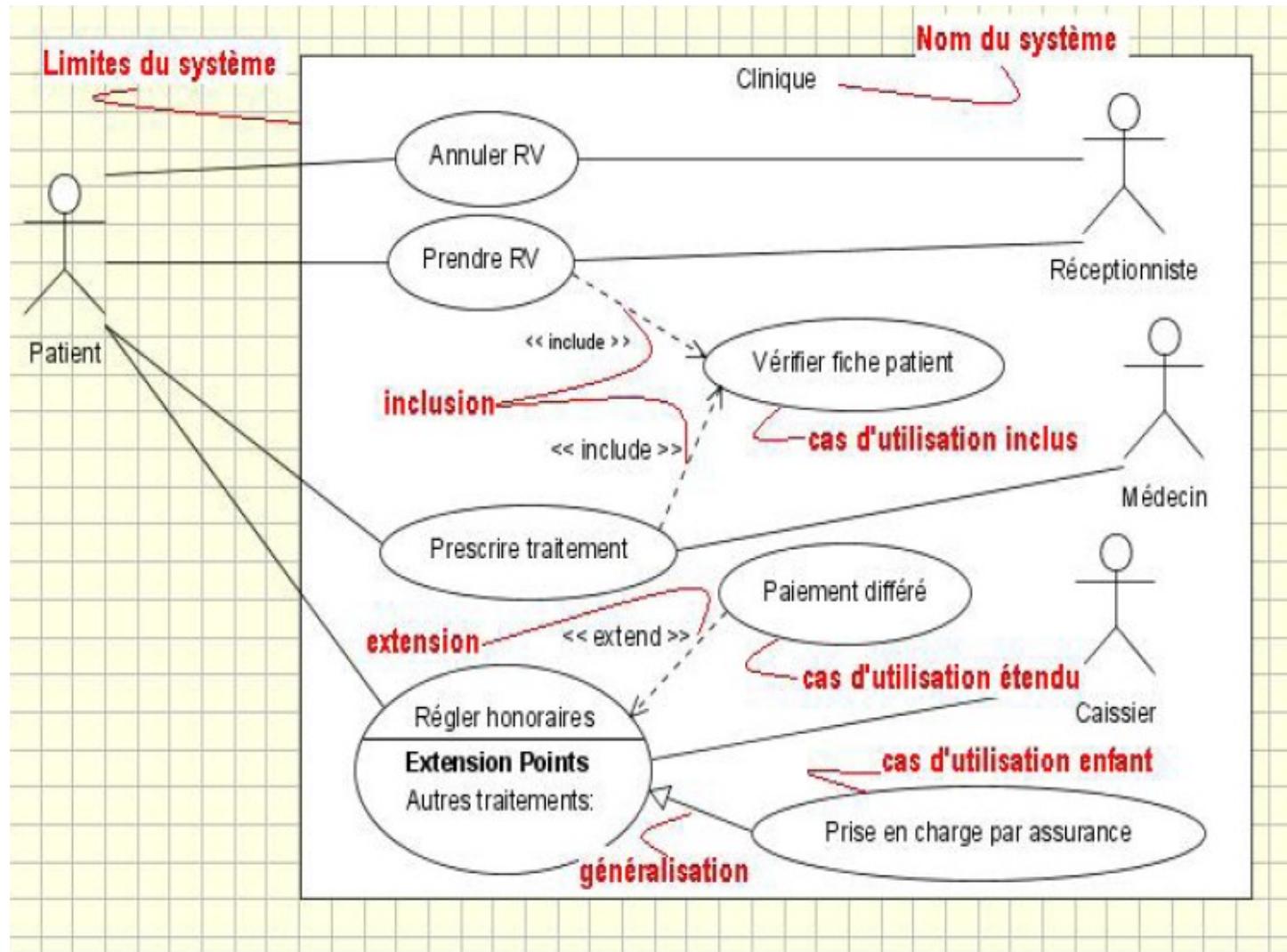
Exemple de relation



Exemple



Exemple

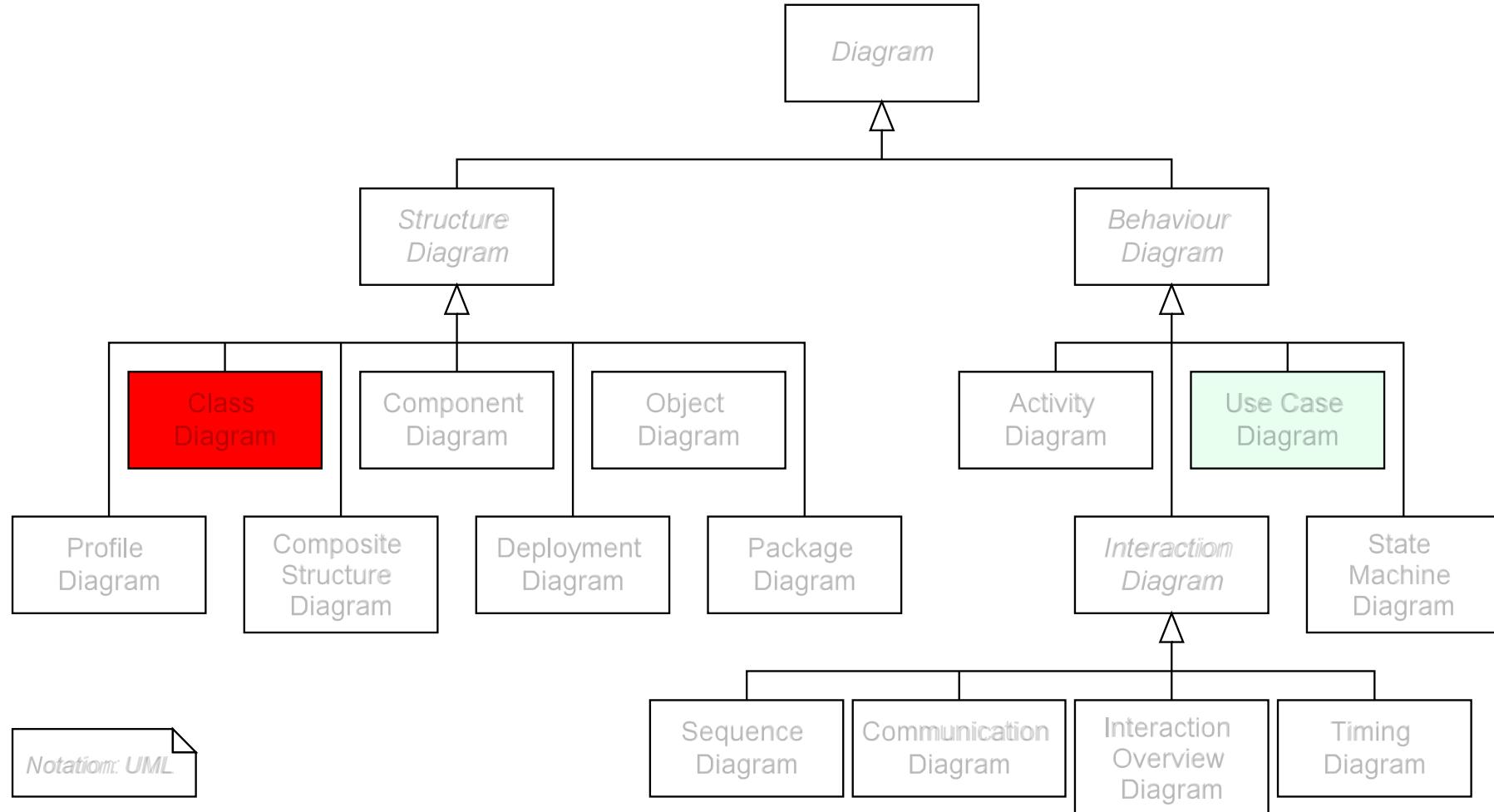


Diagrammes Structurels (statiques)

- Diagramme de classes
- Diagramme d'objets
- Diagramme de composants
- Diagramme de déploiement
- Diagramme de paquetage
- Diagramme de structure composite
(ou diagramme d'architecture)

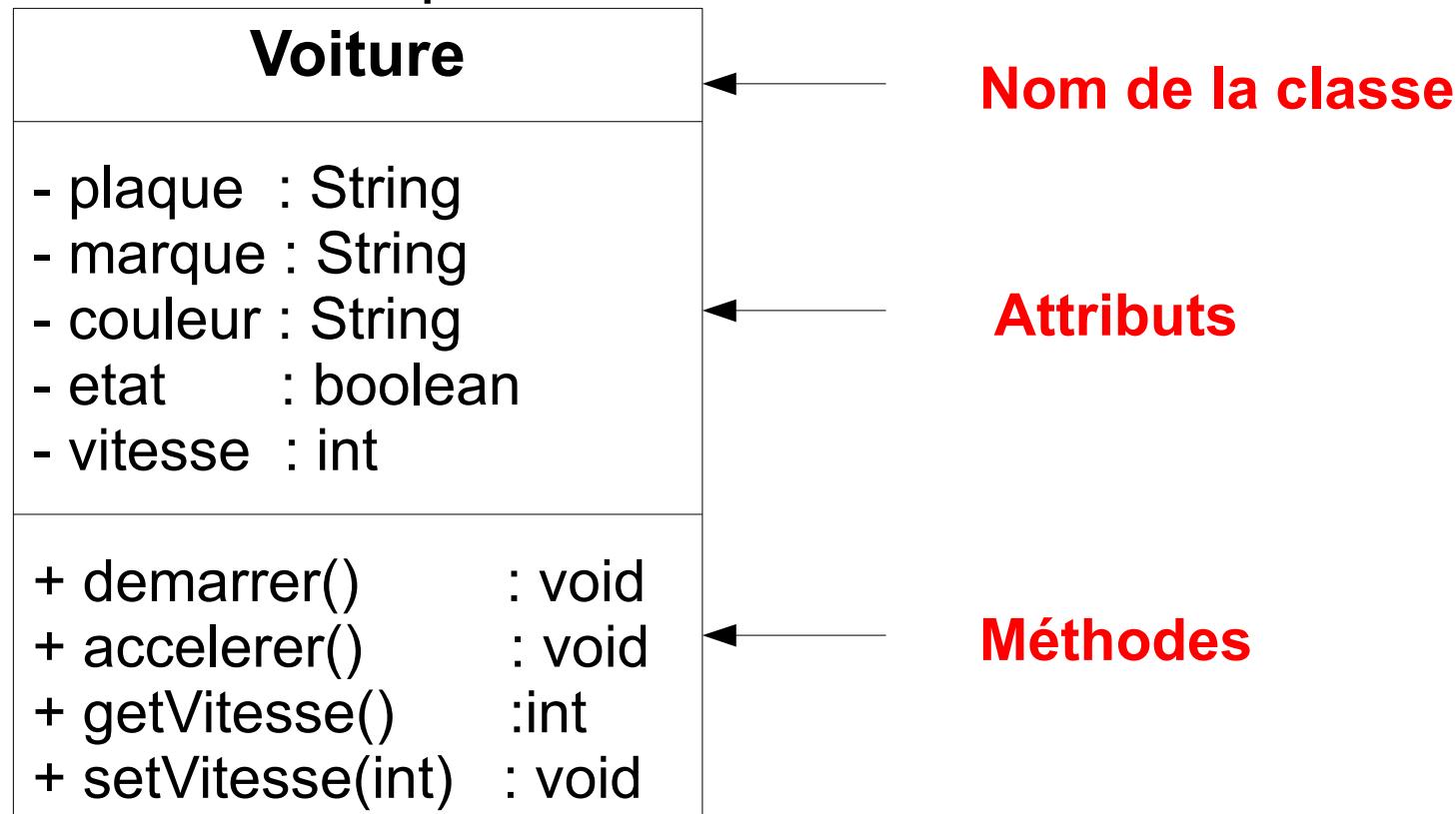
Diagramme de classes

Les diagrammes UML



Description

- Représentation statique d'une classe



Les attributs ou les méthodes peuvent être précédés par un opérateur (+, #, -) pour indiquer le niveau de visibilité
+ : public, # : protected, - : private

Syntaxe

- Nom de la classe centré en gras :
[<espace de nom>]<nom> [abstract, auteur, date...]
(*nom en italique* si la classe est abstraite)
- Interface déclarée avec le mot clé « Interface » suivi du nom en italique.

Syntaxe (suite)

- Déclaration d'un attribut :

[<espace de nom>]<facteurVisibilité><nom>
:<type >

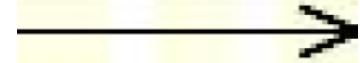
- Déclaration d'une méthode :

[<espace de nom>]<facteurVisibilité><nom>
(<paramètres typés>):<type de retour>

- Membres statiques (variables ou méthodes de classes) soulignés ou précédés par un \$.

Relations entre classes

Association

- Représente les relations qui existent entre objets de différentes classes.
- Relation entre deux classes (association binaire) ou plus (association n-aire).
- Représentation :
 - * Flèche pour une association orientée 
 - * Ligne droite pour une association non orientée 
- Peut posséder un nom et une cardinalité (multiplicité)

1 Un et un seul

0..1 Zéro ou un

N N (entier naturel)

M..N De M à N (entiers naturels)

* De zéro à plusieurs 0..* De zéro à plusieurs

1..* D'un à plusieurs

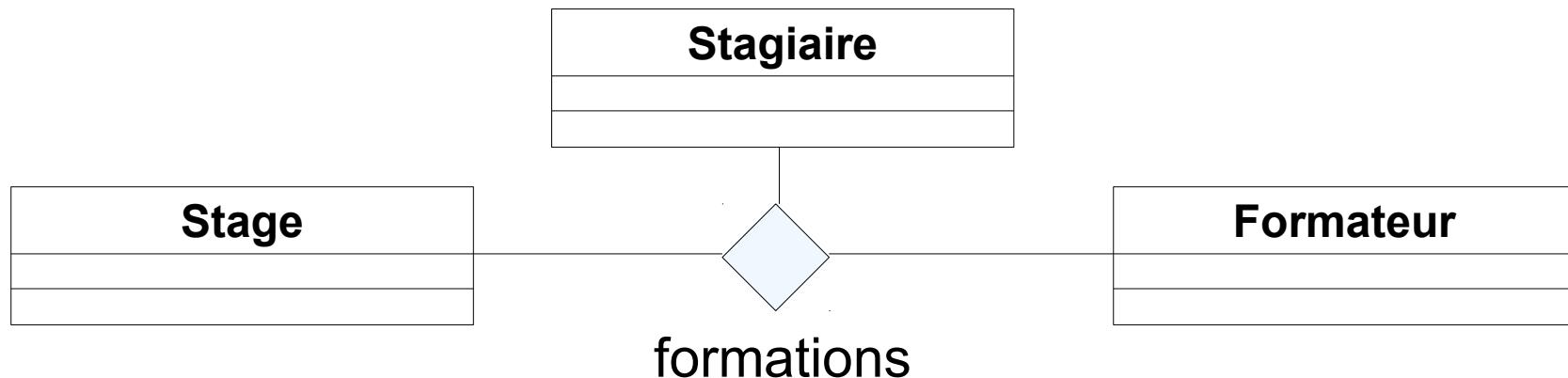
Relations entre classes

Association (2)

- Binaire :

Salarie	employé	travailler pour >	employeur	Entreprise
nom : String prenom : String	*		1	raisonSociale : String ... + chiffreAffaires() :float
calculSalaire() :float				

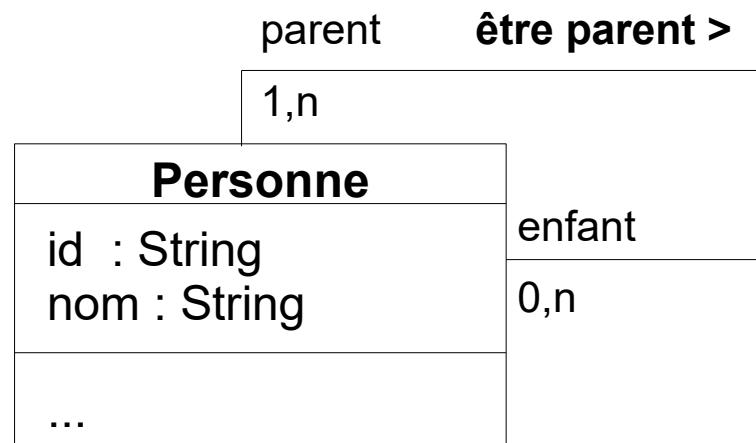
- N-aire :



Relations entre classes

Association (3)

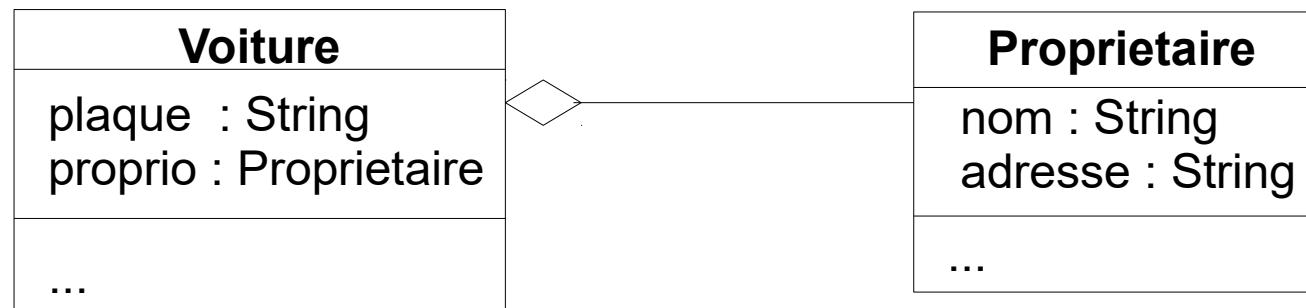
- Cas particulier :



Relations entre classes

Agrégation

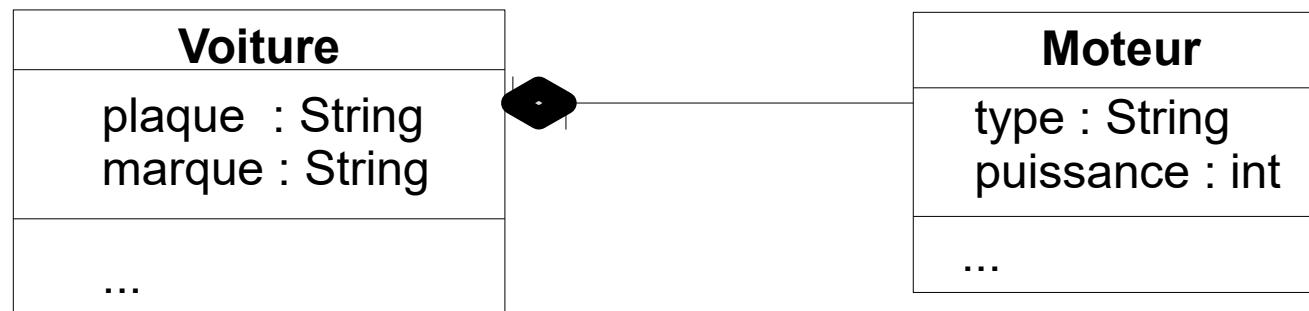
- Représente une relation d'inclusion structurelle ou comportementale d'un élément dans un autre
- Graphiquement représentée par une ligne avec un losange vide du côté de l'agrégat



Relations entre classes

Composition

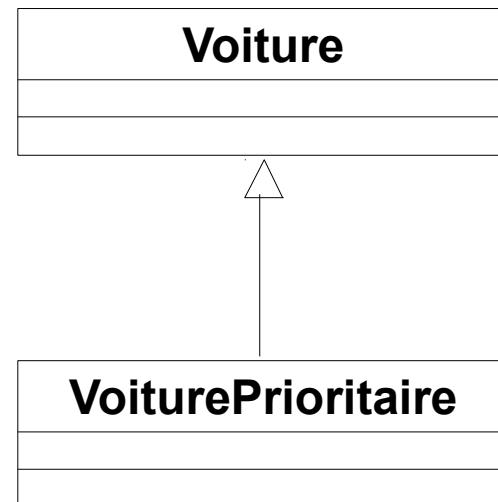
- Représente une "agrégation forte".
- Impose des contraintes : un seul objet peut faire partie d'un composite, et ce dernier doit gérer toutes ses parties.
- Graphiquement représentée par une ligne avec un losange plein



Relations entre classes

Généralisation et Héritage

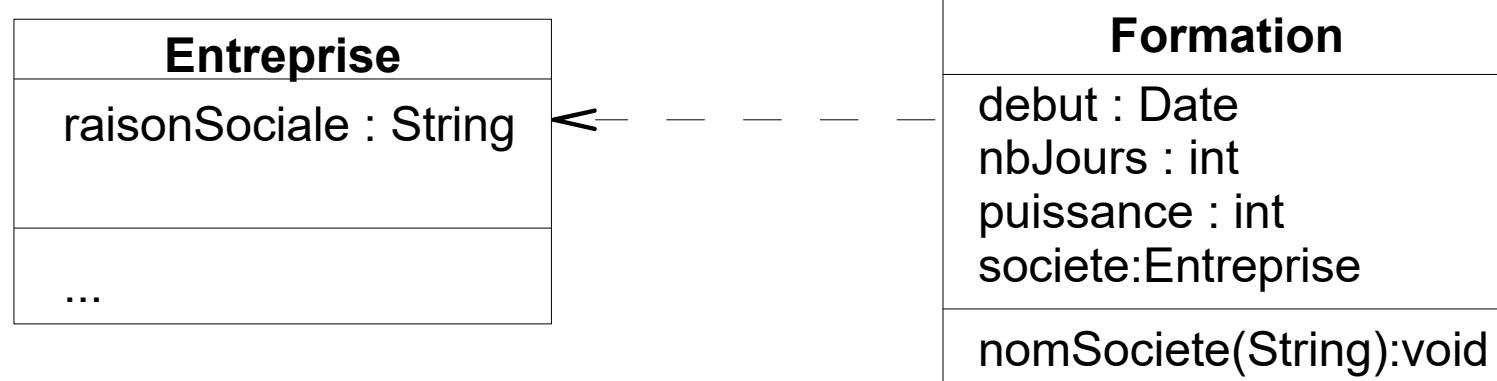
- Factorisation des attributs et méthodes d'un ensemble de classes pour faciliter la maintenance de code
- Création de classes spécialisées
- Représentée graphiquement par un trait avec un triangle à l'extrémité du côté de la classe mère



Relations entre classes

Dépendance

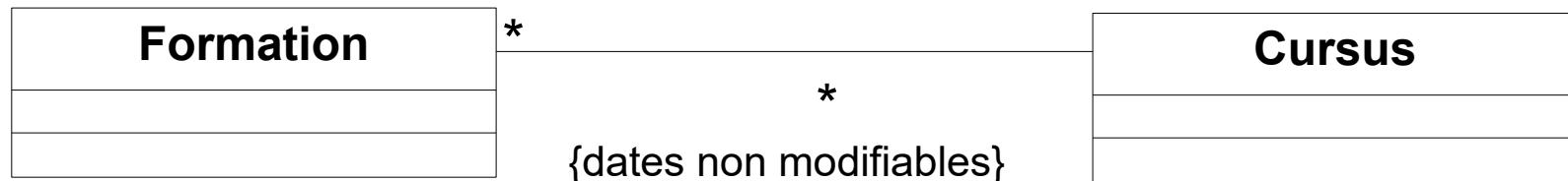
- S'il existe une relation de dépendance entre deux classes, un changement dans l'une d'elles entraînera des changements sur l'autre
- Représentée graphiquement par une flèche avec une ligne en pointillé



Relations entre classes

Contrainte

- Représente une condition qui doit être respectée
- Notée entre accolades.



Relations entre classes

Réalisation

- Implémentation d'une interface
- Représentée par une ligne pointillée et un triangle pointant sur l'interface.
- L'interface peut être représentée par un cercle.

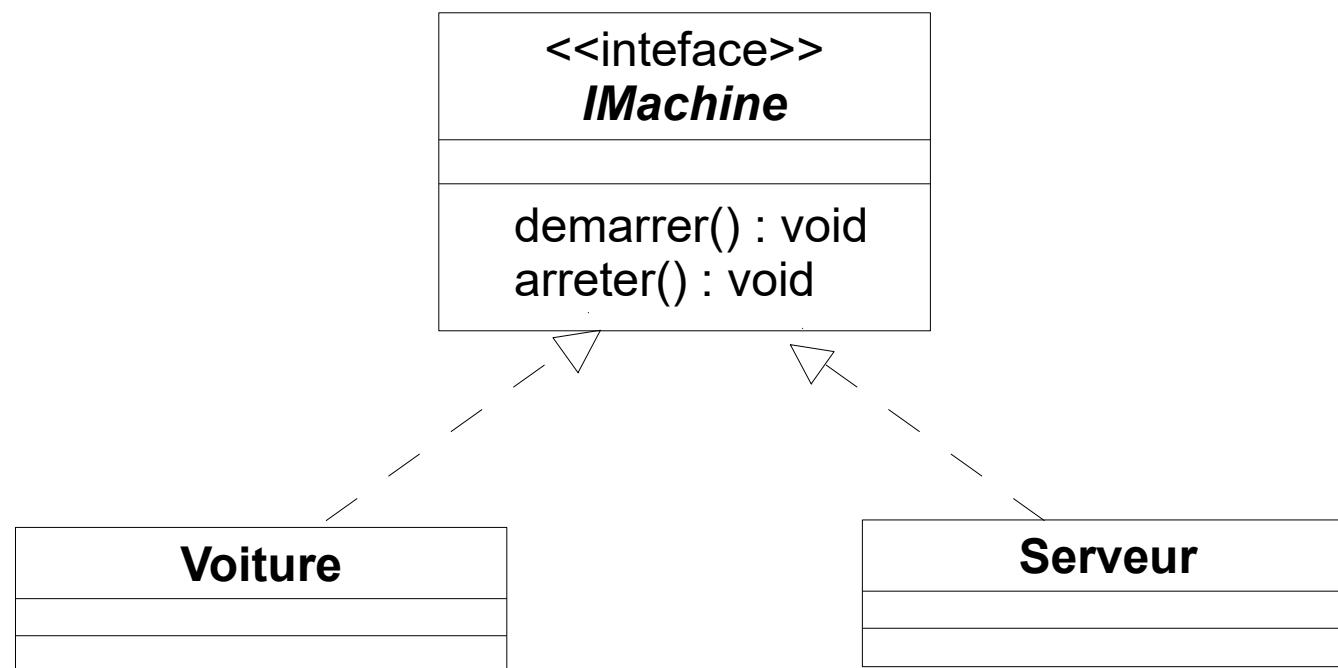
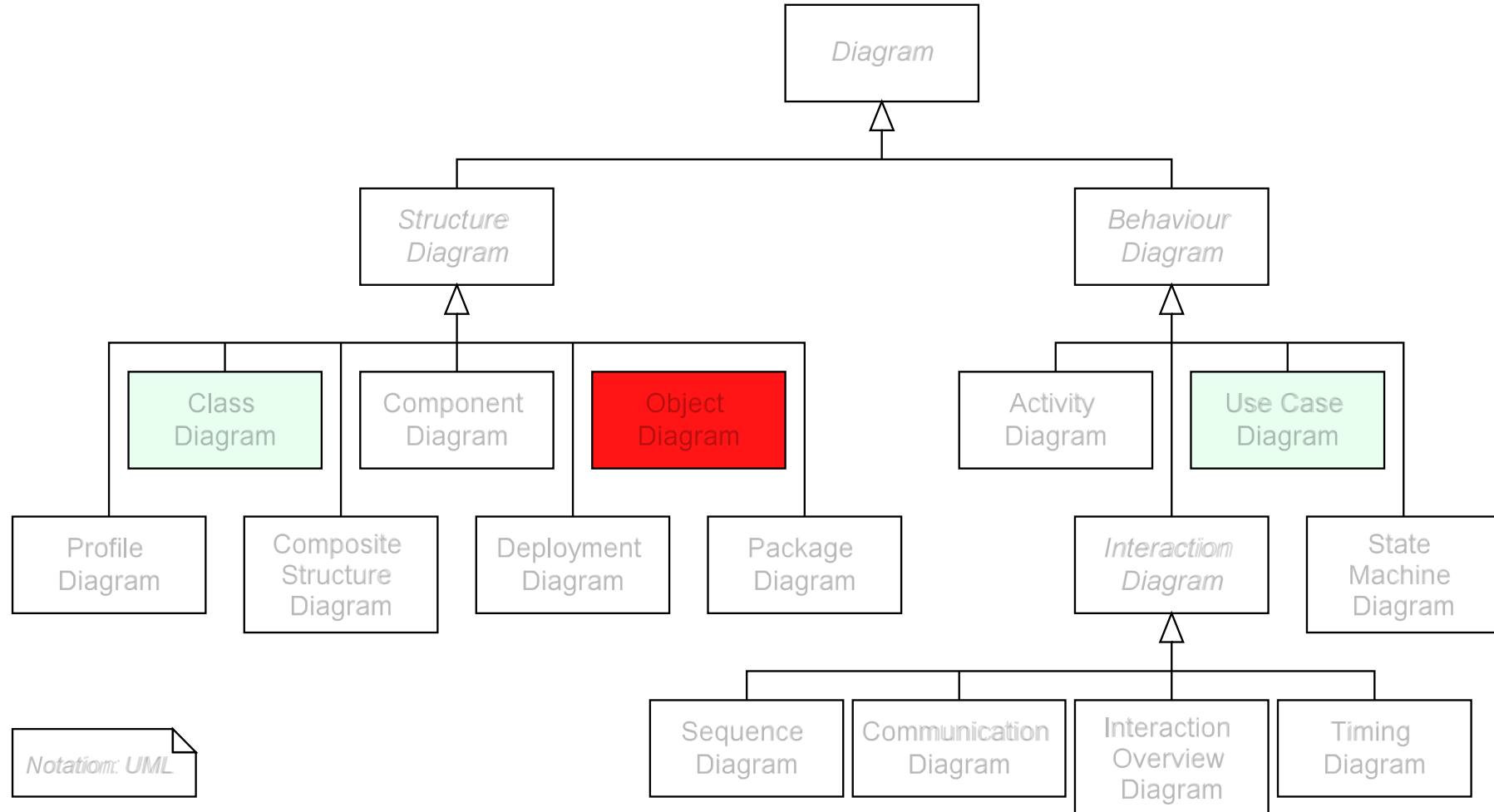


Diagramme d'objets

Les diagrammes UML



Objectifs

- Représenter les instances des classes
- Donner une vision d'une relation qui peut exister à un instant donné entre deux instances d'une même classe qui ne jouent pas le même rôle.
- Offrir une vue du système à un instant donné
- Exprimer une exception en modélisant des cas particuliers.

Représentation

- Nom souligné qui peut être suivi du type
<nom> : <type>, ou **:<type>**

<u>V1 : Voiture</u>
plaque : w75
marque : Clio
couleur : blanche
état : false
vitesse : 0
....

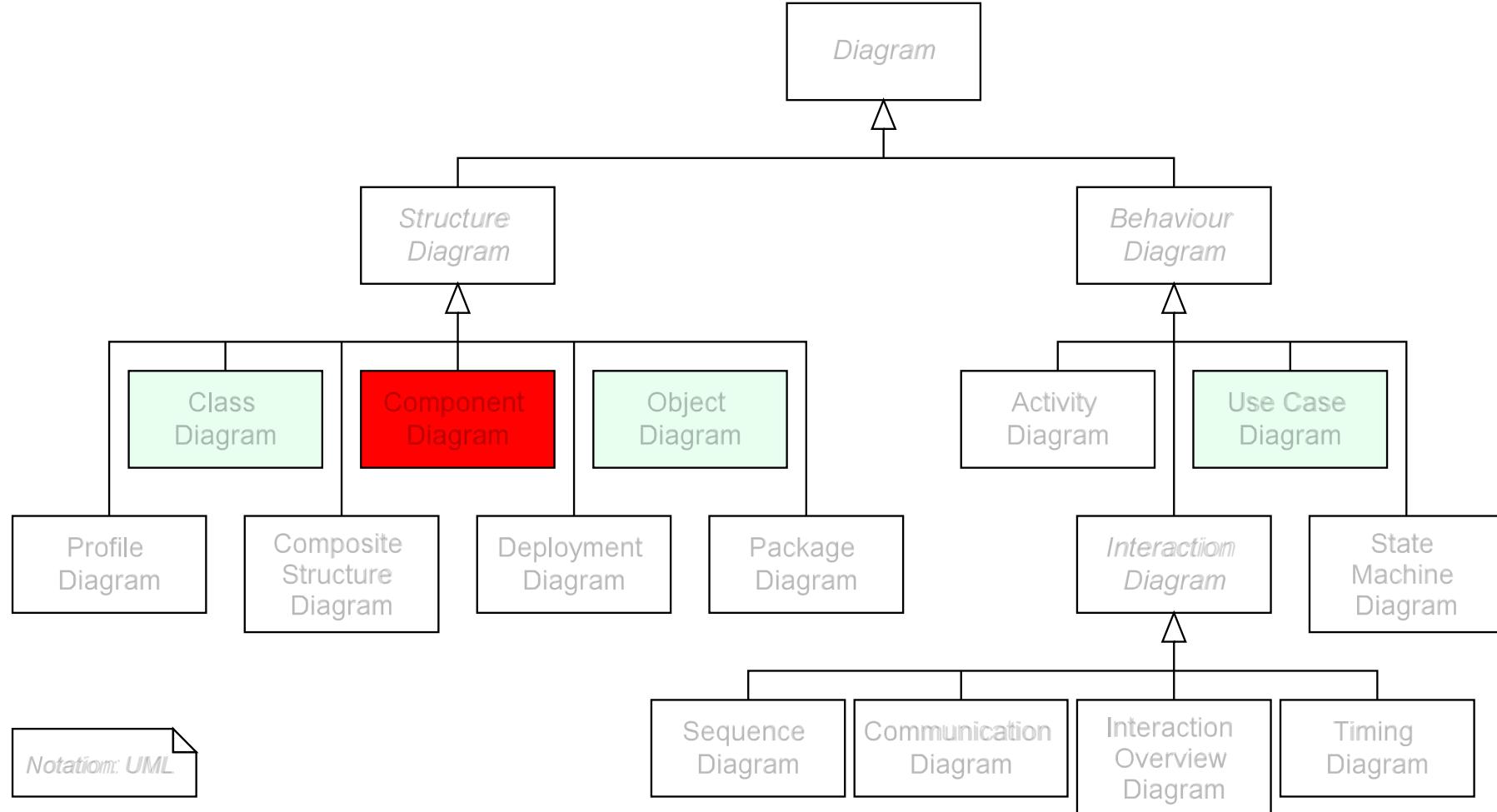
- Instantiation :

L'instanciation est représentée par une flèche en pointillée accompagnée de **<<instanceof>>**.



Diagramme de composants

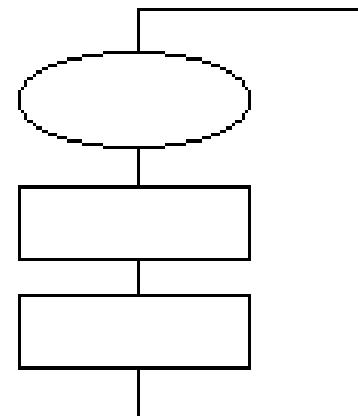
Les diagrammes UML



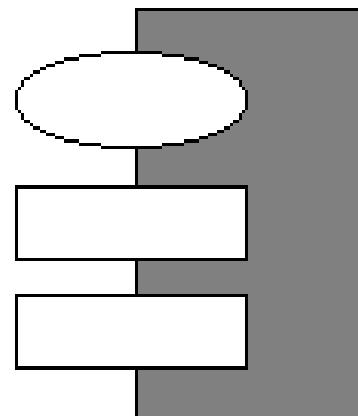
Description

- Représentation des composants du système d'un point de vue physique (fichiers, bibliothèques, bases de données...)
- Un composant est représenté de manière graphique par un rectangle comportant 2 petits rectangles et une ellipse.

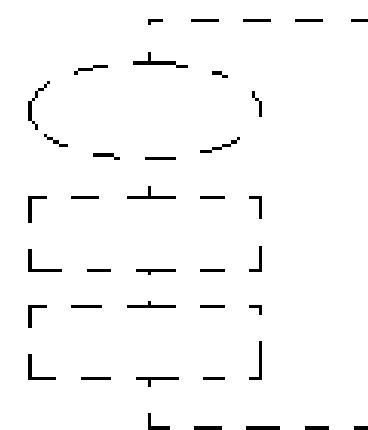
Spécification



Corps

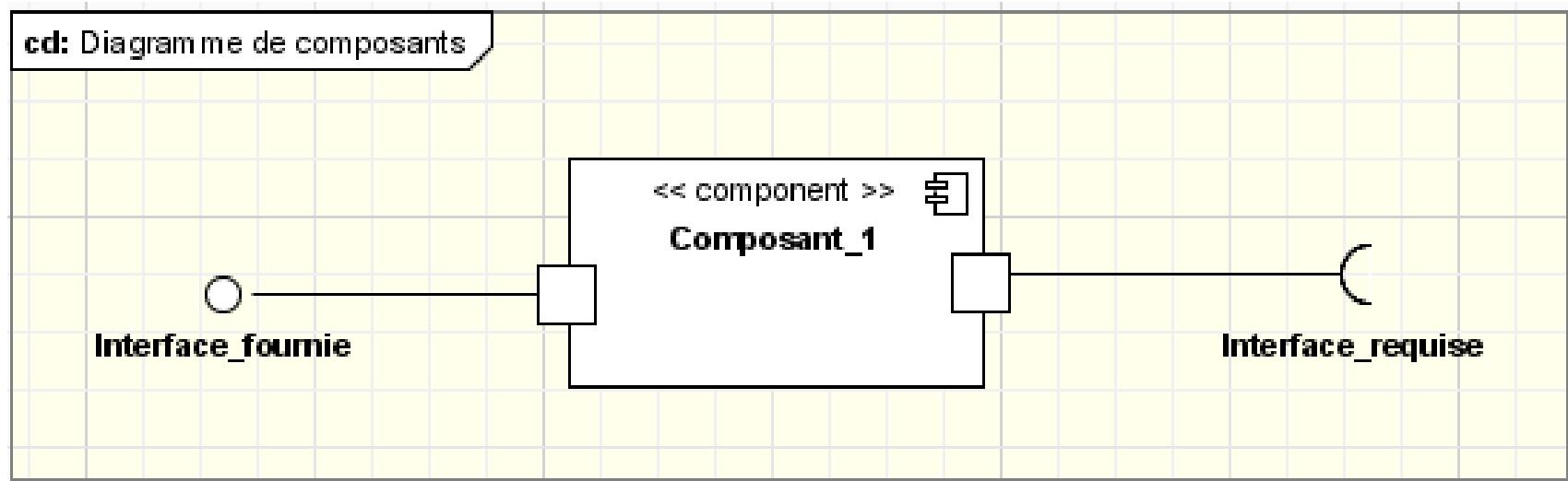


Générique

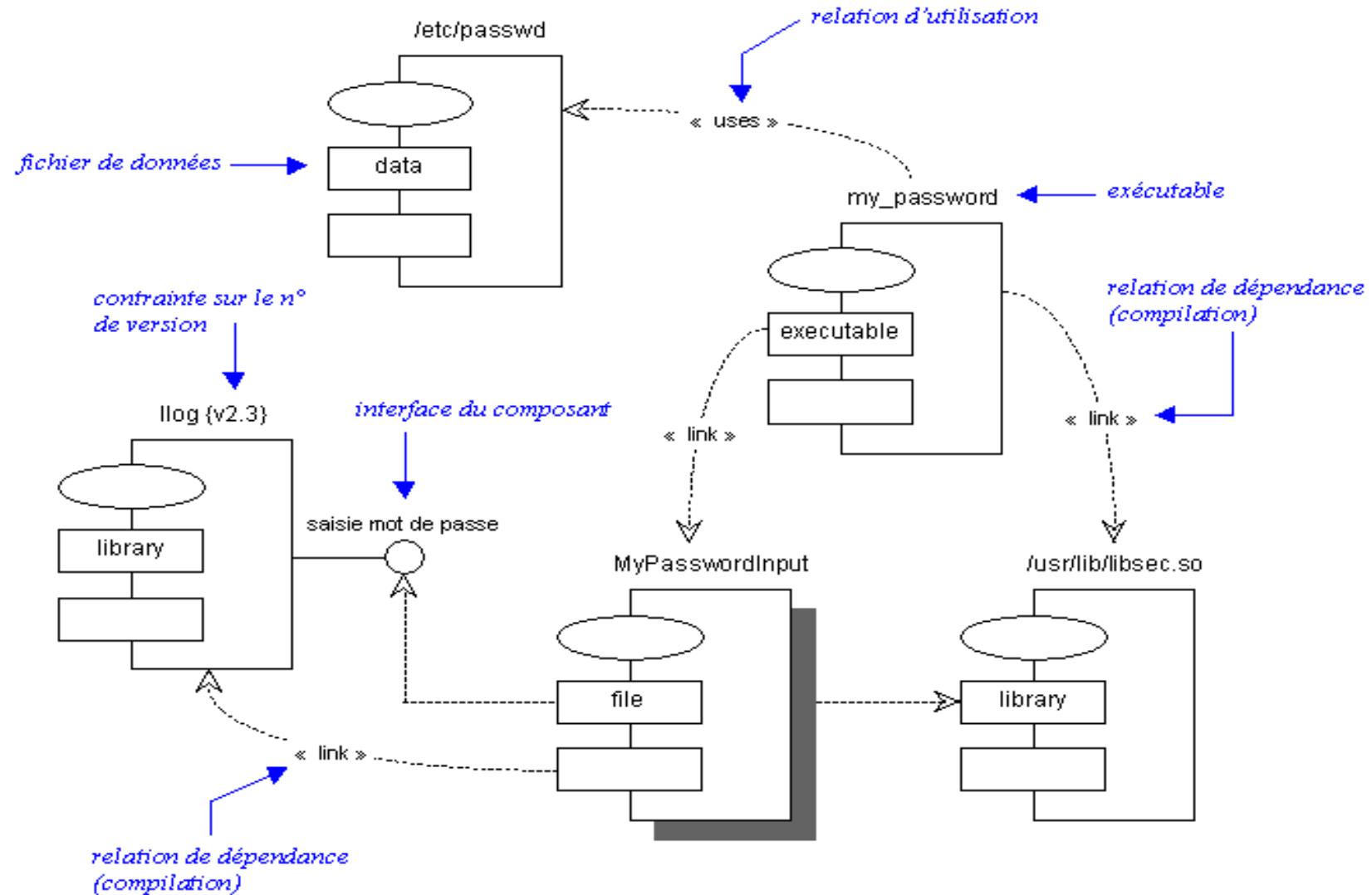


Représentation sous forme de classeur

- Classeur structuré stéréotypé «component».
- On peut faire figurer l'icône du composant dans l'angle supérieur droit.
- Un composant peut avoir des ports, des interfaces requises et d'autres interfaces fournies.



Exemple 1



Exemple 2

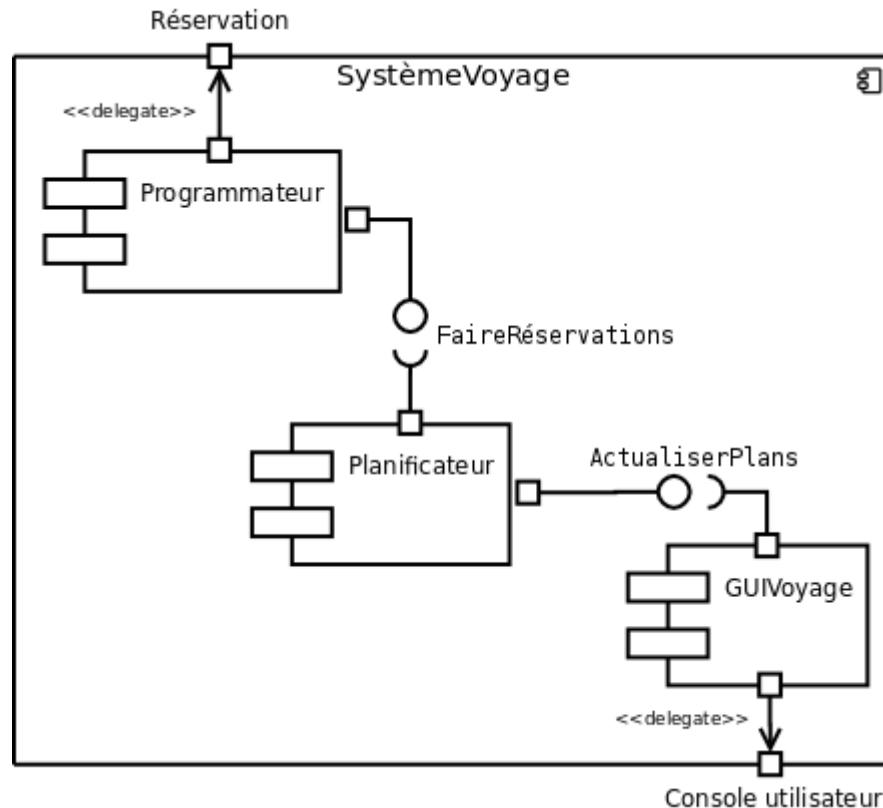
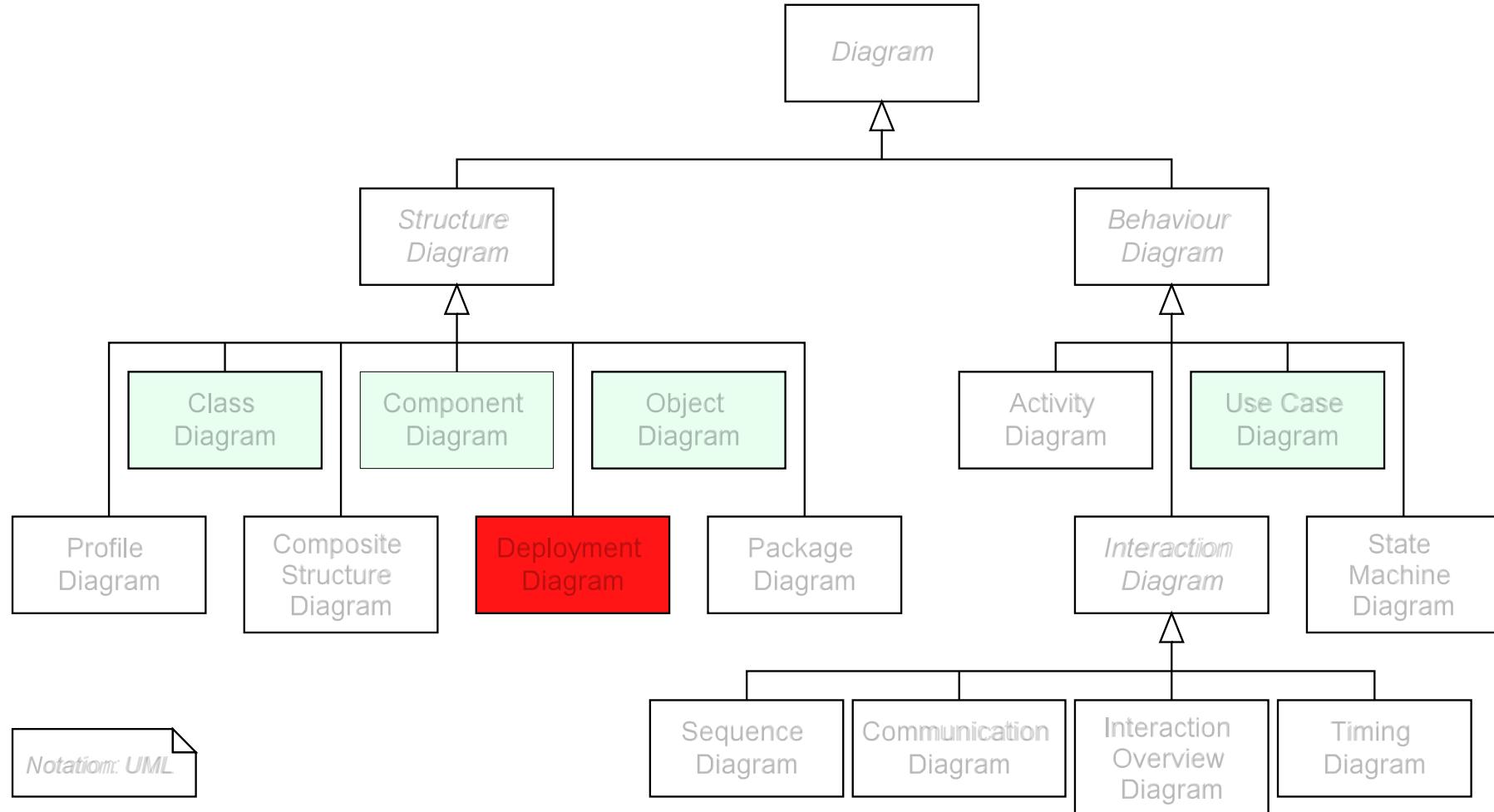


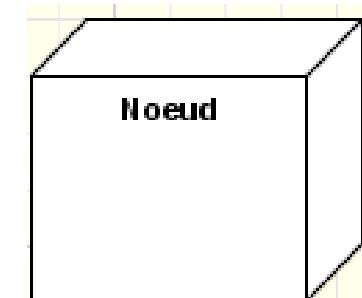
Diagramme de déploiement

Les diagrammes UML

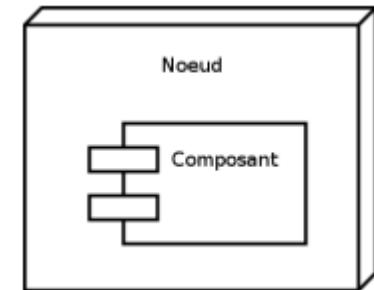
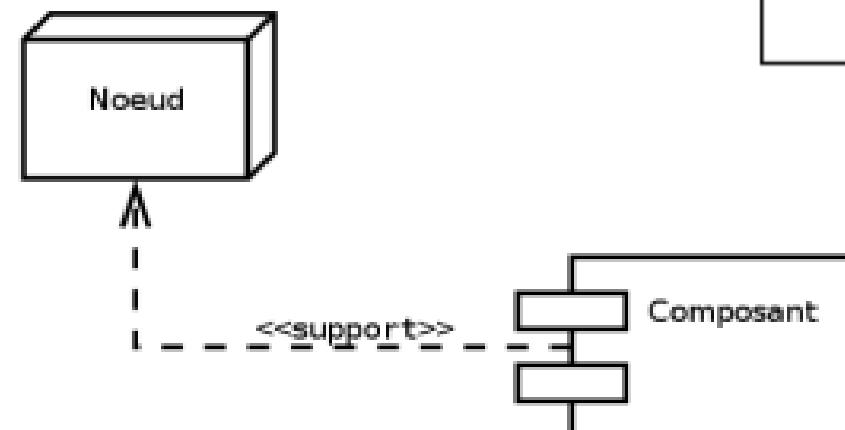


Description

- Description de la disposition physique des ressources matérielles du système.
- Vue sur la répartition des composants.
- Un nœud est représenté par un cube comportant un nom. Il peut contenir des attributs : processeur, mémoire, etc...

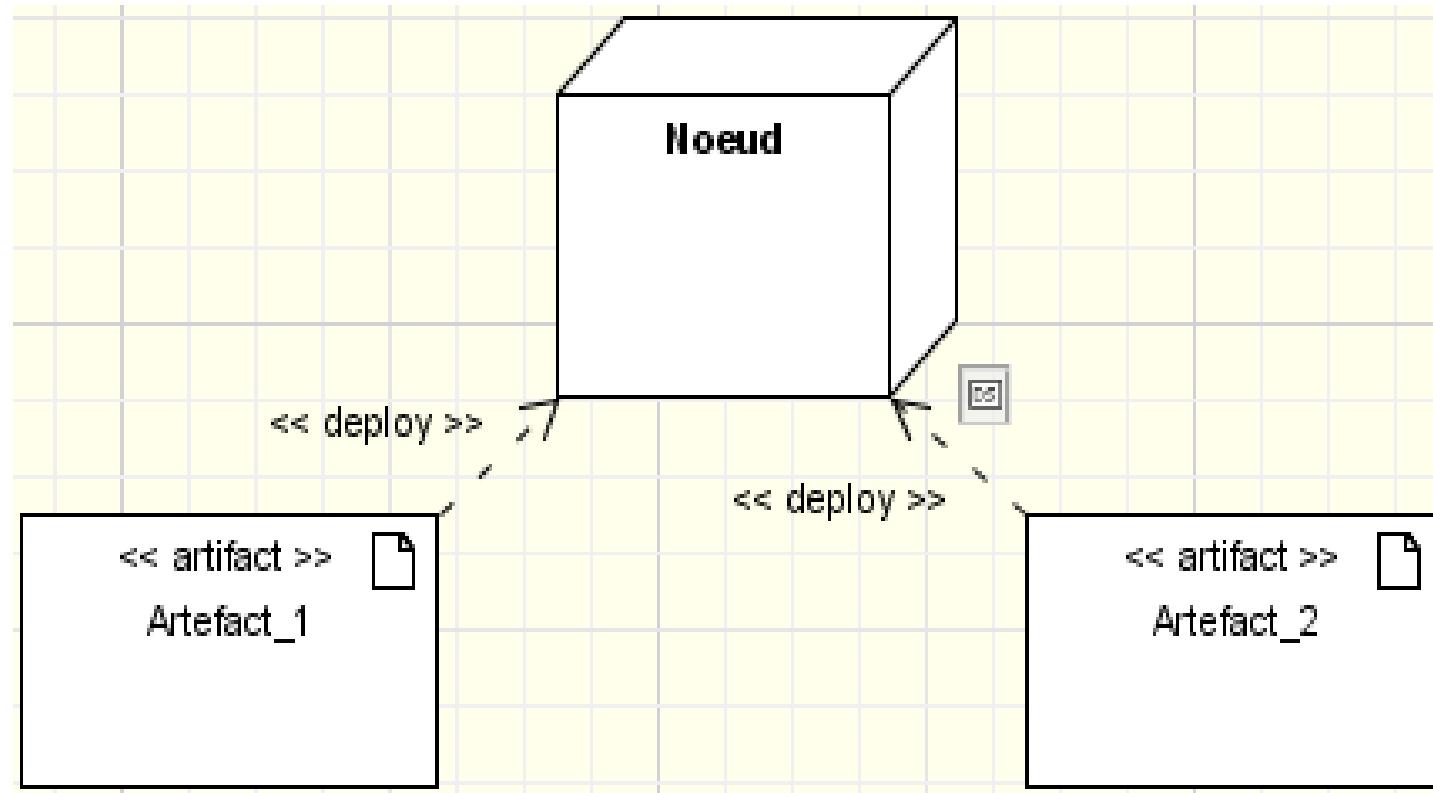


- L'affectation d'un composant à un nœud est faite soit en plaçant le composant dans le nœud, soit en le reliant avec une dépendance stéréotypée «support»



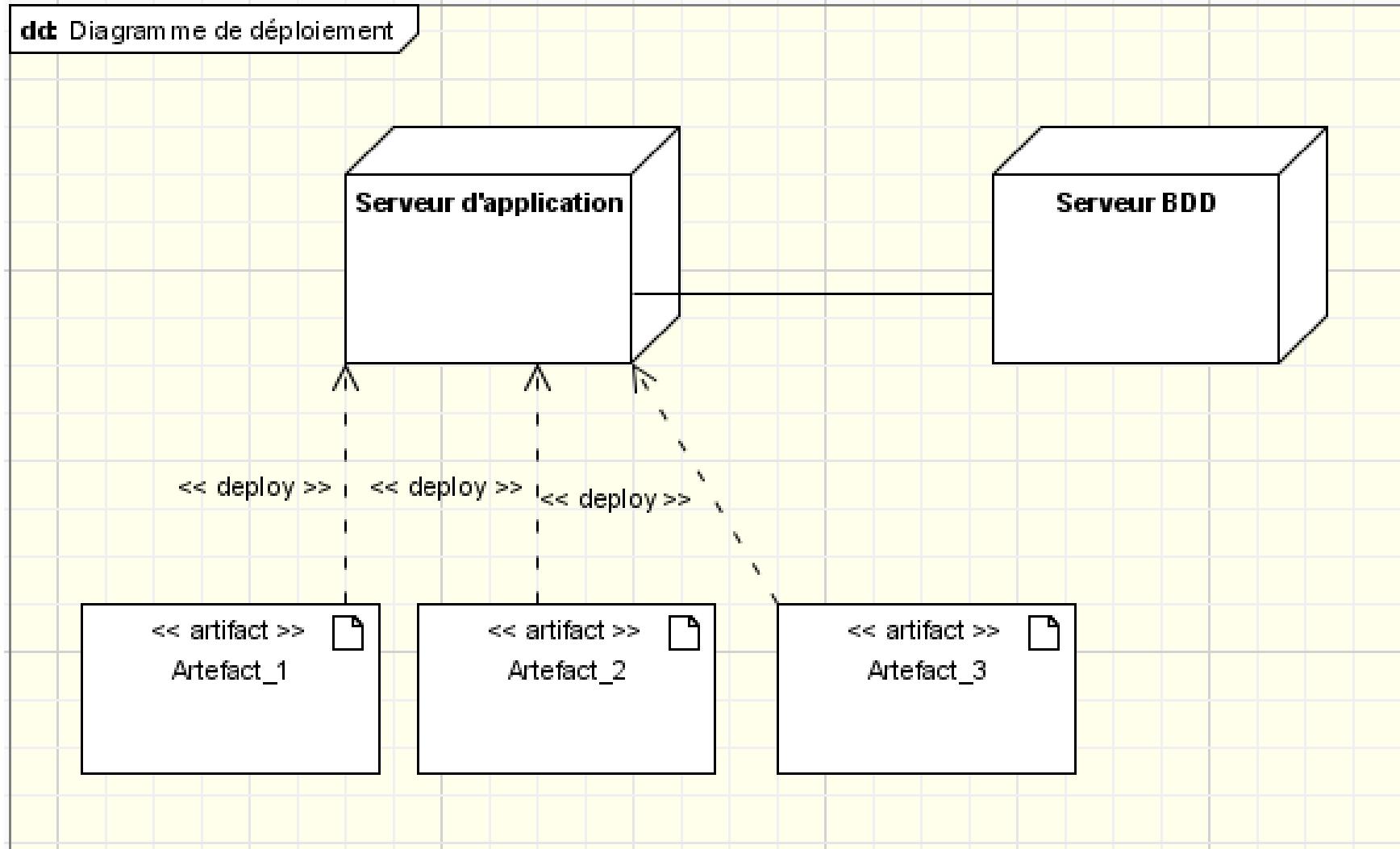
Artifact

- Élément concret existant dans le monde réel (document, exécutable, fichier, tables de BDD,...).

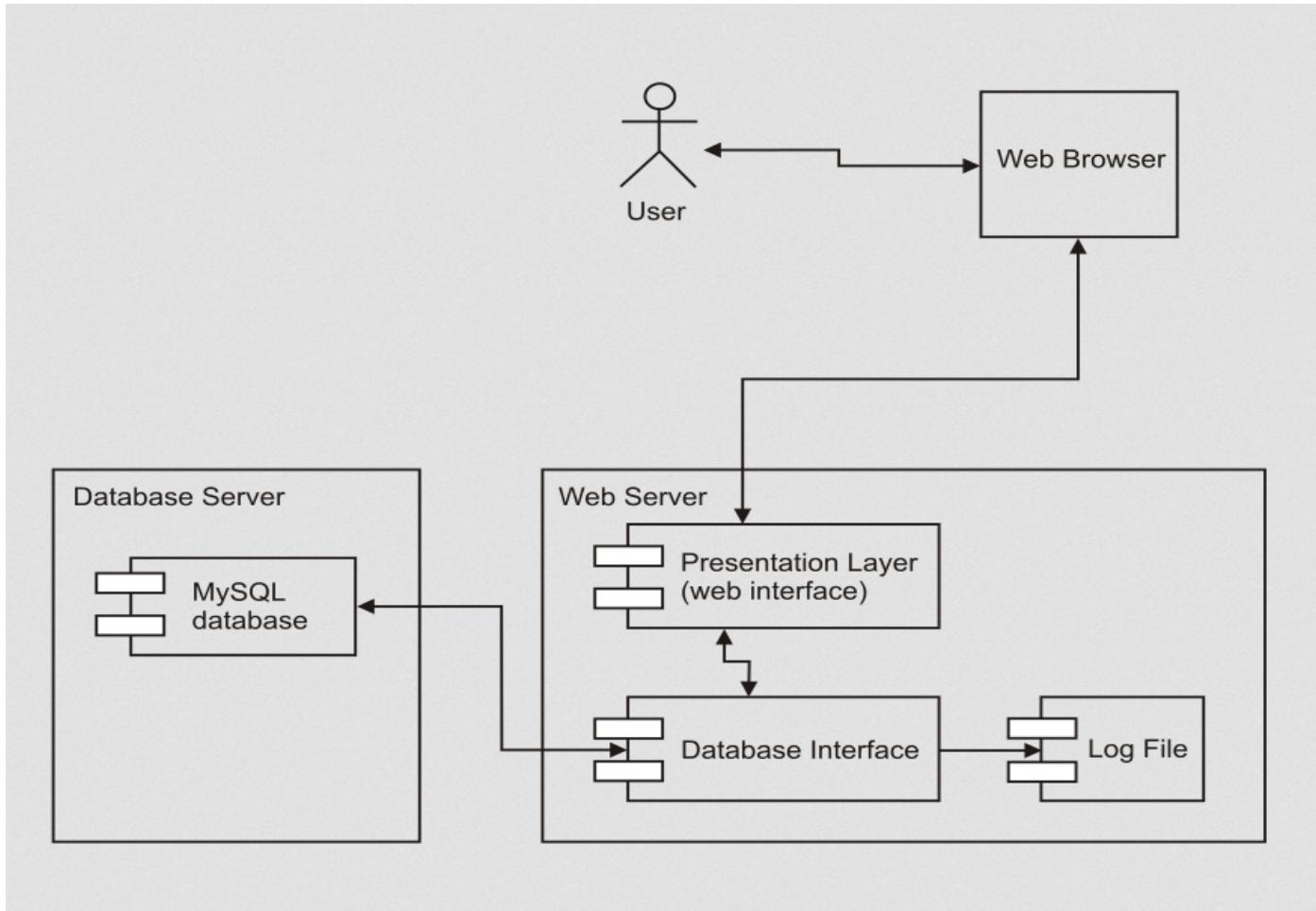


N.B. : on peut également représenter les artefacts à l'intérieur du nœud

Exemple 1



Exemple 2



Exemple 3

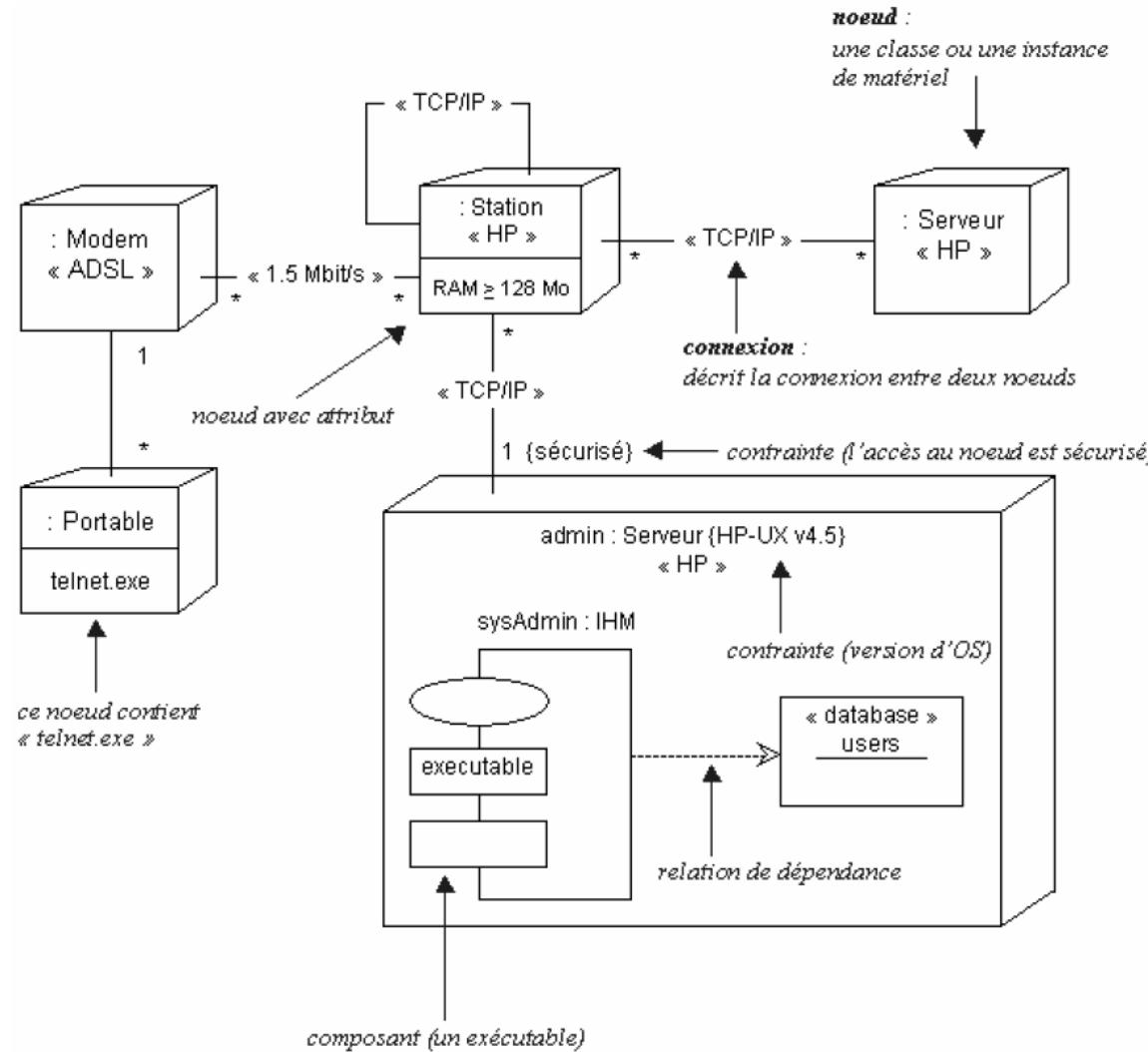
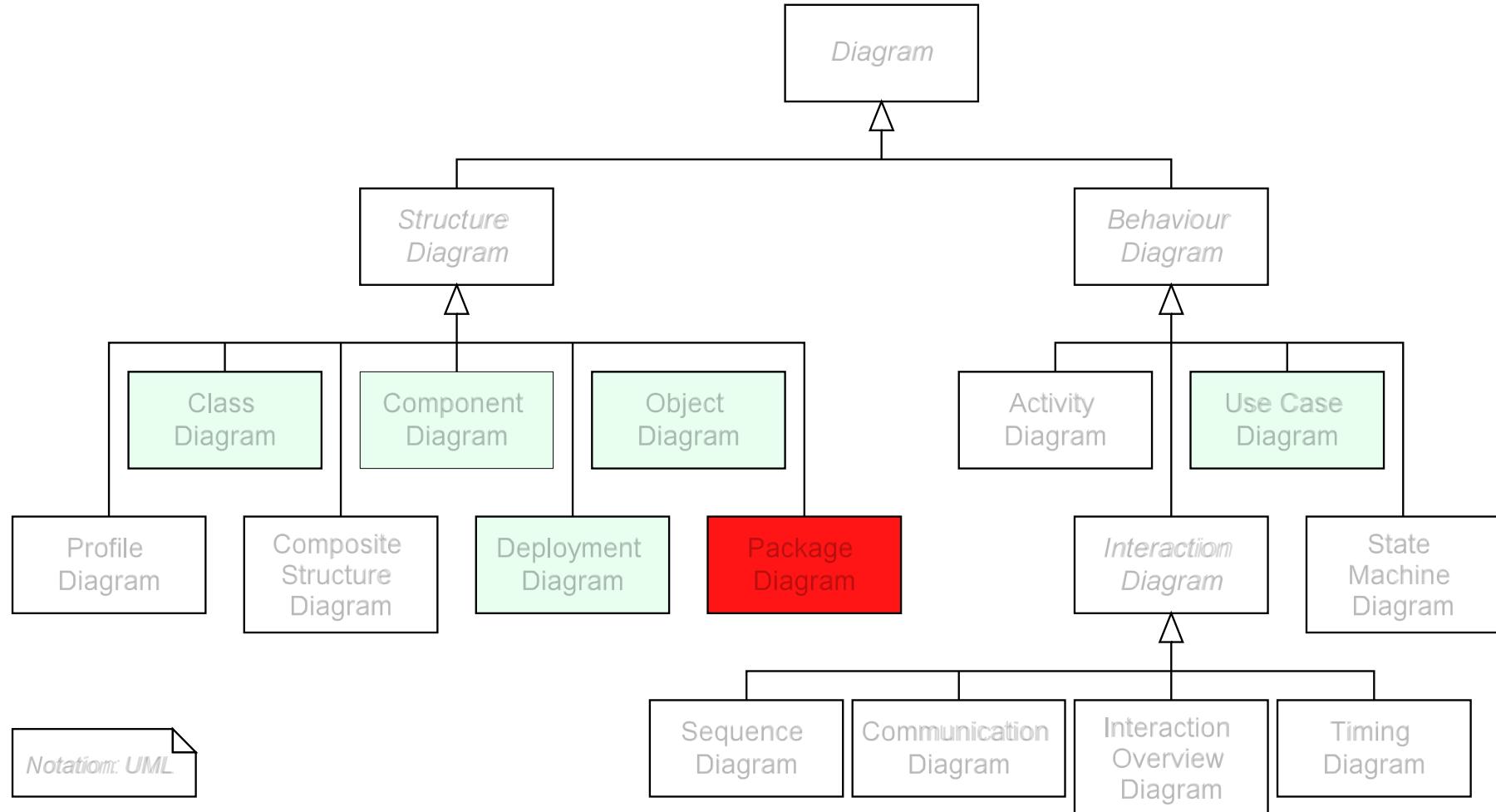


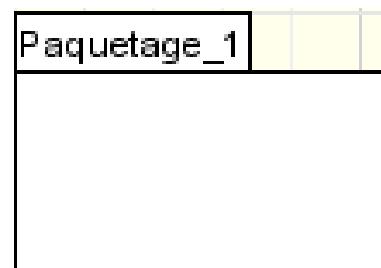
Diagramme de paquetage

Les diagrammes UML



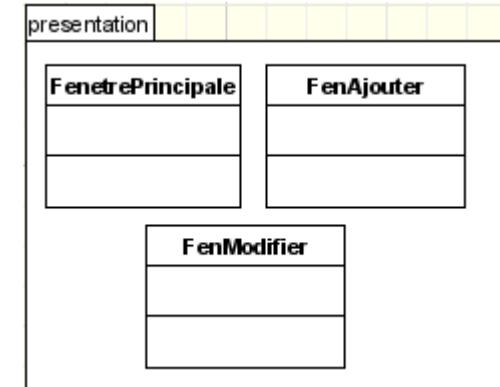
Description

- Paquet, paquetage (ou espace de nom, voire bibliothèque) : regroupement de classes, d'interfaces, de structures (ayant en commun le sujet, leur créateur, etc.)
- Montrer que contiennent les paquets et les relations de dépendances entre eux.
- Représenté graphiquement par une chemise



Relations entre paquets

- On peut montrer les classes appartenant au paquetages, ou seulement celles qui y sont publiques ou importantes



- On peut représenter des sous-paquetages

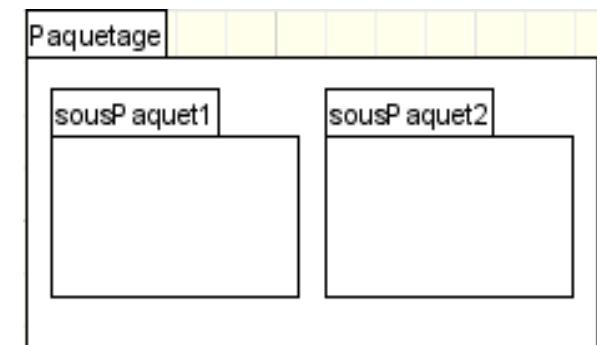
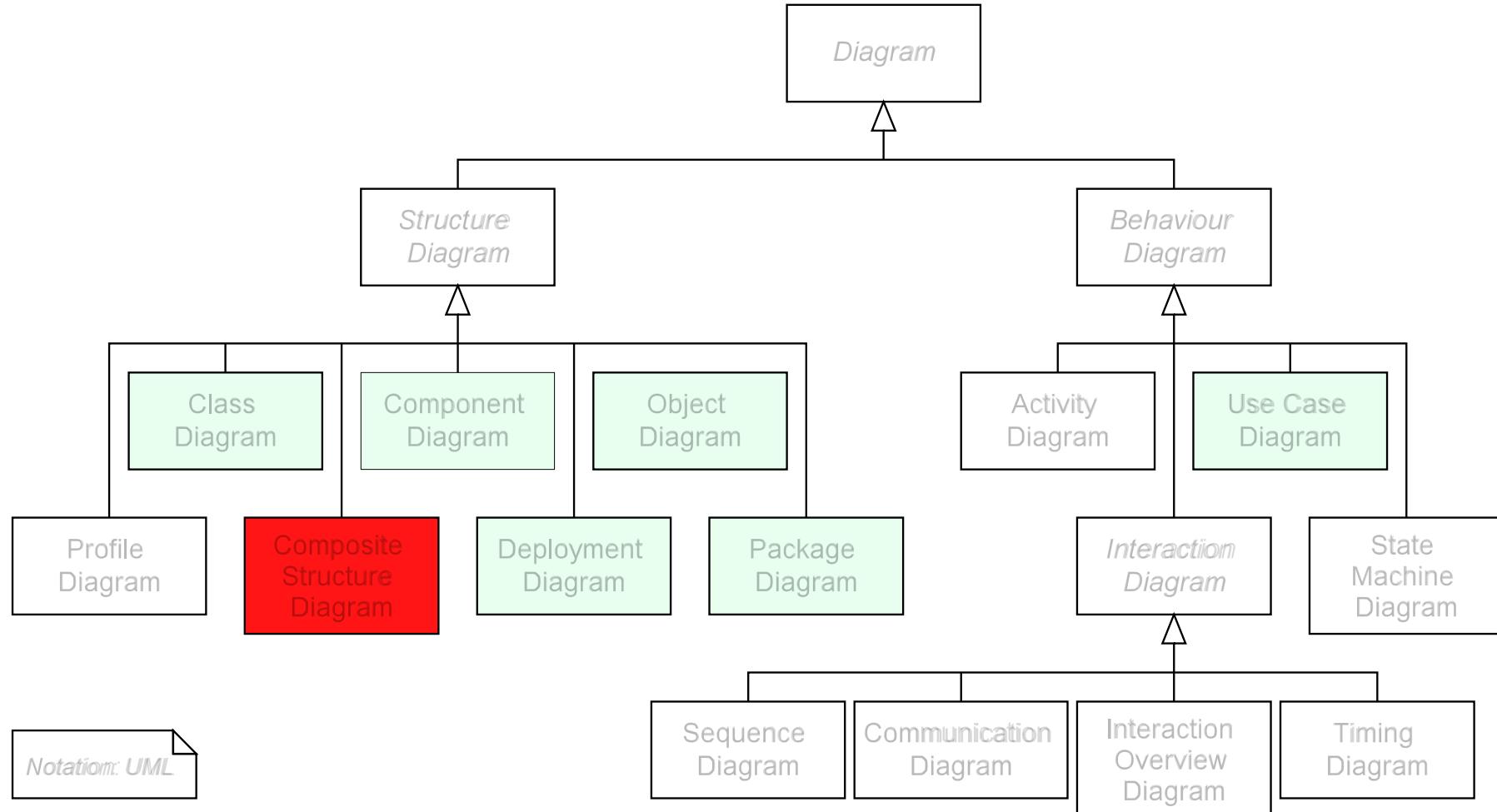


Diagramme de structure composite (diagramme d'architecture)

Les diagrammes UML



Description

- Existe depuis UML 2.0
- Décrire la structure interne d'un classifieur (classe, objet, interface...), et les détails de la collaboration avec d'autres
- Les éléments clés du diagramme de structure composite sont :
 - Classifieurs structurés
 - Parties
 - Ports
 - Connecteurs
 - Collaborations.

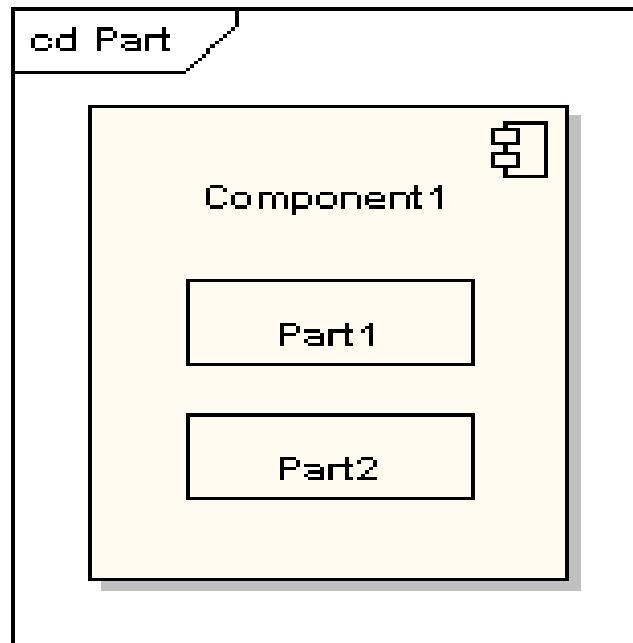
Classifieur structuré

- Symbolise une classe, un composant, un nœud de déploiement...
- La fonction de ce classifieur peut être décrite par la façon dont il interagit avec d'autres éléments
- Contient les parties

Une Partie

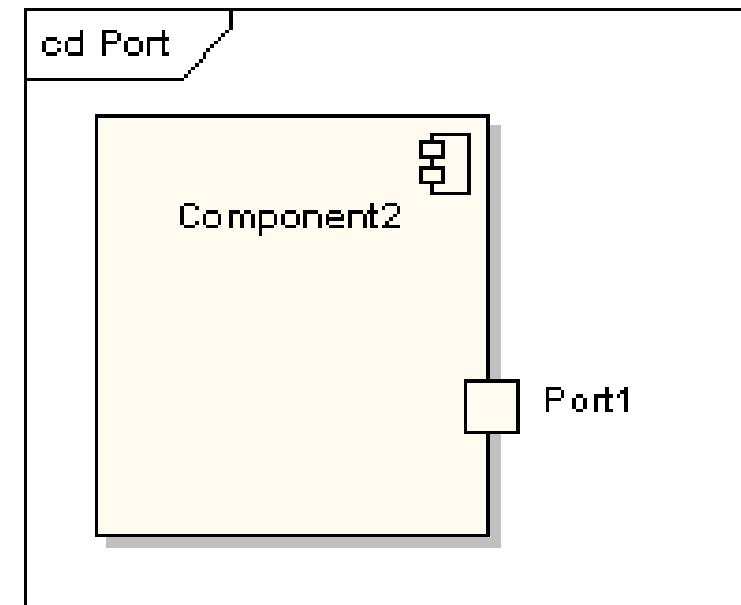
- Élément (une ou plusieurs instances)
- Possédé (composition) par le classifieur conteneur

Exemple (un composant avec deux parties) :



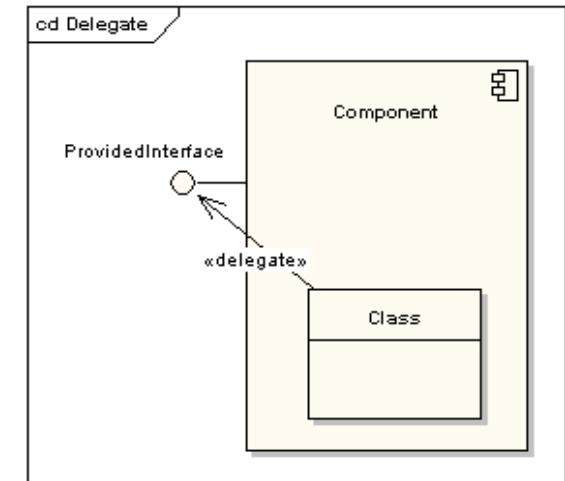
Un port

- Point pour l'interaction entre:
- Une instance d'un classifieur et son environnement ou
- Une instance du classifieur et ses parties
- Exemple (un composant avec un port) :



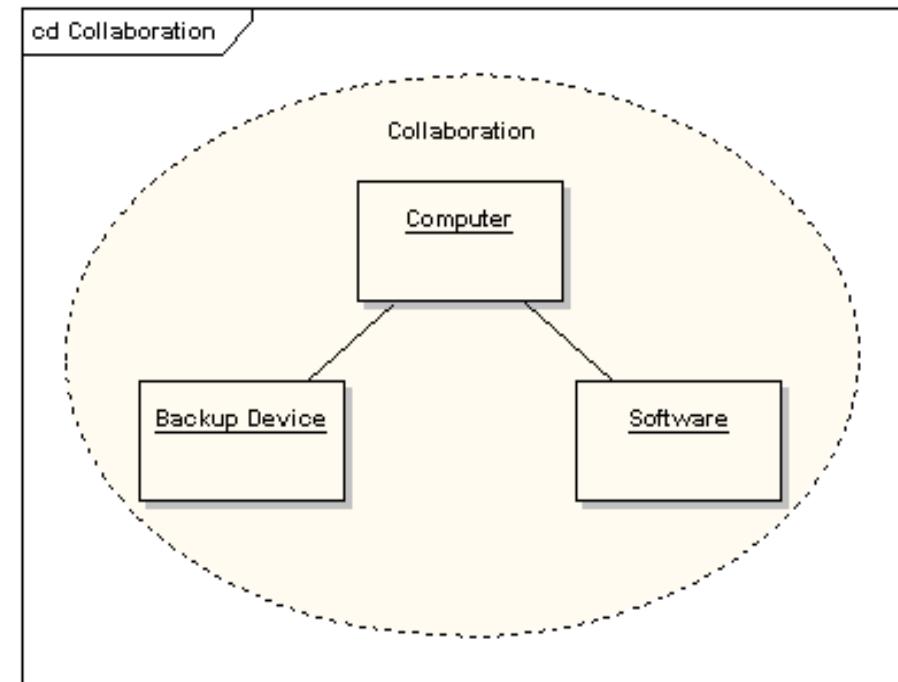
Un connecteur

- Représente une relation
- Connecteur d'assemblage (ligne simple) : entre deux parties
- Connecteur de délégué : (flèche) entre une partie interne et vers l'extérieur (souvent : un port)
- Exemple - connecteur de délégué :

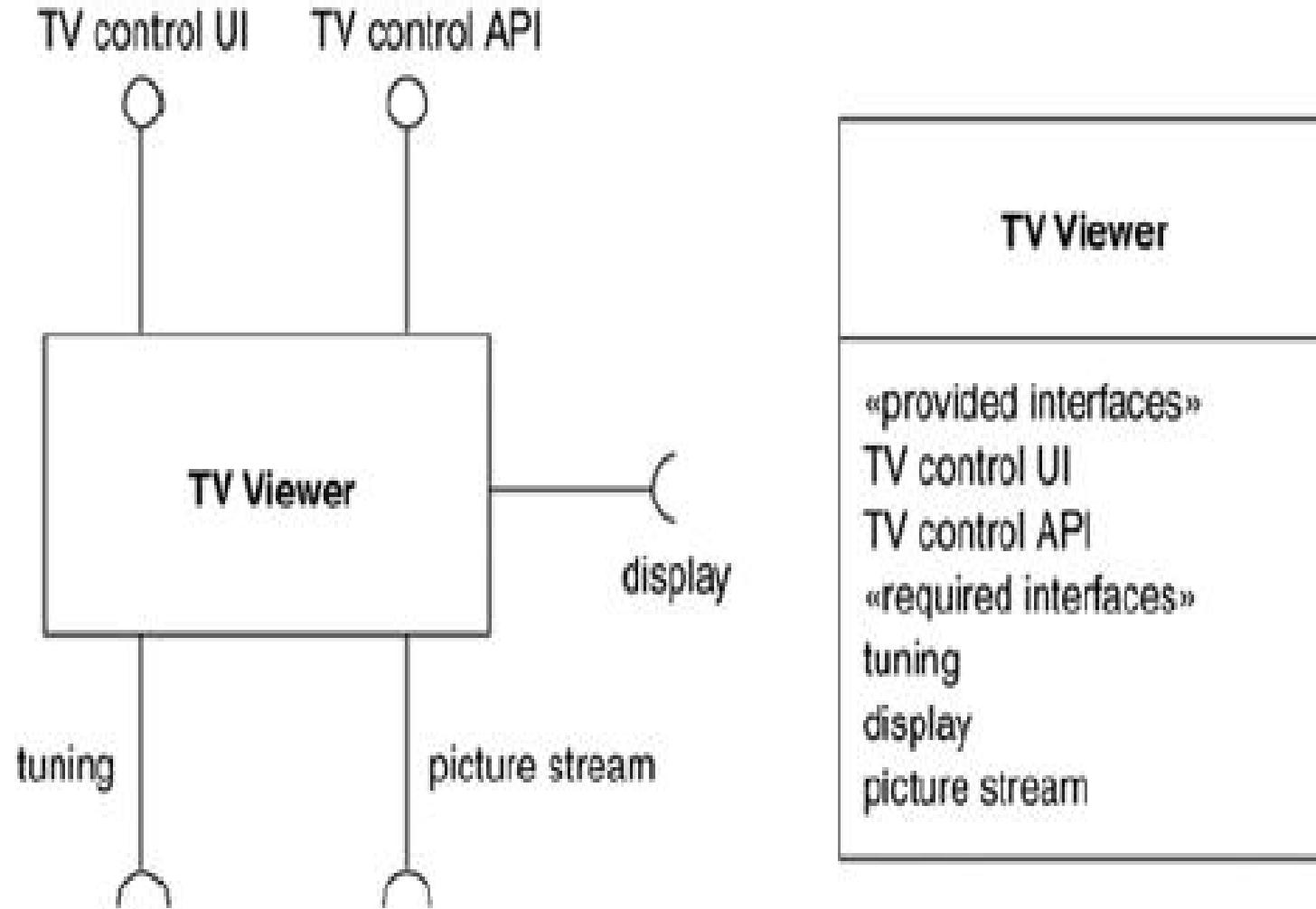


Une collaboration

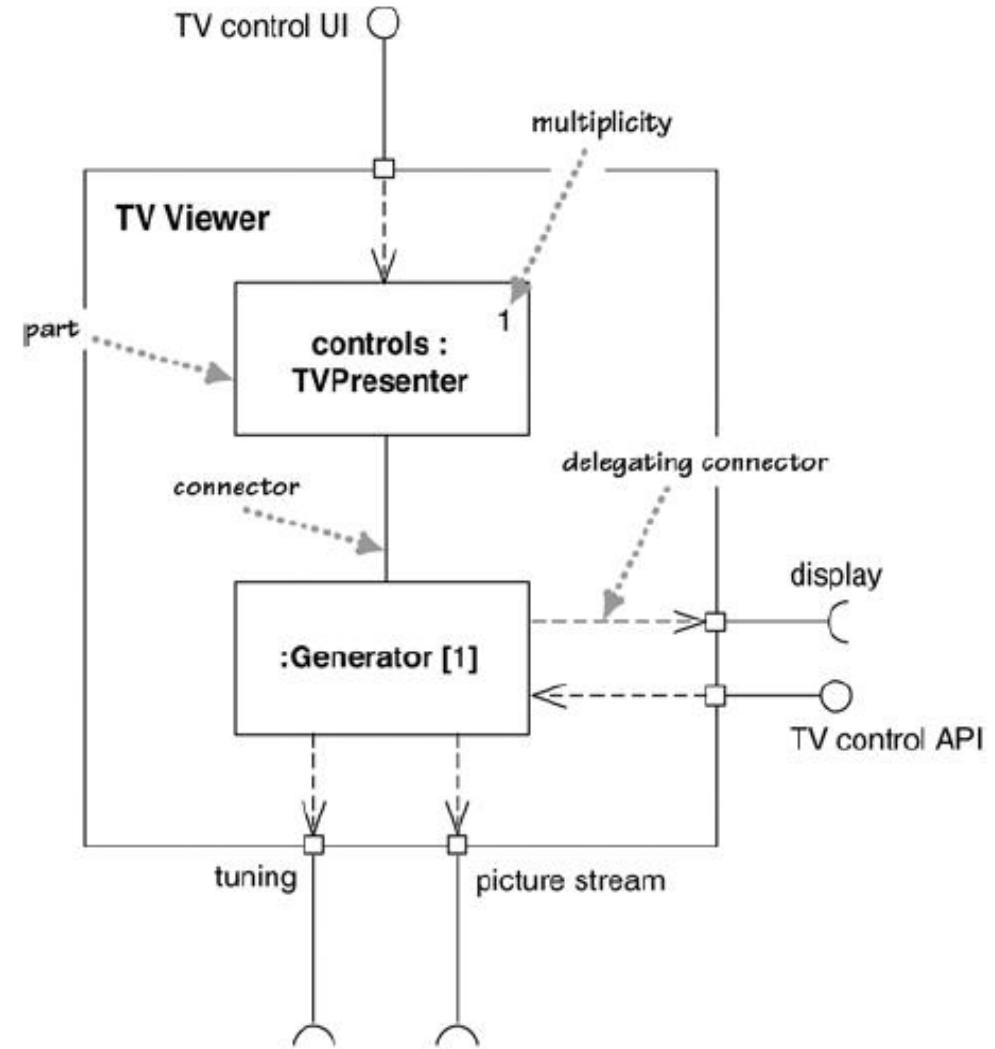
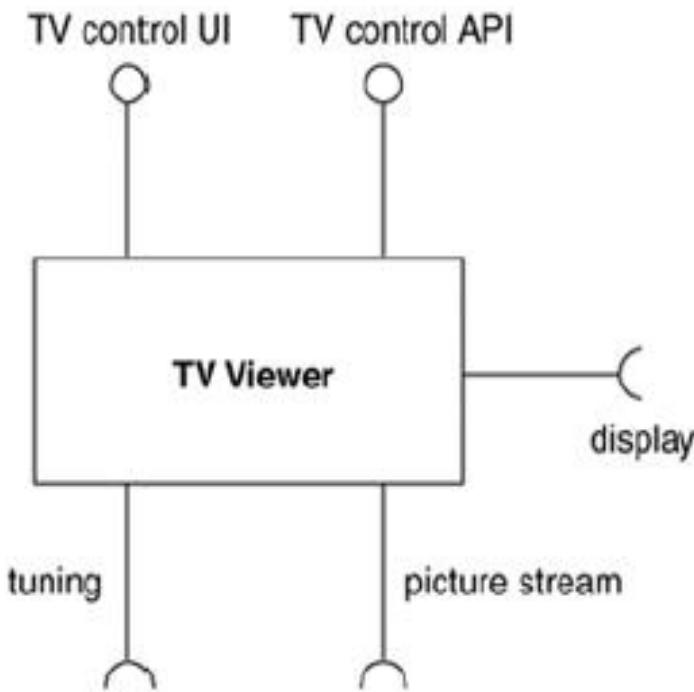
- Montre des rôles collaborant (nécessaires) pour une fonction particulière
- Ellipse en pointillés
- Exemple :



Exemple



Exemple (suite)

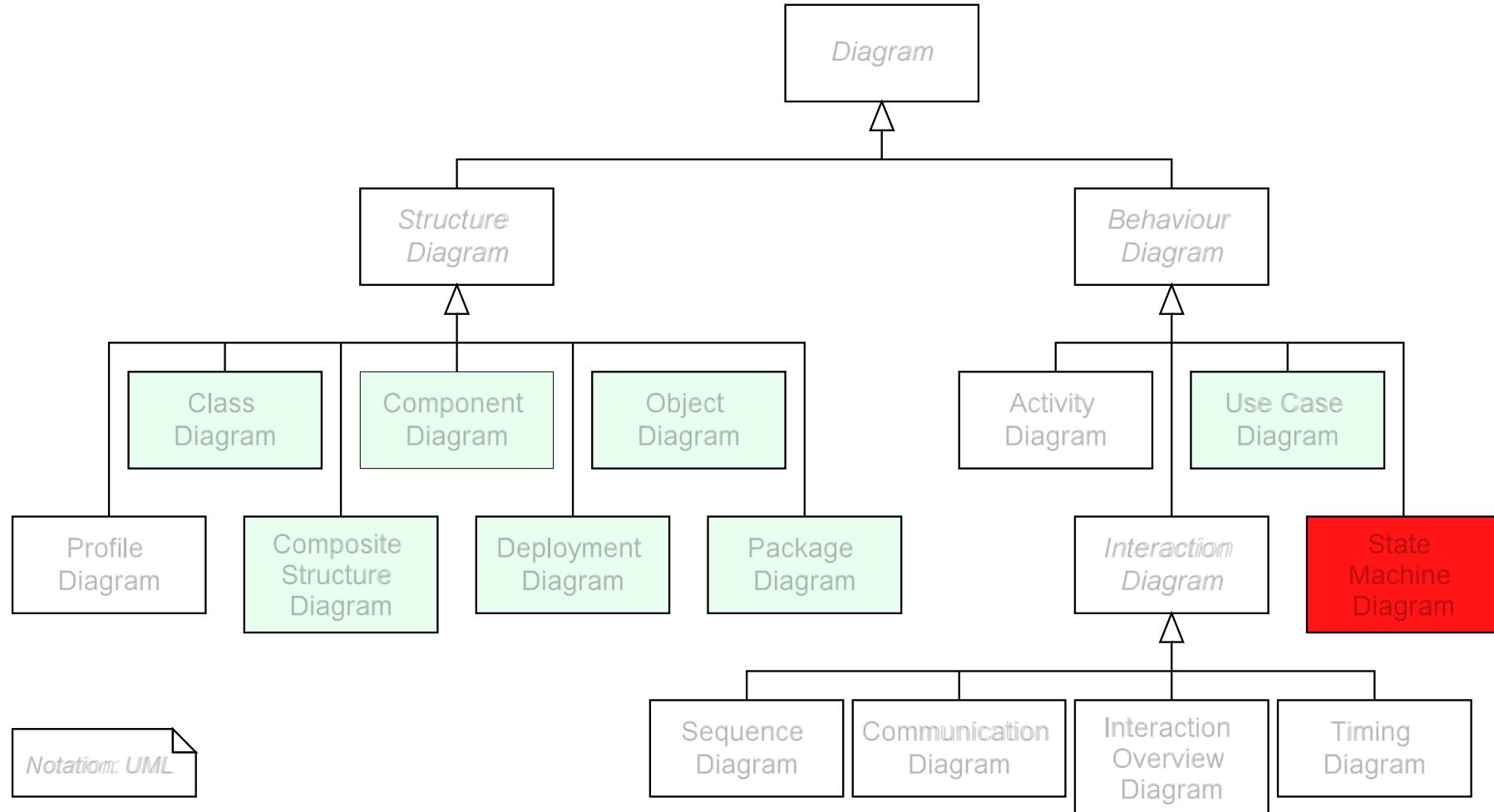


Diagrammes Comportementaux

- Diagramme états-transitions
- Diagramme d'activités

Diagramme états-transitions

Les diagrammes UML



Description

- Décrire le comportement interne d'un objet ou d'un composant en fonction de ses états et des transitions entre ces états.



Eléments du diagramme

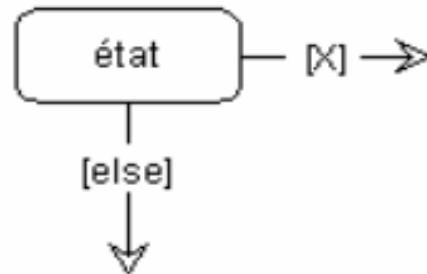
- Etat initial :



- Etat final :



- Transition conditionnelle :



- Déclencheur : activation de la transition par un stimulus externe
il est indiqué sur la flèche précédant la transition

- Action : résultat de la transition.

L'action est indiquée sur la flèche de transition après un « / ».
Elle suit le déclencheur et la condition.

Eléments du diagramme (2)

- Actions propres à un état :

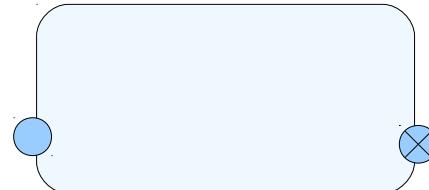
- **entry / action** : action exécutée à l'entrée de l'état.
- **exit / action** : action exécutée à la sortie de l'état.
- **on événement / action** : action exécutée à chaque fois que l'événement cité survient.
- **do / action** : action récurrente ou significative, exécutée dans l'état.

- Etats imbriqués et états composites :

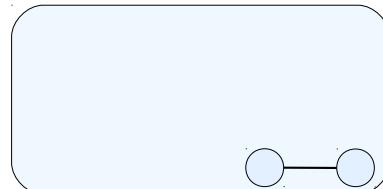
- **Etats imbriqués** : représentation d'une activité impliquant des sous-activités concurrentes ou asynchrones.
- **Etats composites** : regroupement d'états

Eléments du diagramme (3)

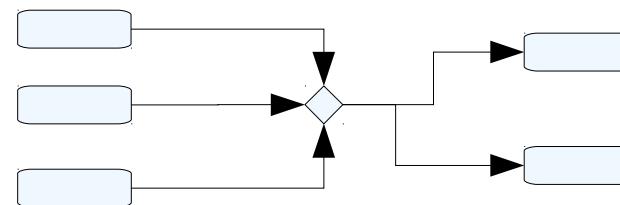
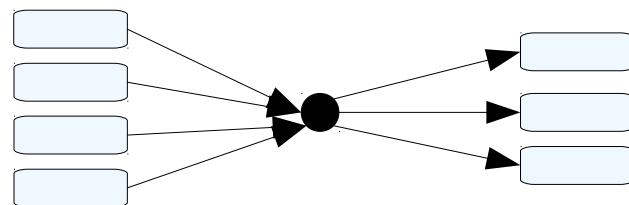
- Point d'entrée, de sortie :



- Etat composite :



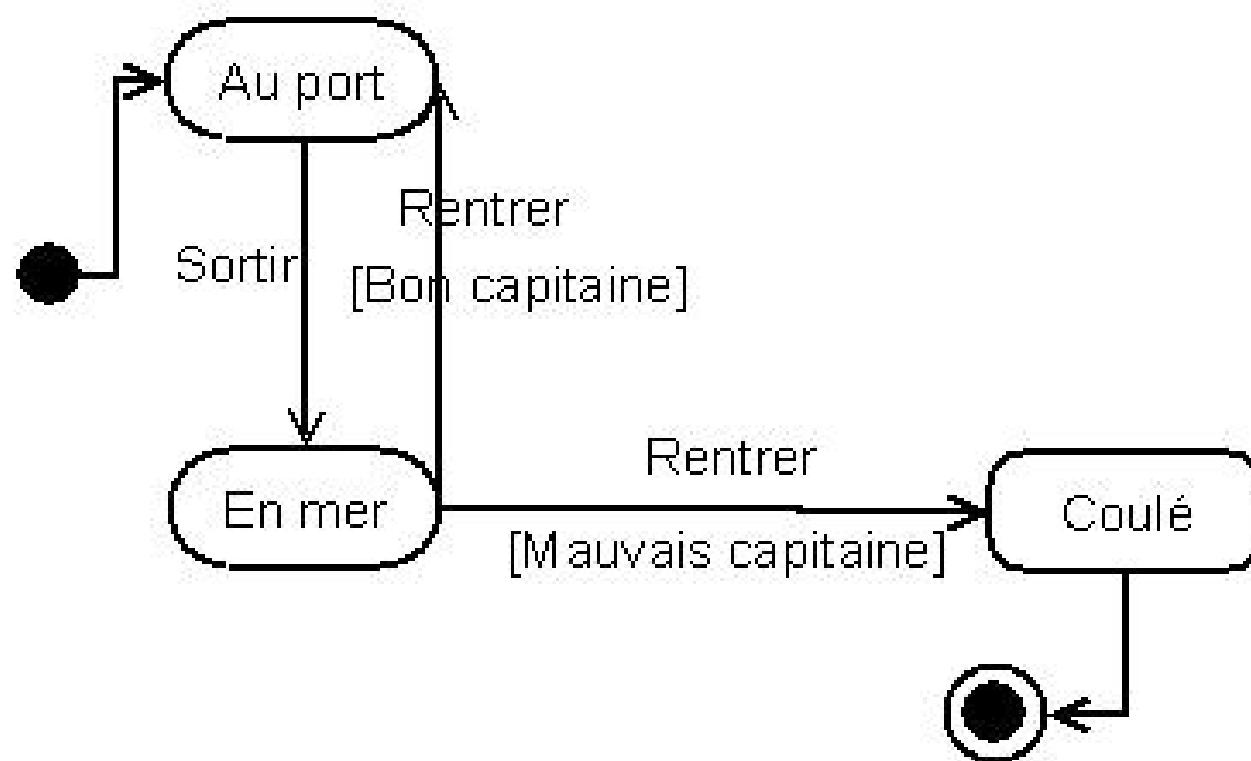
- Point de jonction, de choix :



- Etat historique



Exemple



Etats : imbriqués/composites

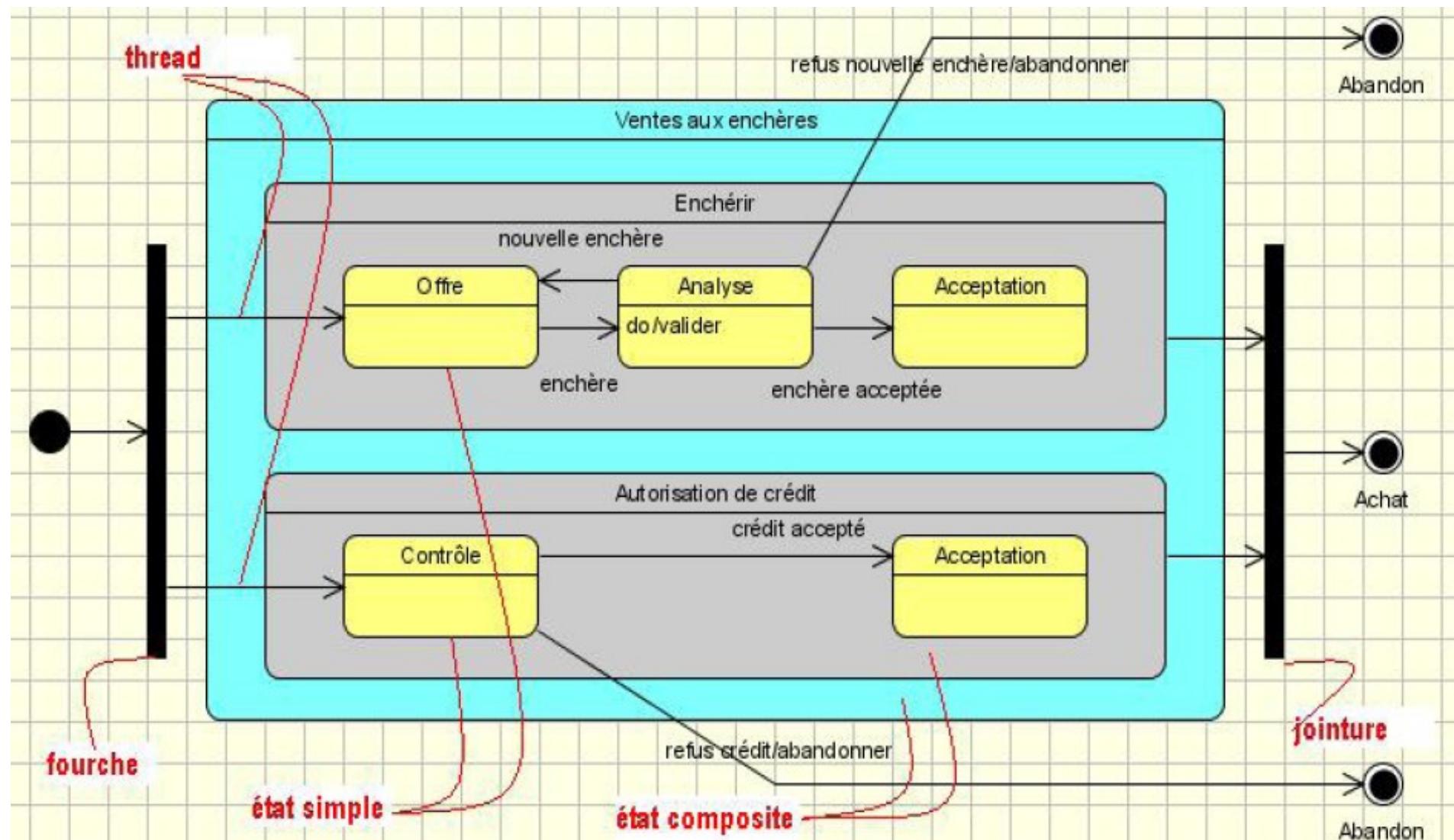
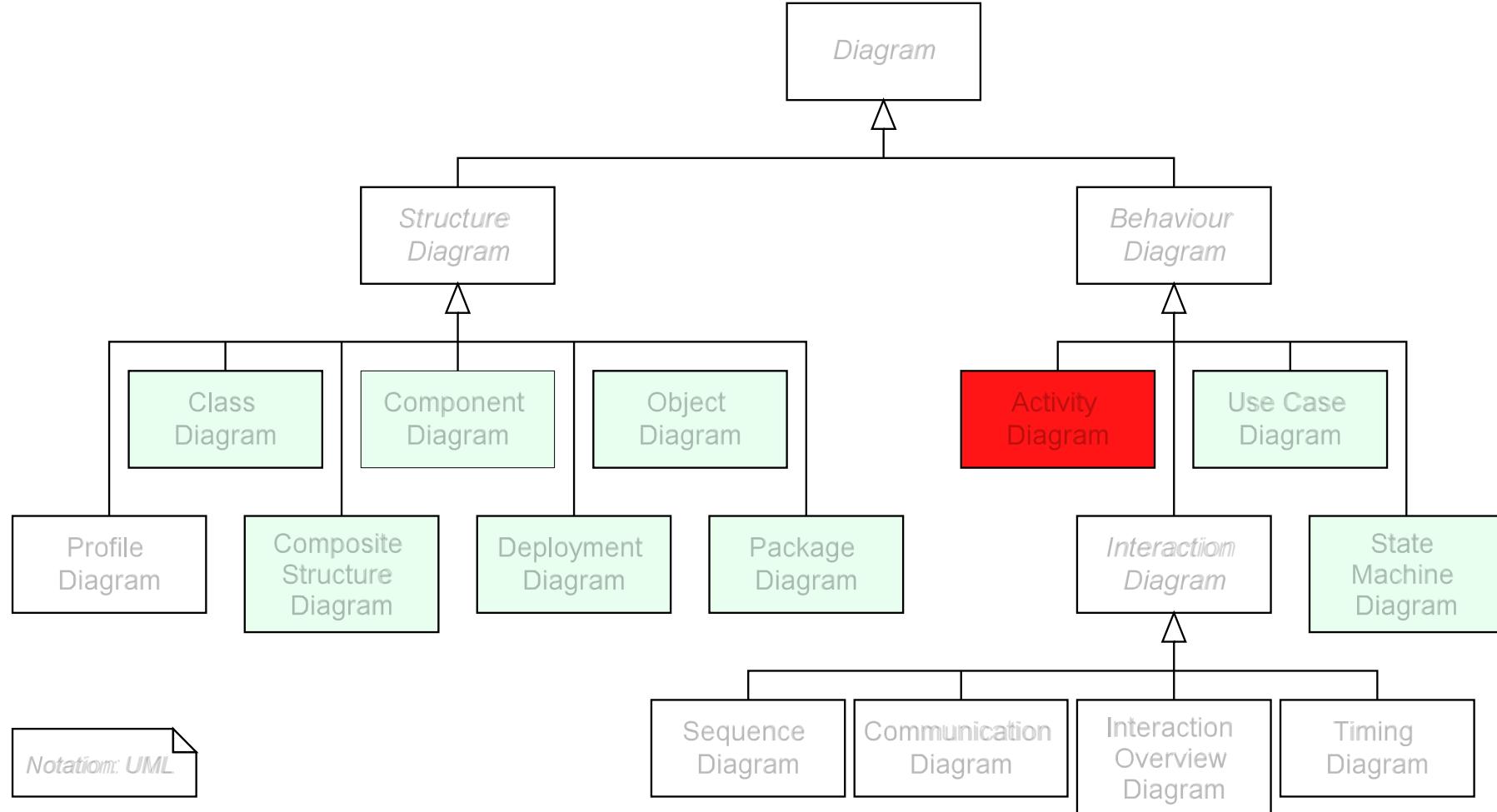


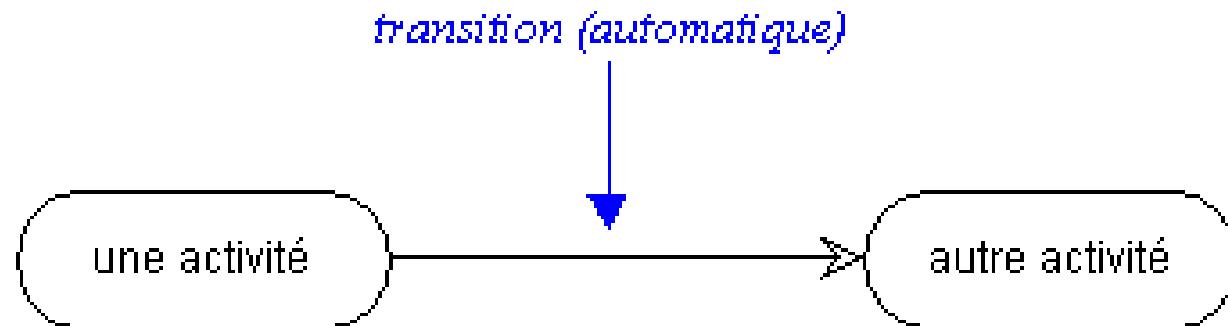
Diagramme d'activités

Les diagrammes UML



Description

- Représentation du déroulement d'un cas d'utilisation ou d'une méthode.
- Variante du diagramme d'états-transitions :
 - * Les transitions sont automatiques entre la fin d'une étape et le début de la suivante.
 - * Les diagrammes d'états-transitions mettent l'accent sur la traversée d'un processus (*process*) par un objet, alors que, les diagrammes d'activités se focalisent sur le flux d'activités concourant à la réalisation d'un processus

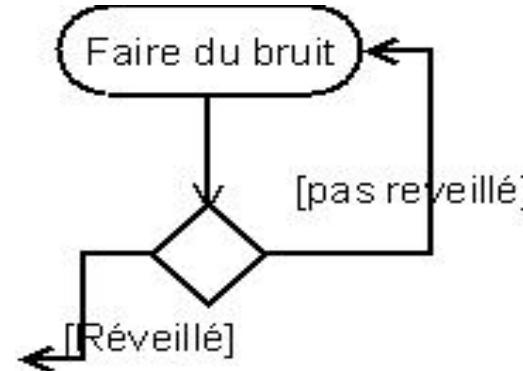


Eléments du diagramme

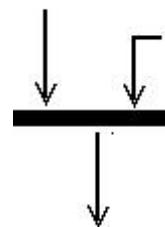
- Début et fin de flux d'activité



- Les transitions conditionnelles

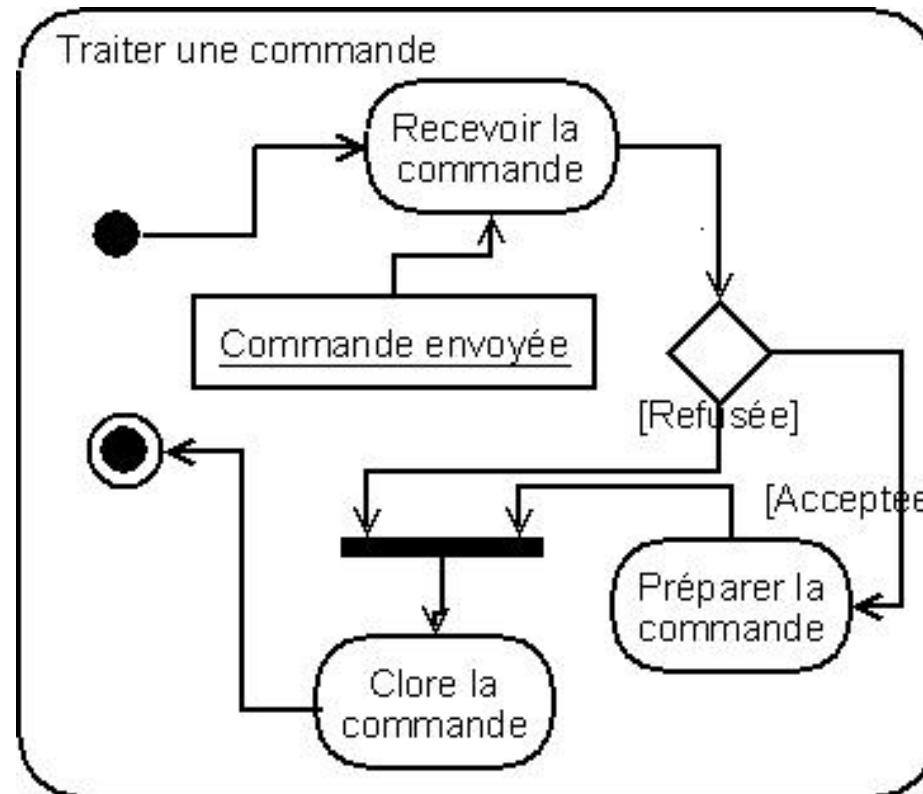


- Séparations et jointures



- Activité (séparable en parties), objets : rectangle

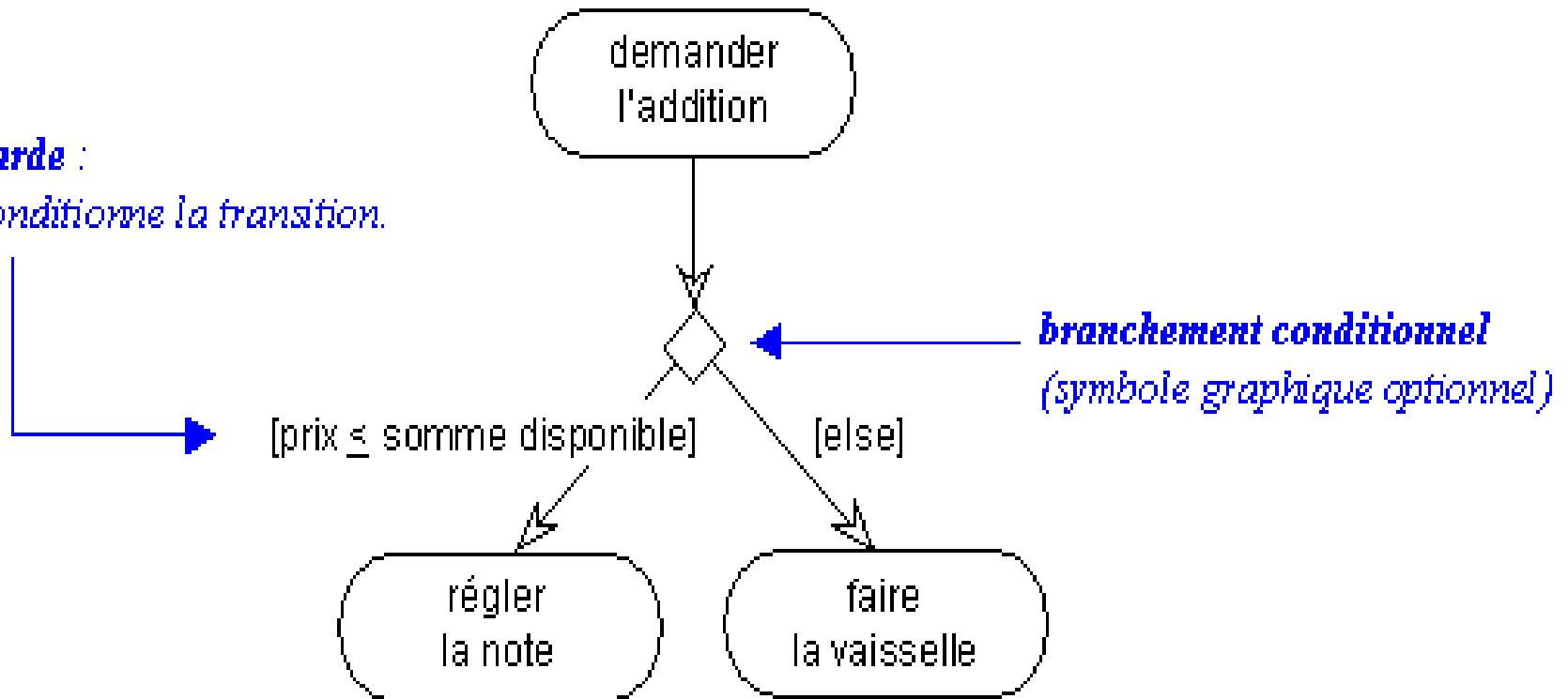
Exemple



Exemple (2)

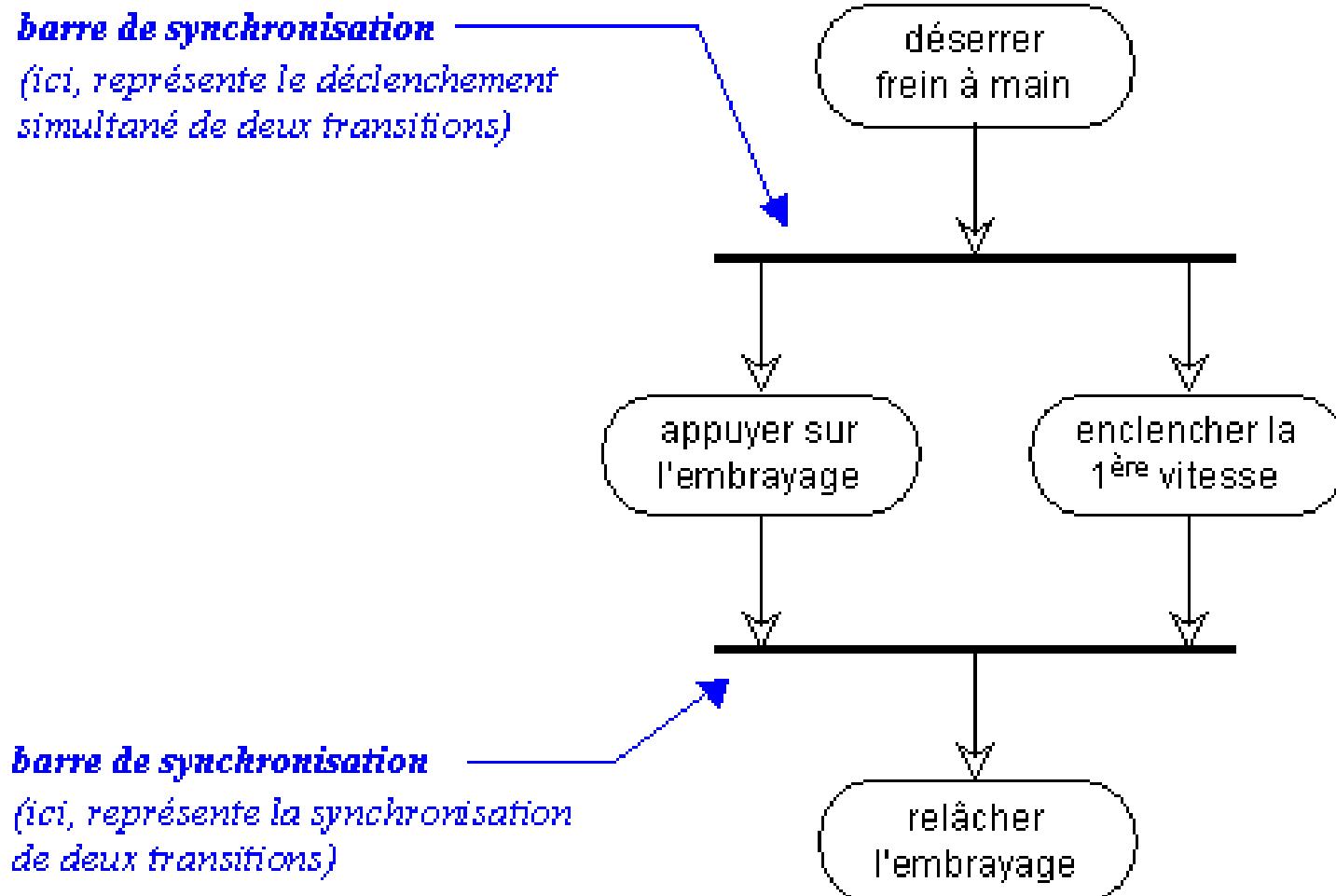
garde :

conditionne la transition.

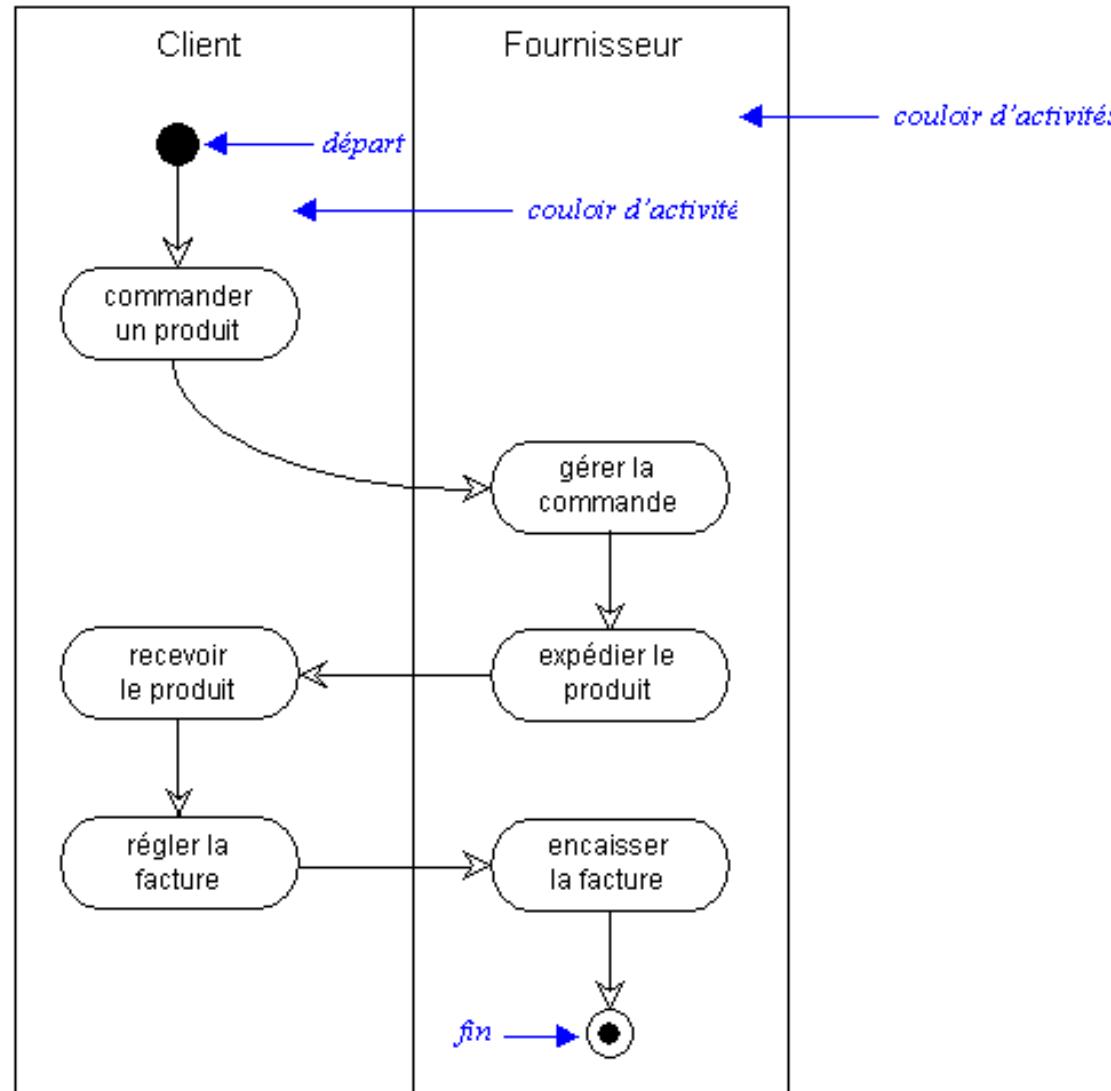


*branchement conditionnel
(symbole graphique optionnel)*

Synchronisation et activités parallèles



Couloirs d'activités

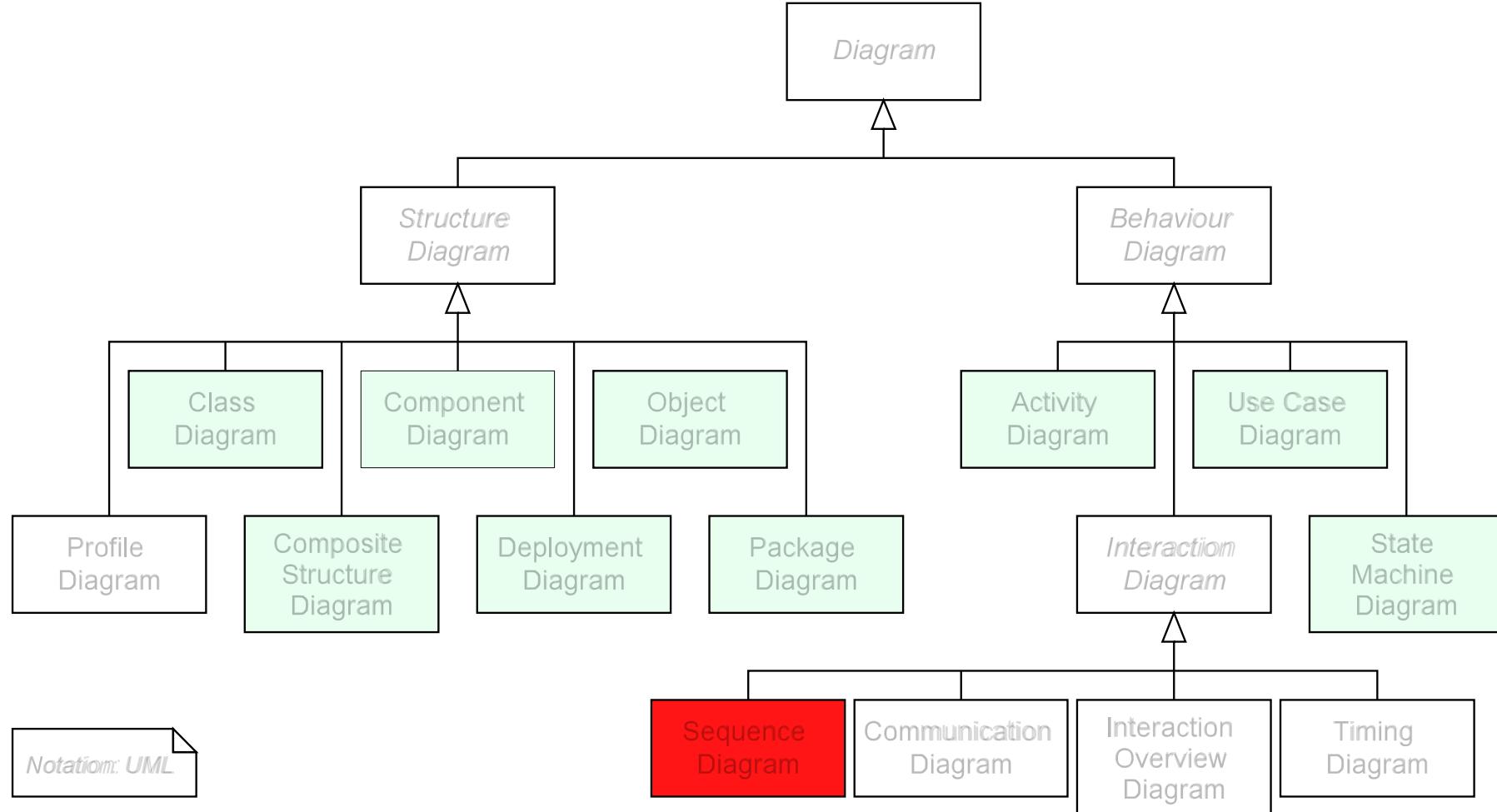


Diagrammes d'interactions (dynamiques)

- Diagramme de séquence
- Diagramme de communication
- Diagramme global d'interaction
- Diagramme de temps.

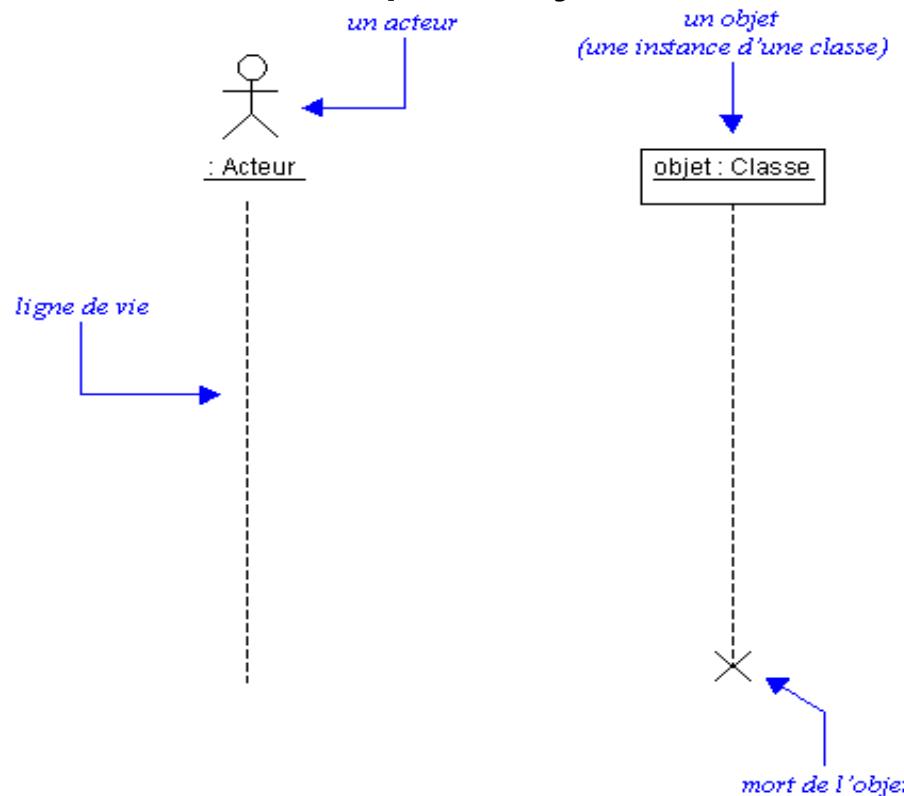
Diagramme de séquence

Les diagrammes UML

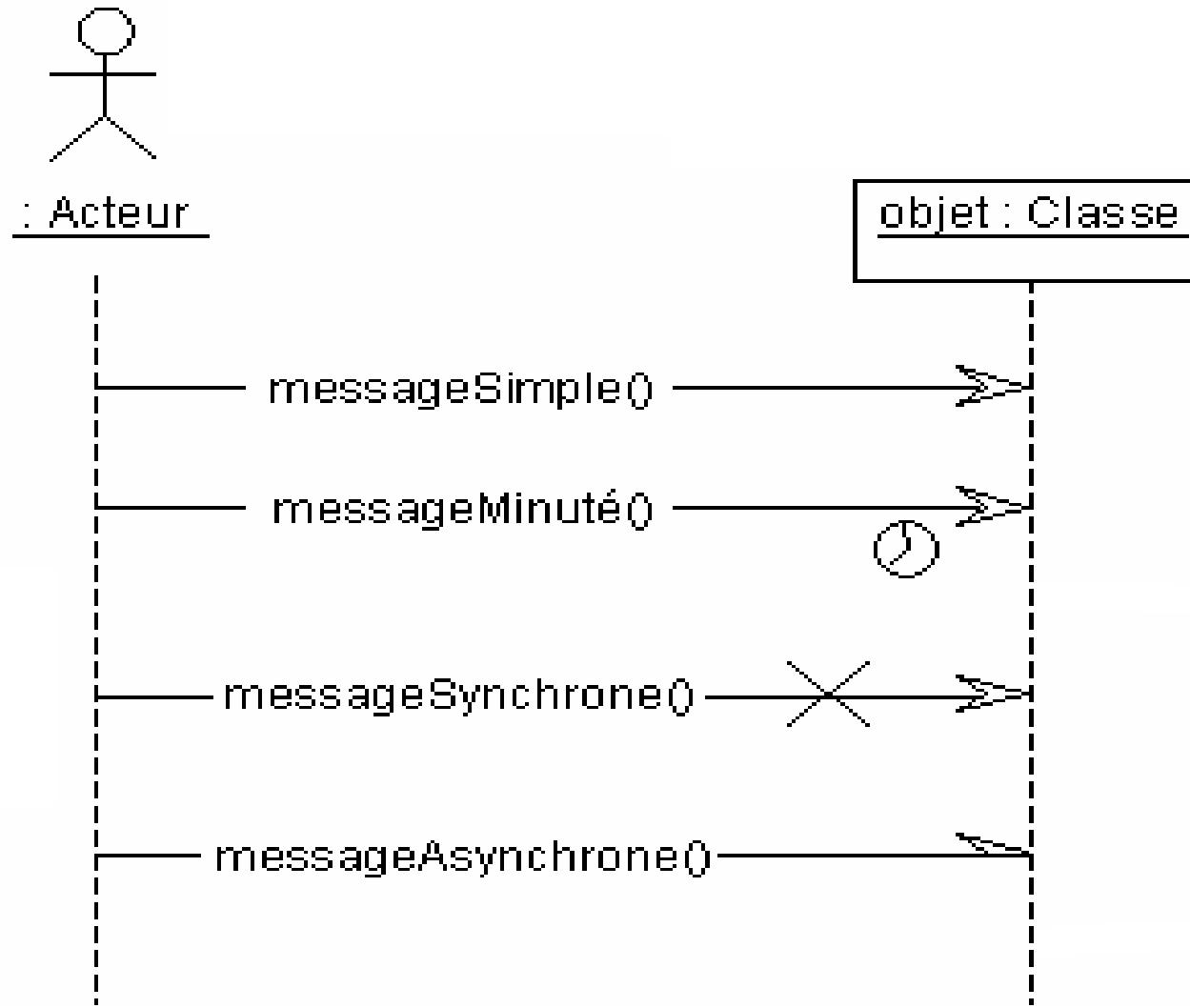


Description

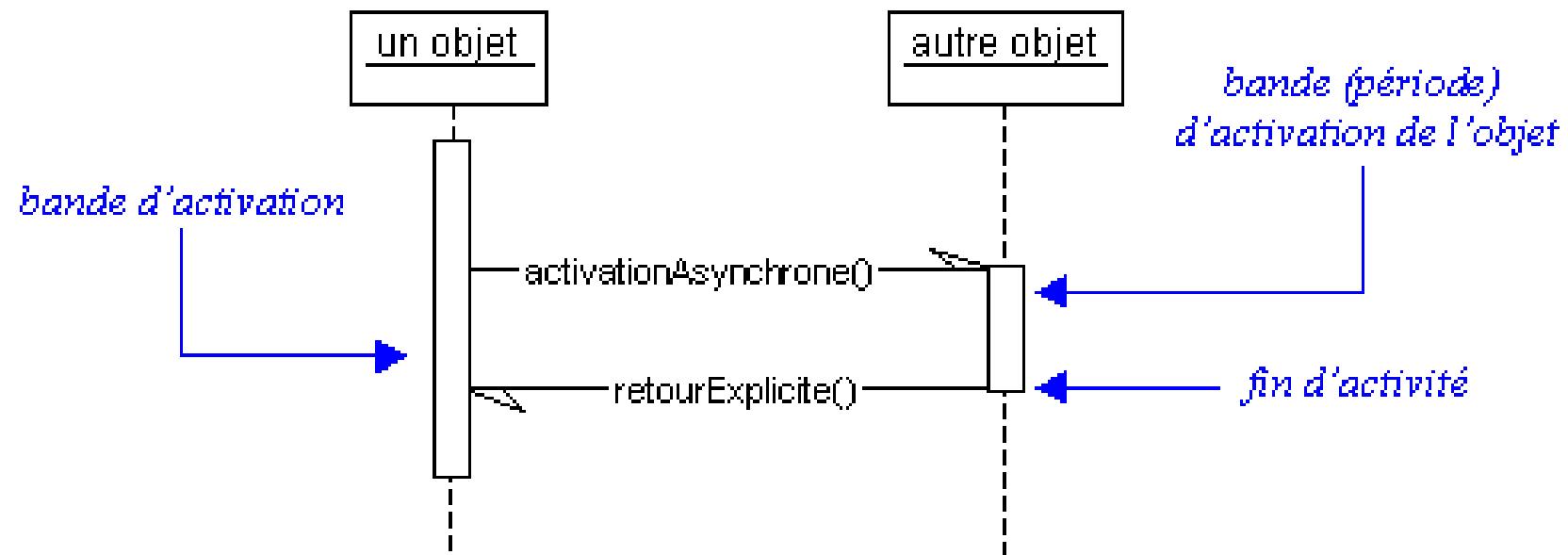
- Modélisation de la collaboration entre les objets au cours du temps : enchainement des opérations, messages envoyés etc.
- Eléments du diagramme : acteurs, objets et une ligne de vie sous chaque objet ou acteur.



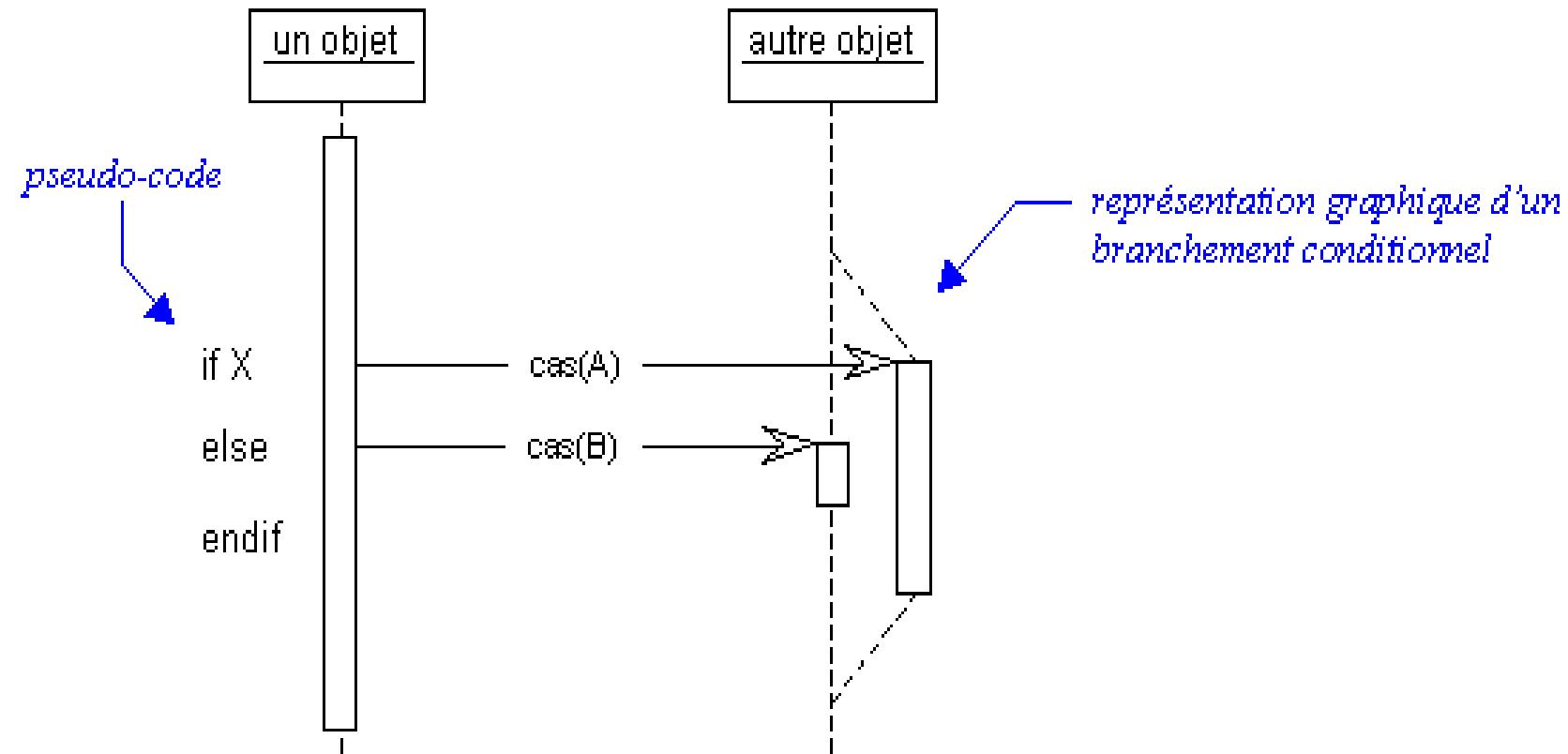
Messages



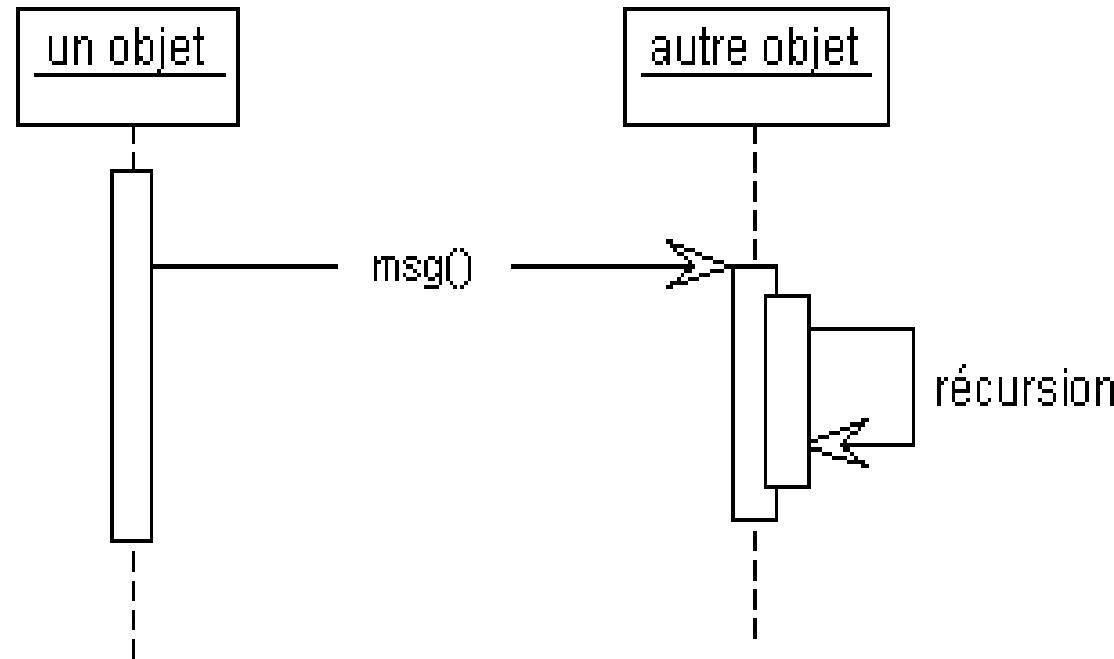
Périodes d'activités



Tests



Boucles



Exemple

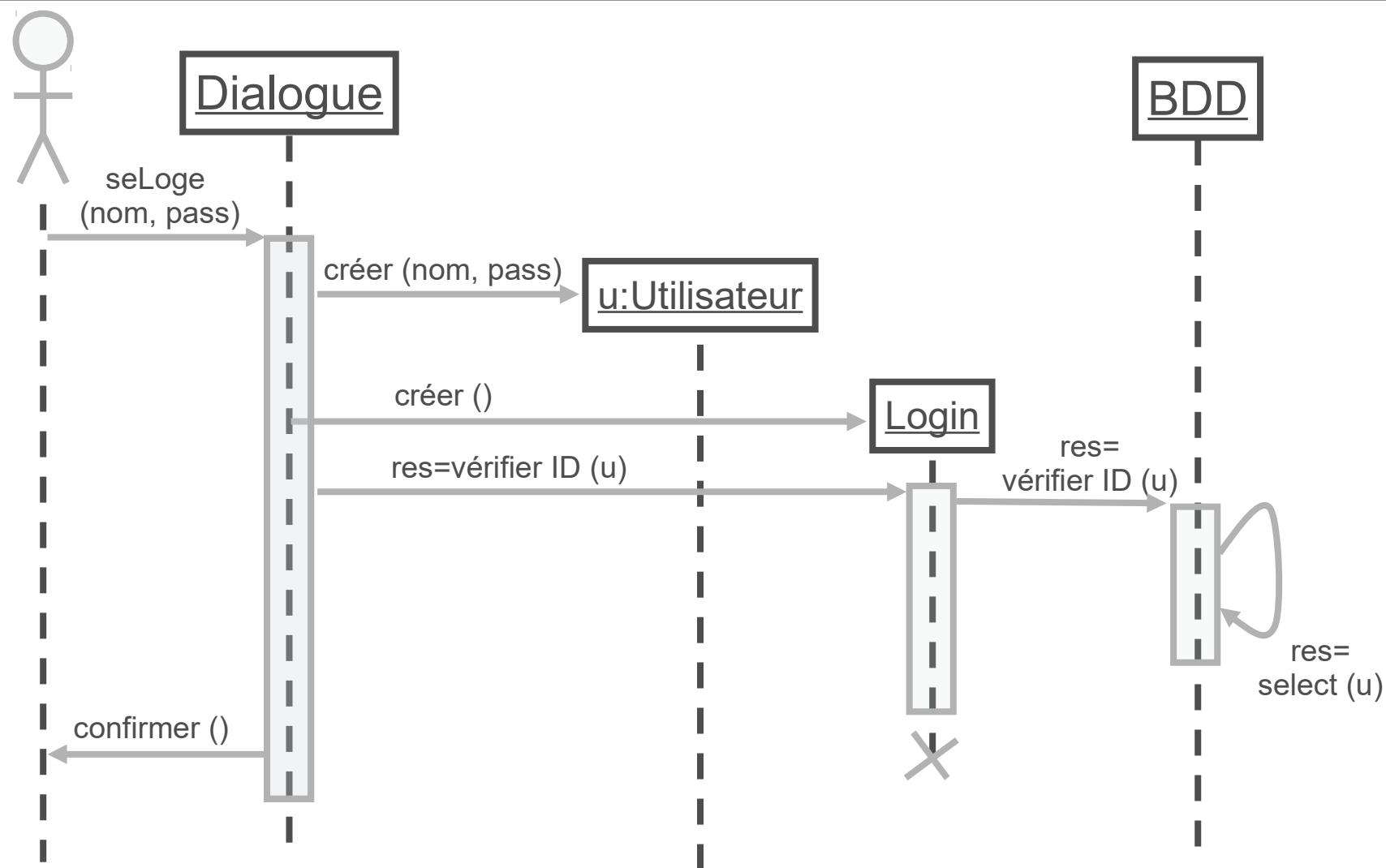
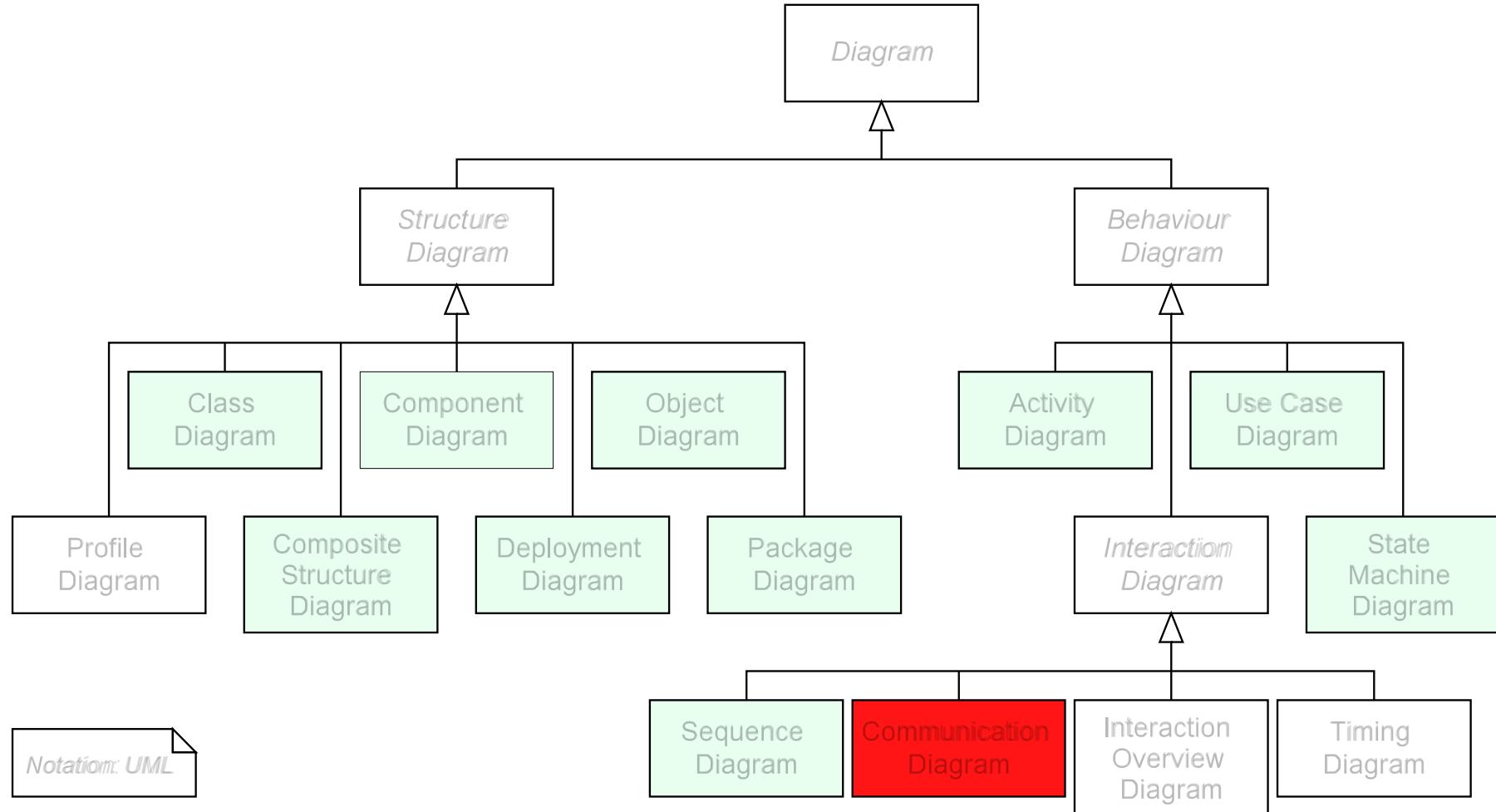


Diagramme de communication

Les diagrammes UML

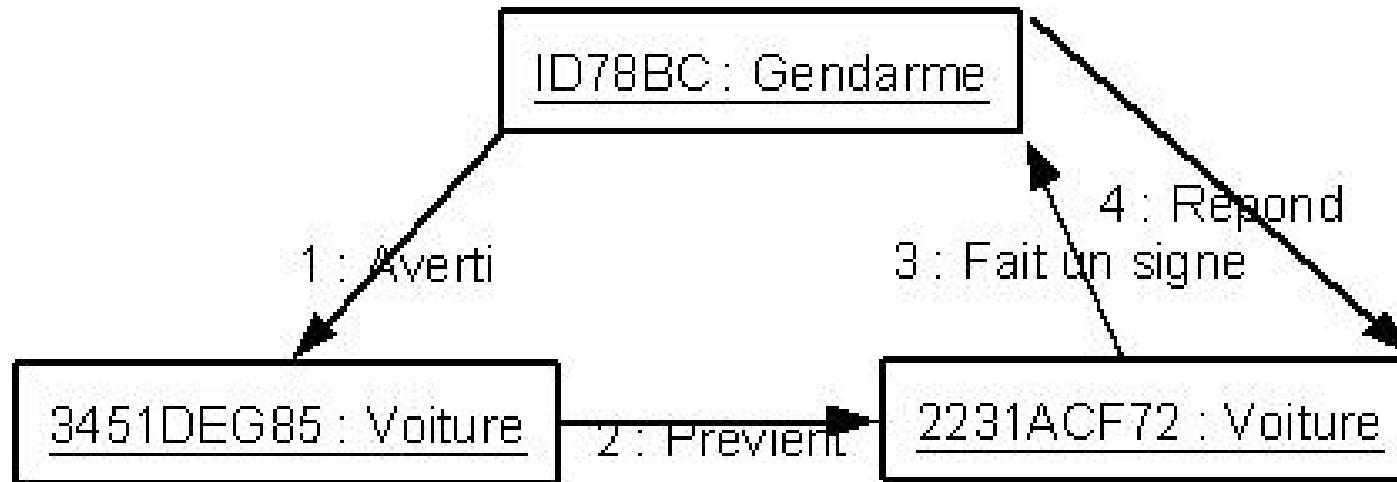


Notation: UML

Description

- Nommé Diagramme de collaboration en UML 1
- Représentation simplifiée d'un diagramme de séquence se concentrant sur les échanges de messages entre les objets.
- Organisation des acteurs aux interactions et dynamiques du système.
- Notion de messages numérotés.

Exemple



La CB

Exemple

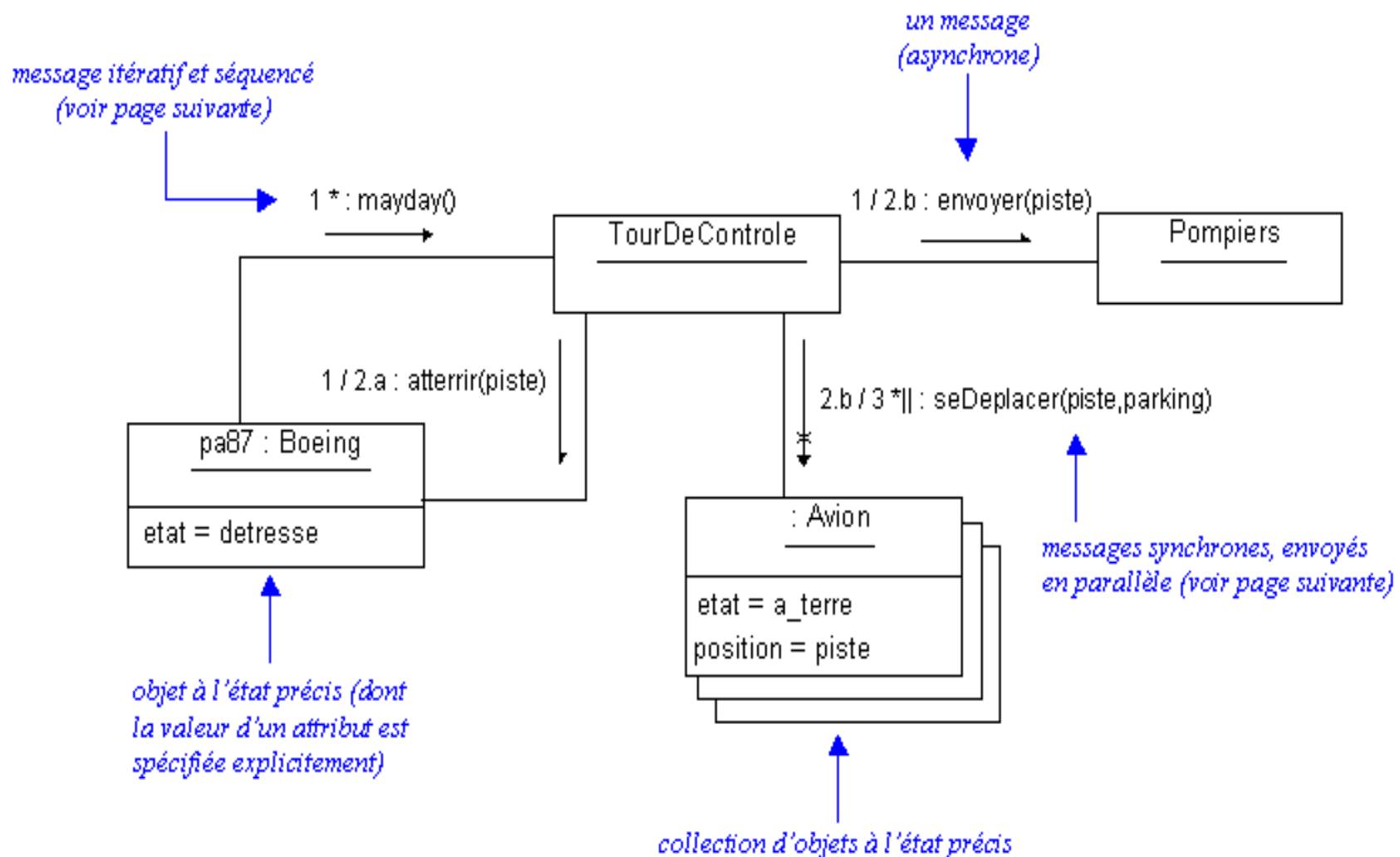
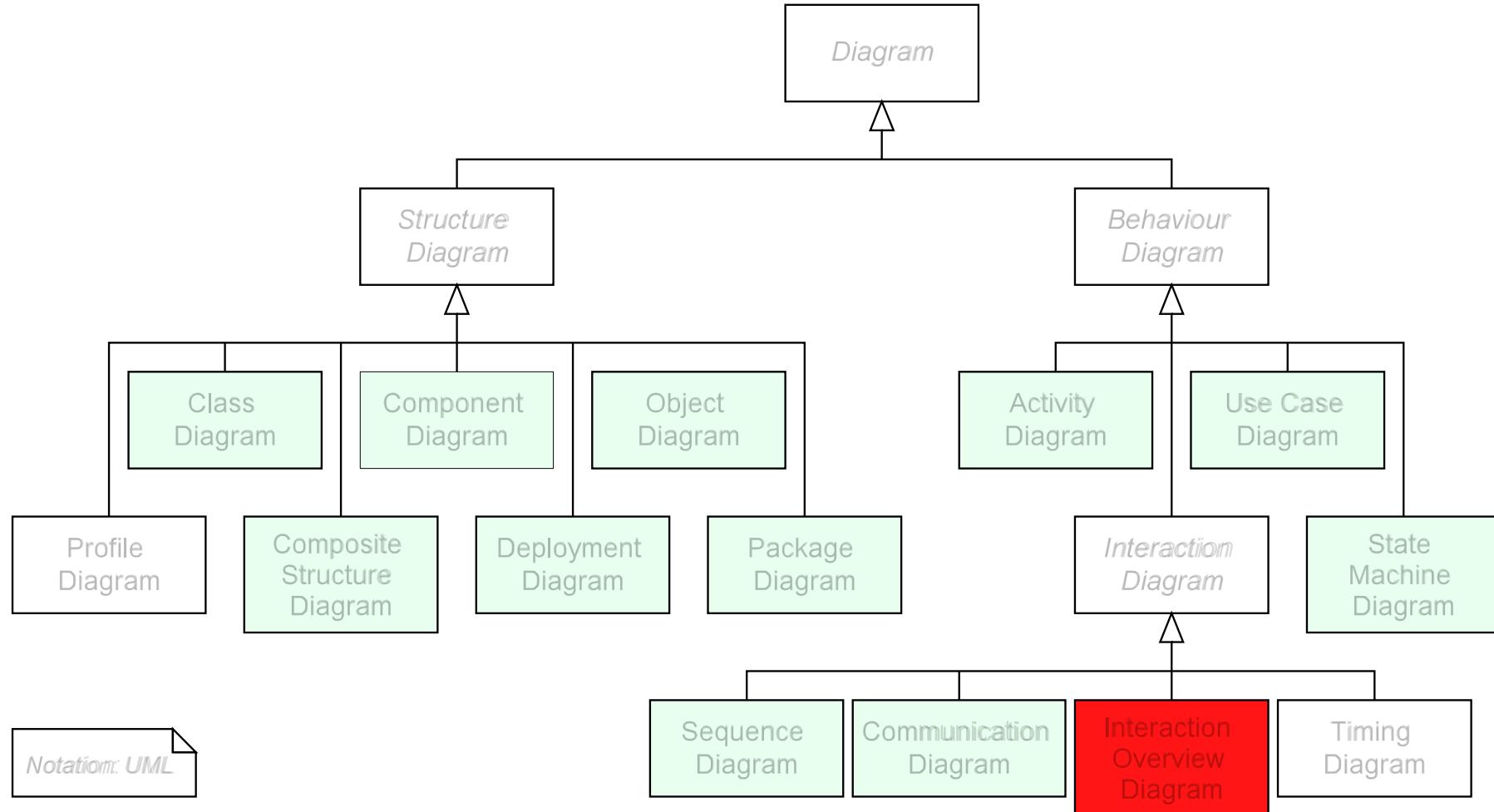


Diagramme global d'interaction

Les diagrammes UML



Description

- Vue d'ensemble du flux de contrôle où les nœuds sont des interactions ou InteractionUses.
- Utilisation des diagrammes d'activités et de séquence pour décrire comment des fragments d'interaction (décrits par des diagrammes de séquence) sont combinés par des points de décision et des flux

Exemple

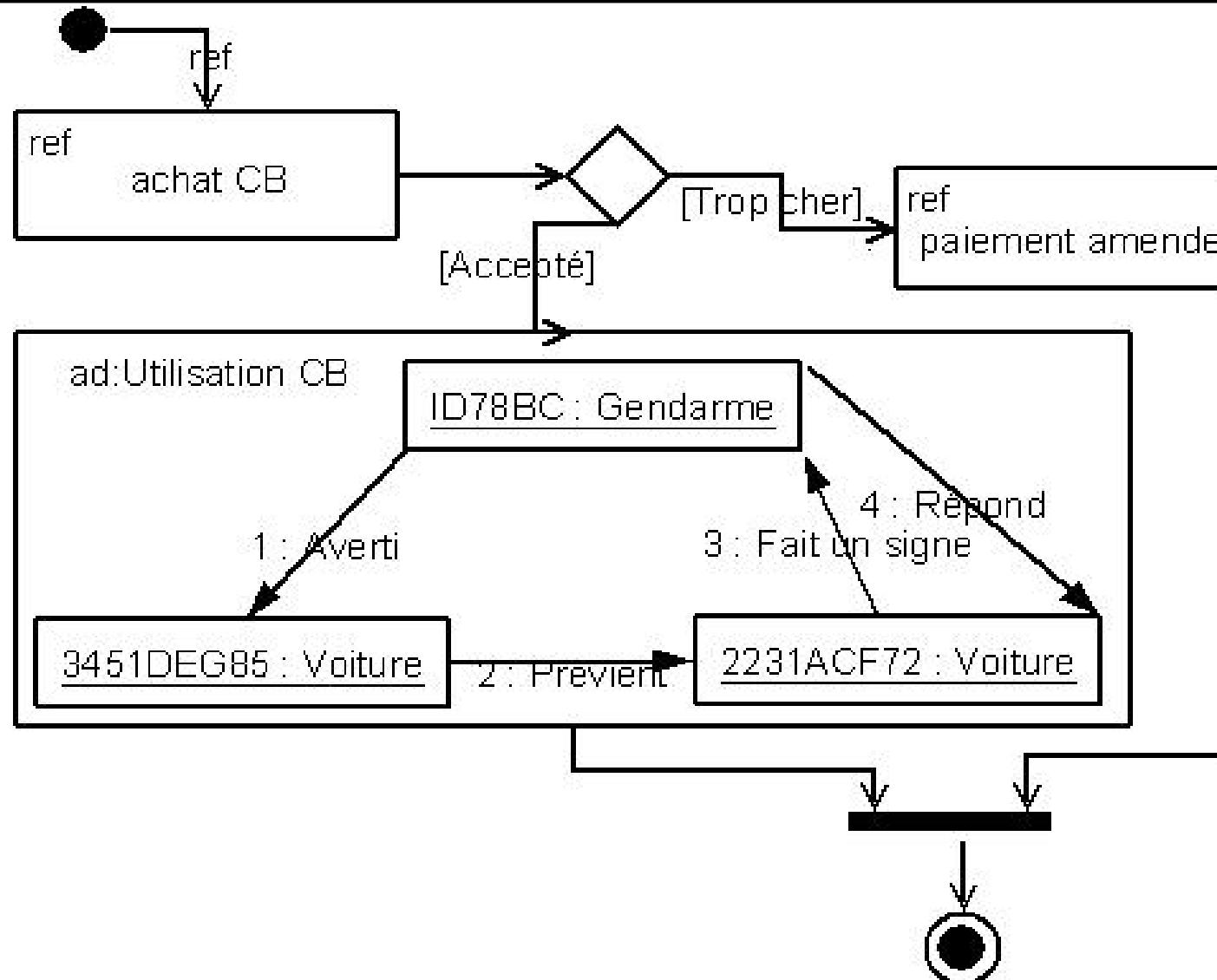
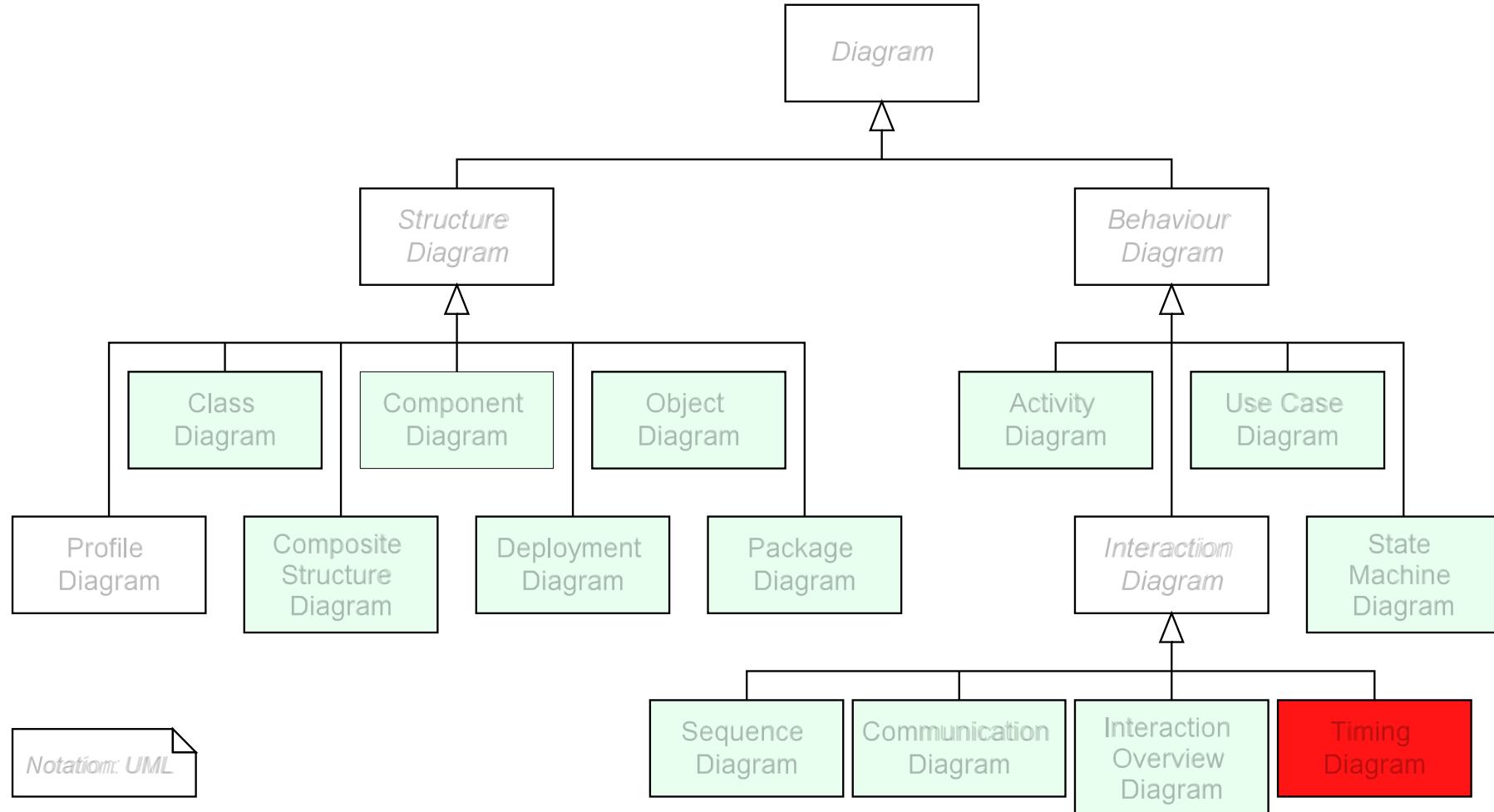


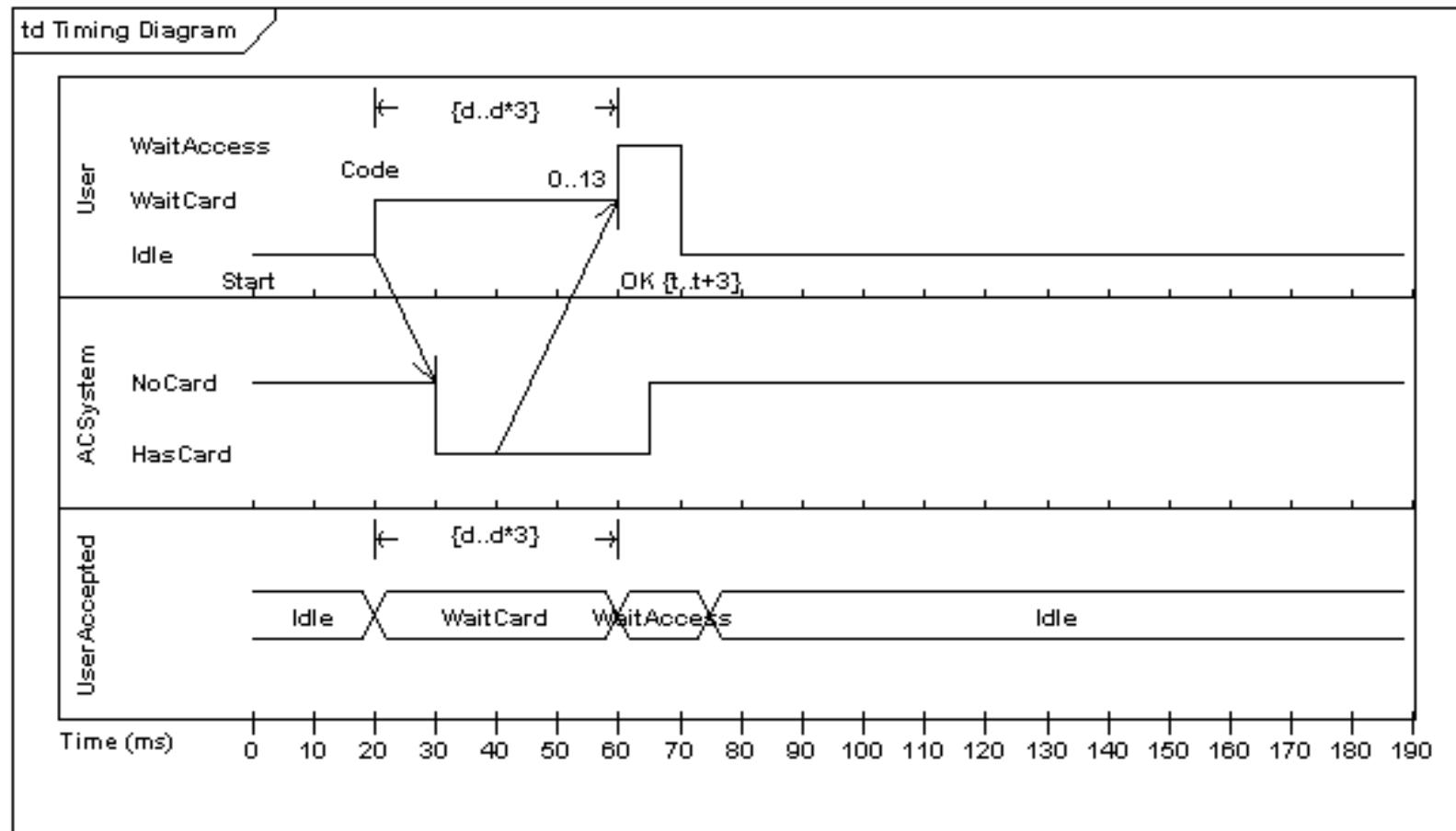
Diagramme de temps

Les diagrammes UML



Description

- Combinaison des diagrammes de séquence et d'état pour montrer l'évolution de l'état d'un objet au fil du temps, et sur les messages qui modifient cet état.



Quel diagramme utiliser ?

Les points de vue de modélisation

Il existe donc de nombreux types de diagrammes.

La question est donc de savoir quels diagrammes sont utiles pour qui et pour quoi.

Il est donc nécessaire de définir les différents points de vue.

Le points de vue de la spécification fonctionnelle

Il concerne l'organisation du modèle des besoins fonctionnels exprimés par les utilisateurs et étudiés par les analystes ou concepteurs.

Le point de vue structurel

Il a trait à l'organisation du modèle des besoins élaboré en classes par les analystes ou concepteurs.

Le point de vue matériel

Il présente la structure physique des machines et des réseaux sur lesquels repose le système informatique.

Il peut concerner les ingénieurs ou intégrateurs système et réseau.

Le point de vue de déploiement

Il représente la structure des composants matériels et localise les composants logiciels sur le réseau physique.

Il concerne les personnes chargées d'installer le logiciel ou d'identifier les causes de pannes.

Le point de vue d'exploitation

Il correspond à l'organisation des composants et identifie les fonctions prises en charge par le logiciel installé.

Il concerne à la fois les ingénieurs d'exploitation et les concepteurs, les uns pour trouver le composant incriminé par une panne, les autres pour cartographier les dépendances logicielles.

Le point de vue des spécifications logicielles



Il concerne les architectes ou concepteurs qui décident de répartir par couches les exigences techniques.

Le point de vue logique

Il est relatif à l'organisation du modèle de solution élaboré en classes par les concepteurs.

Le point de configuration logicielle



Il retrace enfin la manière dont sont construits les composants qui servent à l'élaboration d'un sous-système.

Il permet de mesurer l'impact d'un changement sur la construction du logiciel et d'établir les configurations et les compatibilités entre plusieurs éléments de versions différentes.

Représentation par point de vue

Diagramme UML	Point de vue								
	Spécification fonctionnelle	Structuré	Matériel	Déploiement	Exploitation	Spécification logicielle	Logique	Configuration logicielle	
Classes	O	I				O	I		
Objets		O					O		
Cas d'utilisation	I					I			
Séquence	I					O	I		
Collaboration	I					O	I		
États-transitions		I					I		
Activités	I					I	O		
Composants				I	I				I
Déploiement			I	I					

O : optionnel

I : indispensable

Concepts avancés

Démarche de développement

Introduction

- **Processus classique :**
 - Capturer les exigences
 - Analyse : étudier le problème
 - Conception : solutionner
 - Réalisation : coder
 - Tests
- Les utilisateurs ne savent pas ce qu'ils veulent
- Les besoins évoluent
- Les règles métiers changent
- Le marché change en permanence
- Les technologies évoluent

Introduction (suite)

- L'évolutivité, la maintenabilité et la réutilisabilité impliquent un développement itératif et incrémental
- 1 itération = analyse + conception + codage + test + intégration

==> Méthode UP

Unified Process

- UP est un ensemble d'étapes et de bonnes pratiques dont l'objectif est de produire un logiciel
- Développement itératif
- Basé sur l'architecture
- Piloté par les risques
- Prise en compte des exigences et de la qualité
- Modélisation visuelle avec UML
- Orienté objet

Cycle de développement UP

1. Inception (5%)

- ◆ Recueillir les principales exigences (20%)
- ◆ Identifier une architecture candidate
- ◆ Planifier et estimer la 1ère itération de l'Elaboration

2. Elaboration (20%)

- ◆ Réaliser les 20% d'exigences - Capturer 80% des exigences restantes - Valider l'architecture

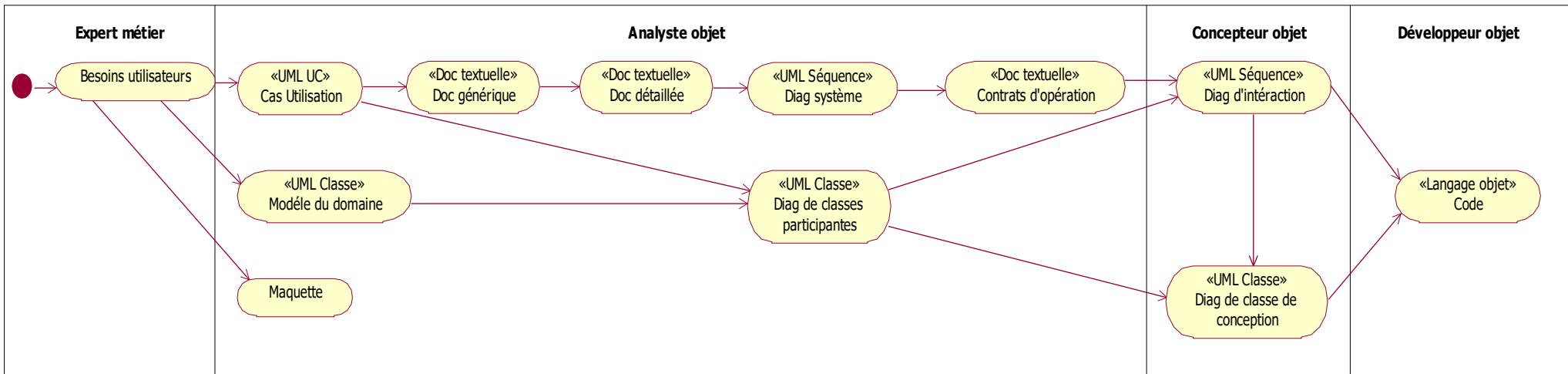
3. Construction (65%)

- ◆ Réaliser les 80% d'exigences
- ◆ Préparer le système à subir les tests

4. Transition (10%)

- ◆ Effectuer les tests - Convertir et reprendre les données existantes - Former les utilisateurs - Transfert de compétences - Déployer

Démarche globale de modélisation





Plus d'informations sur <http://www.dawan.fr>
Contactez notre service commercial au **0810.001.917** (prix d'un appel local)

DAWAN Paris, 11, rue Antoine Bourdelle, 75015 PARIS

DAWAN Nantes, 28, rue de Strasbourg, 44000 NANTES

DAWAN Lyon, Batiment de la banque Rhône Alpes, 2ème étage, montée B - 235, cours Lafayette, 69006 LYON

DAWAN Lille, 16, place du Générale de Gaulle, 6ème étage, 59800 LILLE

formation@dawan.fr