

UML



Facile



Normal



Difficile



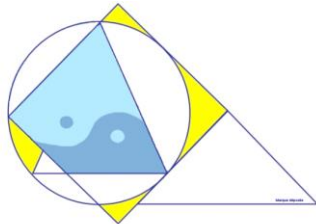
Professionnel



Expert

https://wiki.waze.com/wiki/Your_Rank_and_Points

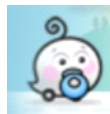
- Génie Logicielle
- Le langage UML
- Les différents diagrammes



Yantra Technologies

www.yantra-technologies.com

Génie Logiciel



Facile



Normal



Difficile



Professionnel



Expert

carole.grondein@yantra-technologies.com
david.palermo@yantra-technologies.com

https://wiki.waze.com/wiki/Your_Rank_and_Points

1.2 – Le langage UML



- Définition
- Points forts & Points faibles
- Historique
- Utilisation UML
- Caractéristiques UML
- Les diagrammes UML 2.0
- Les vues UML
- La vue logique
- La vue de réalisation
- La vue des processus
- La vue de déploiement
- La vue des cas d'utilisation



UML : Unified Modeling Language



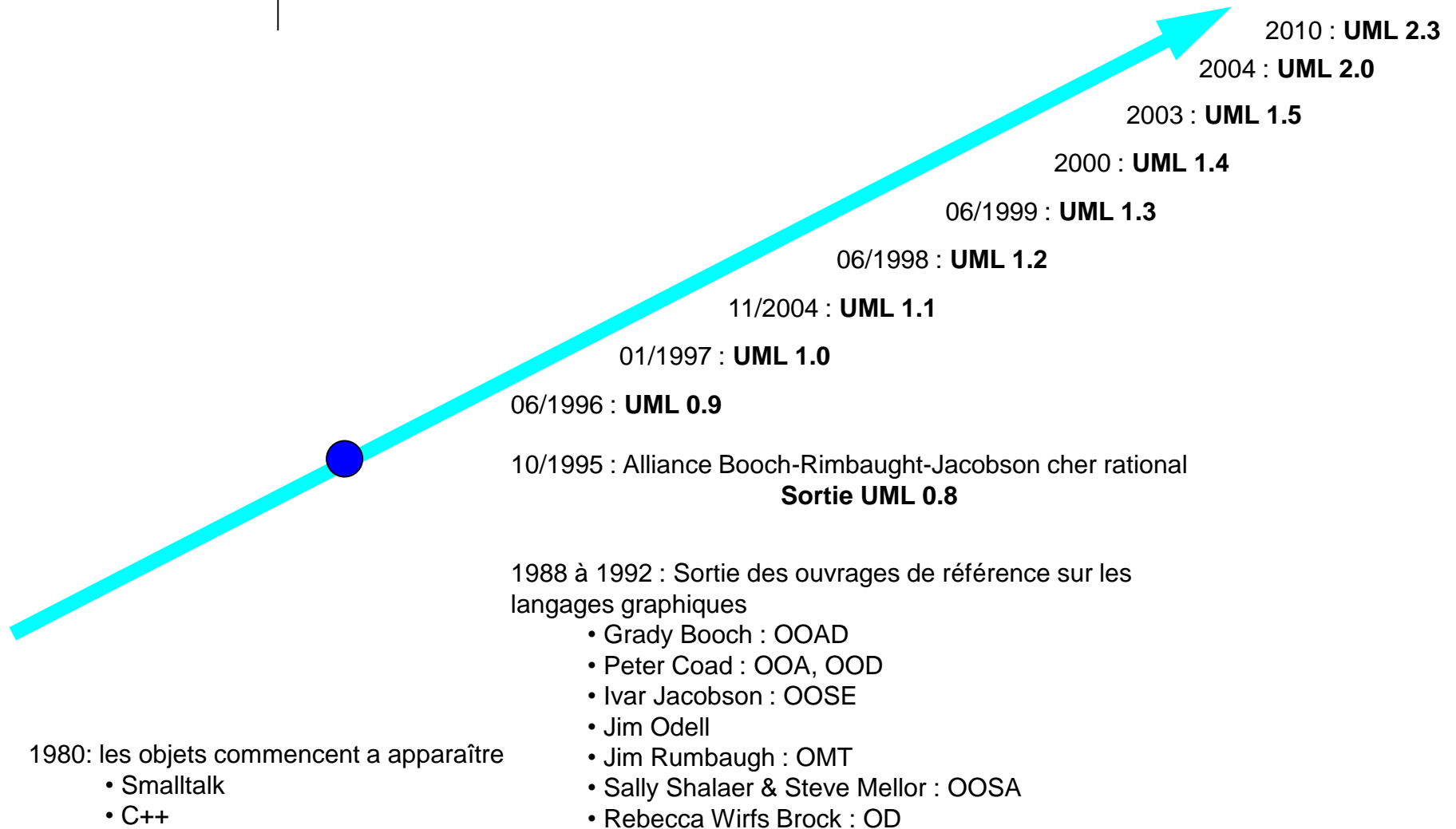
- **UML permet d'exprimer et d'élaborer des modèles objet, indépendamment de tout langage de programmation.** Il a été pensé pour servir de support à une analyse basée sur les concepts objet.
- UML est un **langage formel**, défini par un **métamodèle**.
- Le métamodèle d'UML décrit de manière très précise tous les éléments de modélisation et la sémantique de ces éléments (leur définition et le sens de leur utilisation).
UML normalise les concepts objet.

UML est avant tout **un support de communication performant**, qui facilite la représentation et la compréhension de solutions objet



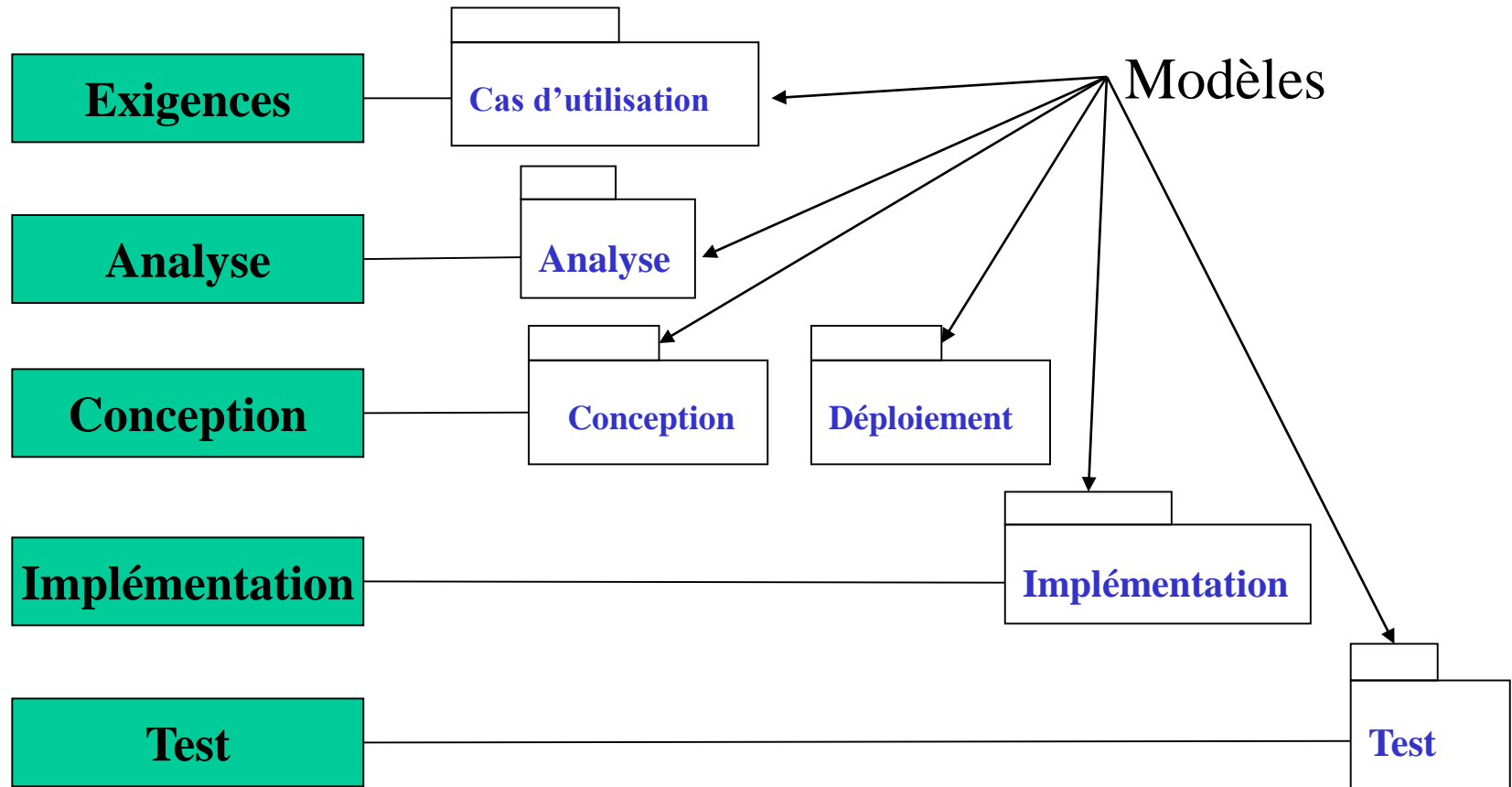
- Les points forts d'UML
 - UML est un langage formel et normalisé
 - UML est un support de communication performant
- Les points faibles d'UML
 - La mise en pratique d'UML nécessite un apprentissage et passe par une période d'adaptation.
 - Le processus n'est pas couvert par UML

1.2.3 - Historique





Modélisation UML



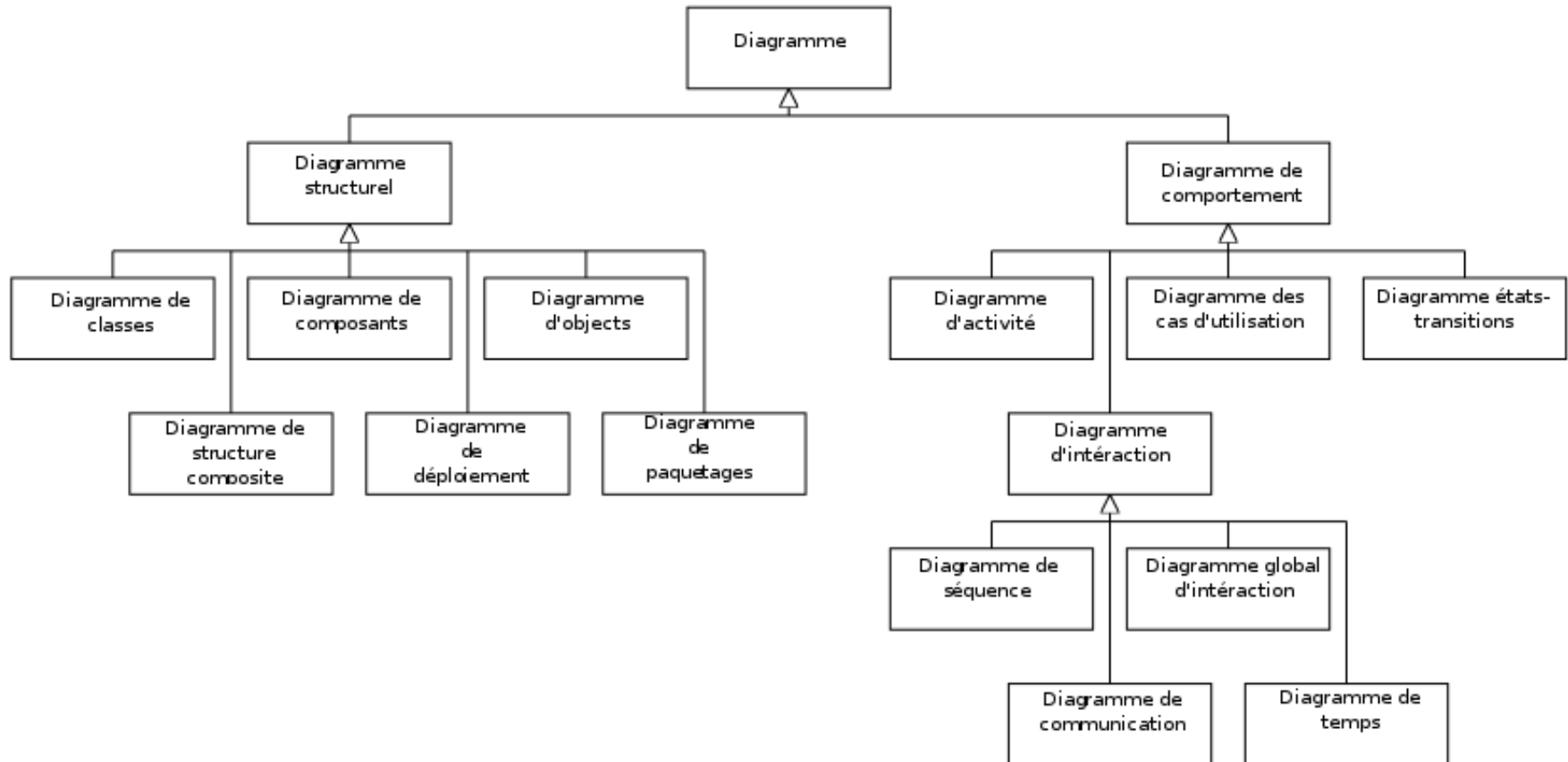
1.2.5 - Caractéristiques UML



Notation	Graphique, Textuelle, Semi-formelle
auto-décrite	méta-modélisation : modélisation récursive des éléments de modélisation eux-mêmes
le paquetage	<p>Organisation du modèle, Espace de noms</p> <p>vue logique \Rightarrow catégorie</p> <p>vue de réalisation \Rightarrow sous-système</p> <p>liens de dépendance entre paquetages</p> <p>\Rightarrow importation = utilisation</p> <p>\Rightarrow inclusion</p>
les diagrammes	13 diagrammes + 1 langage
la démarche UML	<p>UML ne propose pas de démarche, et RECOMMANDE :</p> <p>un processus piloté par les cas d'utilisation, centré sur l'architecture, selon une démarche itérative, et incrémentale, privilégiant la réduction du risque comme objectif de gestion de projet.</p>



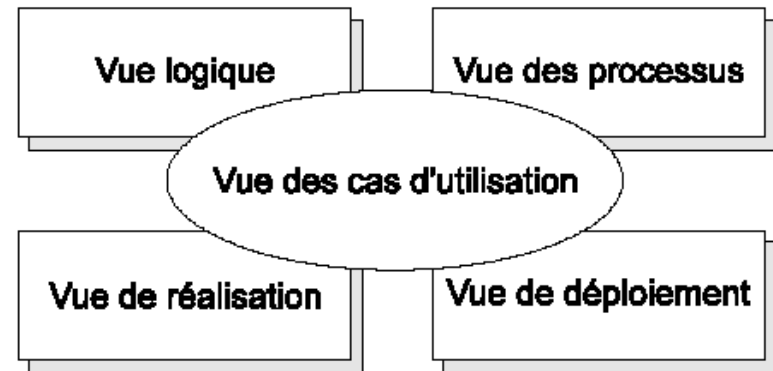
Modélisation UML 2.0





4 vues + 1 :

- Vue logique,
- Vue de réalisation,
- Vue des processus,
- Vue de déploiement,
- *Vue des cas d'utilisation.*





Objectif : analyser le problème (fonctionnel), et le résoudre formellement

La **vue logique** :

- Modélise les éléments et mécanismes principaux du système en se concentrant sur les abstractions et l'encapsulation,
- Identifie les éléments du domaine (métier, savoir-faire), ainsi que les relations et interactions entre ces éléments,
- Organise les éléments du domaine en catégories.

Les éléments :

- Les objets
- Les classes
- Les collaborations
- Les interactions
- Les paquetages <<Catégorie>>



Objectif : décrire la solution logicielle à mettre en œuvre

La vue de réalisation

- Montre l'allocation des éléments de modélisation dans des modules (fichiers sources, bibliothèques dynamiques, BDD, exécutables, etc...),
- Identifie les modules qui réalisent (physiquement) les classes de la vue logique,
- Organise les composants (distribution du code en gestion de configuration, les dépendances entre les composants...) et les contraintes de développement (bibliothèques externes...),
- Montre l'organisation des modules en sous-systèmes et leur interfaces.

Les éléments :

- Les modules
- Les sous-programmes
- Les tâches
- Les paquetages <<sous-système>>



Objectif : décrire le fonctionnement en dynamique de la solution

La **vue des processus** montre :

- La décomposition du système en terme de processus
- Les interactions entre les processus (leur communication).
- La synchronisation et la communication des activités parallèles.

Les éléments :

- Les tâches
- Les threads
- Les processus
- Les interactions



Objectif : décrire la projection du logiciel sur le matériel

La **vue de déploiement** décrit les ressources matérielles et la répartition du logiciel dans ces ressources :

- La disposition et nature physique des matériels, ainsi que leurs performances
- L'implantation des modules principaux sur les noeuds du réseau
- Les exigences en terme de performances

Les éléments:

- Les noeuds
- Les modules
- Les programmes principaux



Objectif : décrire le besoin (fonctionnel).

La vue des cas d'utilisation :

- Unifie les quatre autres vues de l'architecture,
- Définit les besoins des clients du système et centre la définition de l'architecture du système sur la satisfaction et la réalisation de ces besoins,
- Conduit à la définition d'un modèle d'architecture à l'aide de scénarios et de cas d'utilisation,
- Motive les choix et permet d'identifier les interfaces critiques et se concentrer sur les problèmes importants.

Les éléments:

- Les acteurs
- Les cas d'utilisation
- Les classes
- Les collaborations

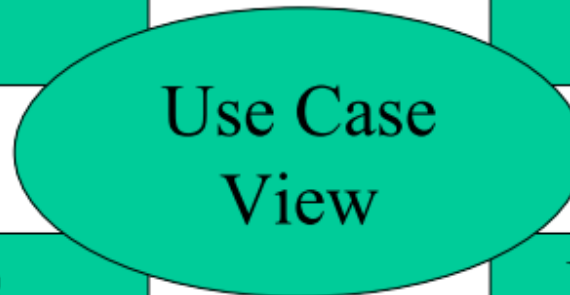
1.2.13 – Résumer des vues



- Classes
 - Interfaces
 - Collaboration
- => Les services du systèmes



- Composant
 - Fichiers Source
- => Configuration du système



=> Comportement du système



- Thread
 - Process
 - Concurrence
 - Synchronisation
- => Performance du système



- Architecture
 - Hardware
 - Distribution
- => Topologie du système

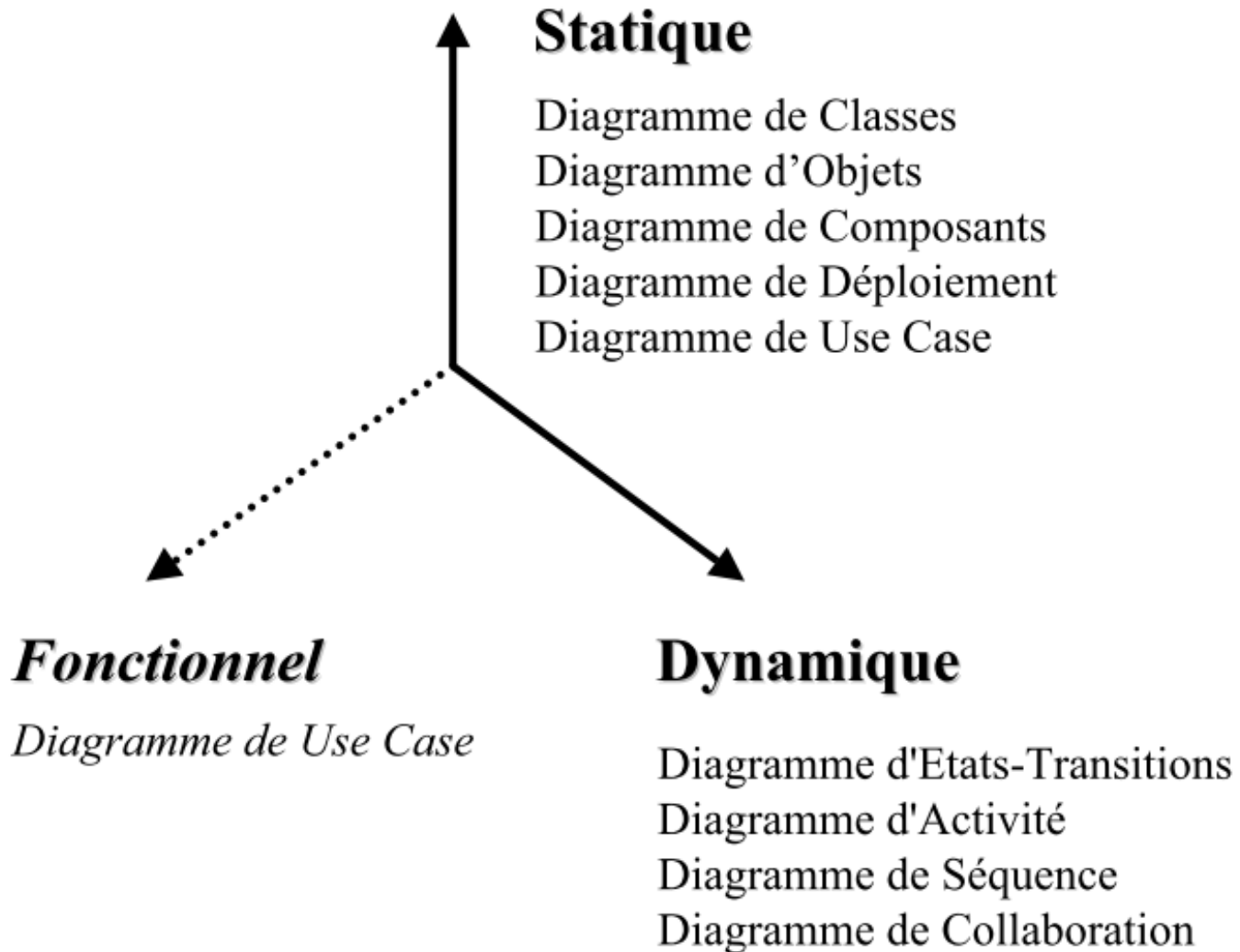
1.2.13 – Résumer des vues



	Vue des cas d'utilisation	Vue logique	Vue de réalisation	Vue des processus	Vue de déploiement
Diagramme de cas d'utilisation	Acteurs Cas d'utilisation				
Diagramme de classes		Classes Relations			
Diagramme d'objets	Objets Liens	Classes Objets Liens			
Diagramme de séquence	Acteurs Objets Messages	Acteurs Objets Messages		Objets Messages	
Diagramme de collaboration	Acteurs Objets Liens Message	Acteurs Objets Liens Messages		Objets Liens Messages	



1.3 - Trois Axes de Modélisation





- 1 - UML pour les décideurs
- 2 - UML pour les concepteurs
- 3 - UML pour les développeurs



- Capture initiale des besoins
 - Cahier des charges préliminaires
- Capture des besoins fonctionnels
 - Cahier des charges
- Analyse
 - Expression des besoins
 - Architecture logique



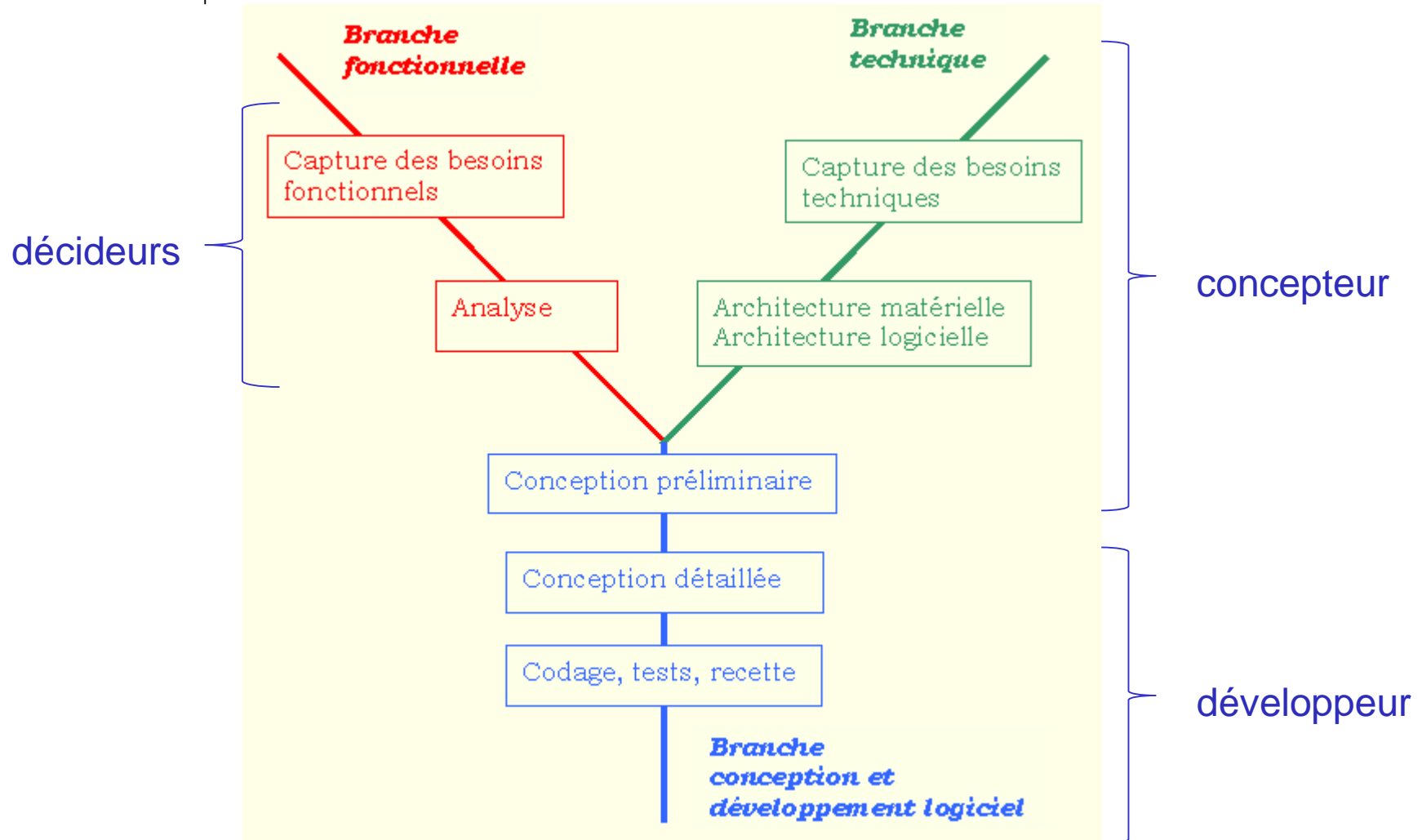
- Capture des besoins techniques
- Conception générique
 - Prototype
 - Générateur de code
 - Design Patterns
- Conception préliminaire
 - Interface
 - Design Patterns
- Architecture Technique

1.4.3 - UML pour les développeurs

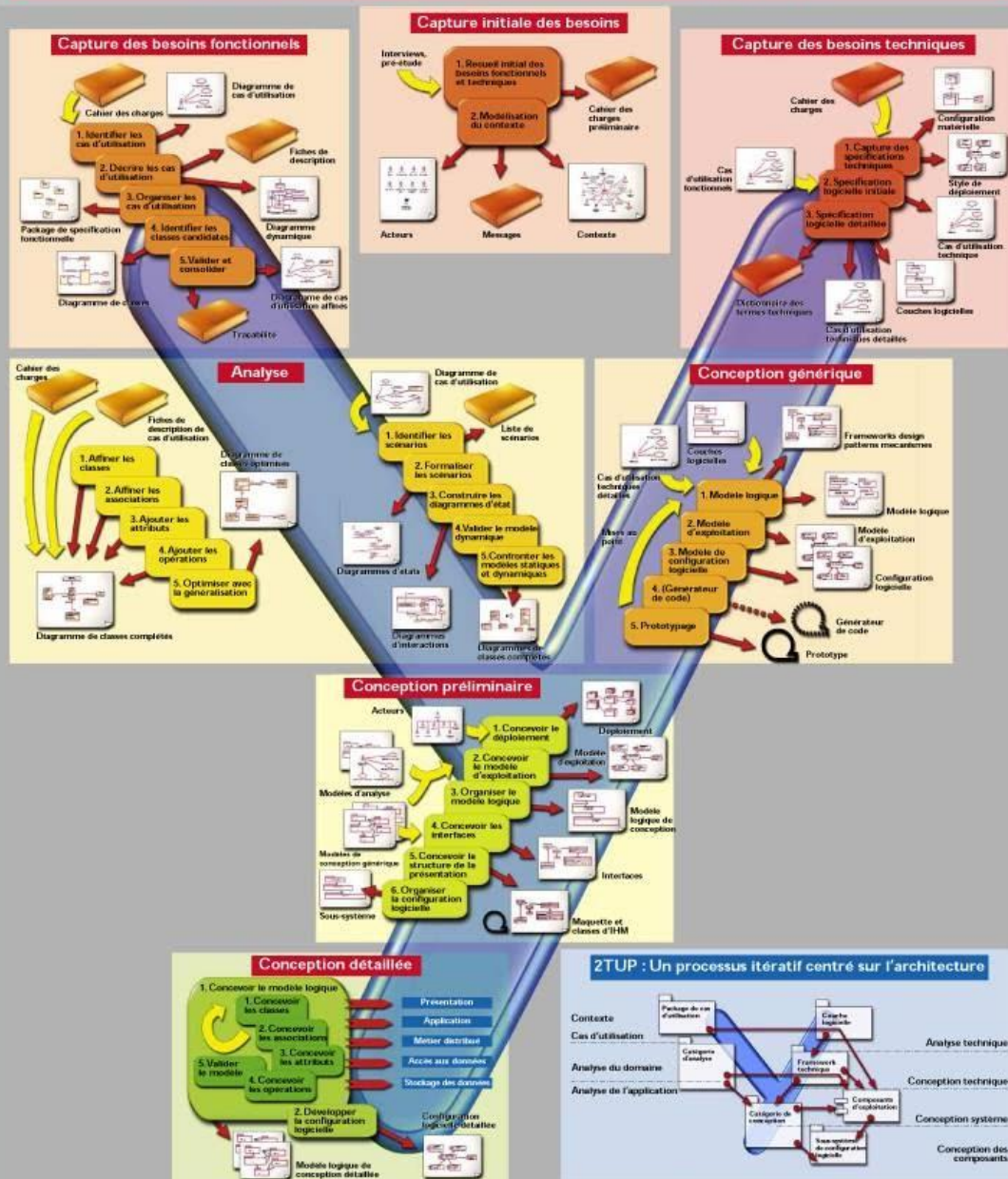


- Conception détaillée
- Générateur de code
- Retro ingénierie

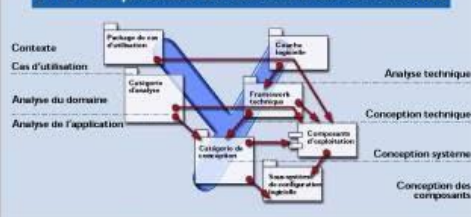
1.5 - Two Track Unified Process



Two Track Unified Process



2TUP : Un processus itératif centré sur l'architecture





SysML est un langage de modélisation spécifique au domaine de l'ingénierie système.

Il permet la spécification, l'analyse, la conception, la vérification et la validation de nombreux systèmes et systèmes-de-systèmes.

SysML se définit comme une extension d'un sous-ensemble d'UML (Unified Modeling Language) via l'utilisation du mécanisme de profil défini par UML.

1.6 - Pourquoi SysML

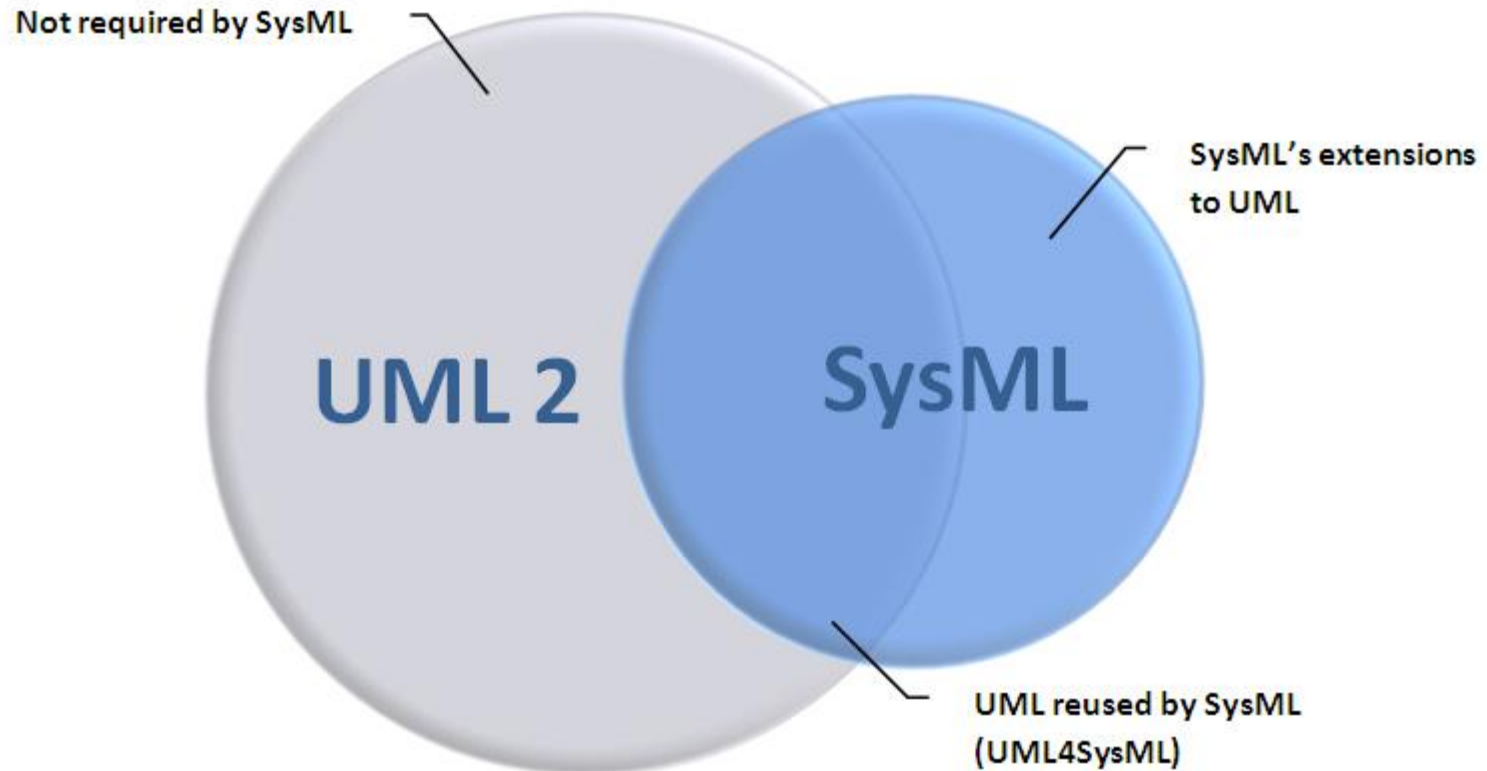


le langage UML permet par son caractère à usage général d'adresser de nombreux besoins pour l'IS mais il ne répond pas à tous les besoins.

SysML apporte une simplification et une standardisation du vocabulaire, plus question ici de classes, d'objets, ou d'héritage.

Ce langage, ajoute la possibilité de représenter les exigences du système comme elles sont définies dans un cahier des charges, les éléments non-logiciels (mécanique, hydraulique, capteur...), les équations physiques, les flux continus (matière, énergie, etc.) et les allocations.

1.6 - UML et SysML



1.6 - Comparatif des diagrammes UML et SysML



SysML	Description	UML
Use Case diagram	Identique en UML et en SysML, il modélise les fonctionnalités que le système doit fournir. Le cas d'utilisation est une unité fonctionnelle utilisée pour la description et la recette du système.	Use Case diagram
Sequence diagram	Identique en UML et en SysML le diagramme de séquence modélise la chronologie des interactions entre les éléments du système ou entre le système et l'extérieur.	Sequence diagram
Activity diagram	Même utilisation en UML et en SysML. Le diagramme d'activité modélise les flux d'informations et les flux d'activité du système.	Activity diagram
State Machine diagram	Identique en UML et en SysML, il représente les différents états que peut prendre un élément ou une opération ainsi que ses réactions aux événements extérieurs.	State Machine diagram
Block Definition diagram	Le diagramme de Bloc en SysML est semblable au diagramme de Classe en UML. Il donne une représentation statique des entités du système, de leurs propriétés, de leurs opérations et de leurs opérations.	Class diagram
Internal Block diagram	Le diagramme interne de bloc SysML et le diagramme composite UML donnent une représentation « Boîte blanche » qui matérialise les imbrications des parties et leurs interconnexions par les ports.	Composite Structure diagram
Package diagram	Le diagramme de Package montre l'organisation générale du modèle en UML comme en SysML. En SysML il sert en plus à donner différentes vues du système.	Package diagram
Parametric diagram	Nouveau dans SysML ce diagramme modélise les paramètres physiques du système. Il sert à tester les performances physiques et quantitatives du système.	N/A
Requirement diagram	Le diagramme de spécification est nouveau dans SysML et il permet de collecter et d'organiser toutes les exigences textuelles du système.	N/A
Allocation tables	Nouveau en SysML. Les tables d'allocation sont de simples tableaux et non des diagrammes qui récapitulent les spécifications afin de faciliter le suivi de projet.	N/A
Component diagram, Communication diagram, Deployment diagram, Interaction diagram, Overview diagram, Object diagram et Timing diagram : n'existent pas en SysML.		















Le langage SysML a été intégré dans de nombreux outils AGL, commerciaux ou open source:

- Sparx Systems Enterprise Architect (plugin SysML ou version Ultimate requise)
- IBM Rational Software Modeler (plugin d'une société tierce disponible)
- Magicdraw (plugin SysML requis)
- Open source : Topcased (environnement Eclipse)

1.7 – Outils UML



 <p>Windows, Linux, Solaris, Mac OS X</p>	 <p>Enterprise Architect Version 7.0 <small>© 1998 - 2007 Sparx Systems. All rights reserved.</small></p> <p>Enterprise Architect Sparx Systems</p>	 <p>Argo UML</p>	 <p>IBM : Rational Rose Le Leader Mondial</p>
 <p>I-Logix Rhapsody Modeler</p>	 <p>Objecteering Software Objecteering/UML</p>	 <p>gentleware just model Poseidon for UML</p>	 <p>Python UML Tool</p>
 <p>ModelMaker Un extraordinaire outil UML pour Delphi Windows</p>	 <p>Papyrus</p>	 <p>Visual Paradigm Visual Paradigm for UML Windows, OSX, Linux</p>	 <p>BOUML</p>

<https://uml.developpez.com/telecharger/>

http://www.objectsbydesign.com/tools/umltools_byPrice.html


http://fr.wikipedia.org/wiki/Comparaison_des_logiciels_d%27UML



www.uml-sysml.org



Recherche

 Rechercher

☐ Seulement dans le dossier courant

Accueil

Modéliser

UML

SysML

Diagrammes

Documentation







Infos pratiques

[Se connecter](#) [Plan de site](#) [Accessibilité](#) [Contact](#) [Images du Site](#) [Extranet](#)

Vous êtes ici : [Accueil](#) → [Modéliser](#) → Pourquoi UML ?



Navigation

-  [Modéliser](#)
-  [Pourquoi UML ?](#)
-  [Processus de modélisation](#)
-  [L'approche Top Down](#)
-  [Rédaction du rapport](#)
-  [Evolution d'un projet](#)

Pourquoi UML ?

De la même façon qu'il vaut mieux dessiner une maison avant de la construire, il vaut mieux modéliser un système avant de le réaliser.

UML pour :

- Obtenir une modélisation de très haut niveau indépendante des langages et des environnements.
- Faire collaborer des participants de tous horizons autour d'un même document de synthèse.
- Faire des simulations avant de construire un système.
- Exprimer dans un seul modèle tous les aspects statiques, dynamiques, juridiques, spécifications, etc...
- Documenter un projet.
- Générer automatiquement la partie logiciel d'un système.

Références

- Le dernier livre de Pascal Roques sur SysML : [cliquez ici](#)
- Les précédents livres de Pascal Roques sur UML 2.0 : [cliquez ici](#)
- L'excellent site de Laurent Audibert : [cliquez ici](#)
- Le tutoriel SysML de IncoSe : [cliquez ici](#)

SysML France

SysML France :

OPTIONcarriere

Emploi recherché :

modélisation uml

Localité :

France

Recherche d'emploi > Offres d'emploi France > [Emploi modélisation uml](#)

Offres d'emploi Modélisation UML - France

1 à 20 de 464 offres d'emploi

Dernières recherches

modélisation uml, France
modélisation uml, Bouches-du-Rhône
modélisation uml, Provence-Alpes-Cote d'Azur

Tri par

- ☒ Pertinence
- ☐ Date
- ☐ Salaire

Localités

- ☒ France
- Alsace
- Aquitaine
- Bourgogne
- Bretagne
- Centre
- Franche-Comte
- Ile-de-France
- Limousin
- Midi-Pyrénées
- Nord-Pas-de-Calais
- Pays de Loire
- Provence-Alpes-Cote d'Azur
- Rhône-Alpes

Type de contrat

- ☒ Tous
- Temps-partiel
- Temps-plein
- CDI
- Contrat
- CDD
- Stage
- Volontariat

n Analyste fonctionnel pour Modélisation spécifications fonctionnelles/ UML dans le domaine Ferroviaire

Smarteo - Paris - Rennes, Ile-et-Vilaine

: **Modélisation** spécifications fonctionnelles- **UML** - IMPERATIF Critère 2 : Formalisation d'expression de besoin dans le domaine industriel - IMPERATIF...Description : ASAP! Smarteo recherche pour l'un de ses clients Un Analyste fonctionnel pour **Modélisation** spécifications fonctionnelles...

www.freelance-info.fr - 22 décembre - [Enregistrer](#) - [Envoyer à un ami](#)

ARCHITECTE LOGICIEL EMBARQUE MODELISATION UML, domaine automobile F/H

Ferchau Engineering - Toulouse

Pour accompagner le développement de notre site de Toulouse, nous recrutons en CDI un(e) : ARCHITECTE LOGICIEL EMBARQUE MODELISATION UML... de la **modélisation** de l'architecture logicielle en langage **UML**. A ce titre, vous serez amené(e) à : concevoir et documenter l'architecture et la structure...

www.apec.fr - 16 décembre - [Enregistrer](#) - [Envoyer à un ami](#)

Consultant Business Analyse / UML

Coriom Conseil - Lille, Nord

une démarche de **modélisation UML** et sensibiliser les équipes projet. Compétences : - Rigueur, diplomatie, pédagogie, curiosité, capacité d'écoute... organisations métiers, recherche pour un de ses clients de la métropole lilloise un Consultant Business Analyse / **UML**. Mission : Le consultant...

www.freelance-info.fr - 6 janvier - [Enregistrer](#) - [Envoyer à un ami](#)

Business Analyst / Consultant UML (H/F)

Espace Freelance - Villeneuve-d'Ascq, Nord

des formations, - Amorcer une démarche de **modélisation UML** et sensibiliser les équipes projet.... / Consultant **UML** (H/F) - Accompagner les chefs de projet dans le recueil des besoin, fonctionnelles et non fonctionnelles, Participation...

www.freelance-info.fr - 4 janvier - [Enregistrer](#) - [Envoyer à un ami](#)

Analyste/Concepteur UML & Architecture

Softeam - Sophia-Antipolis, Alpes-Maritimes

recensement des exigences, la **modélisation UML** des besoins et l'analyse métier jusqu'à l'architecture des applications, le prototypage de nouvelles... une expérience significative dans les technologies objets et la **modélisation UML**, de bonnes qualités relationnelles et des compétences autour: Des...

[Postuler facilement](#)

www.softeam.fr - 24 décembre - [Enregistrer](#) - [Envoyer à un ami](#)

Développeur .NET / UML

Softeam - Sophia-Antipolis, Alpes-Maritimes

.Net, Conception objet et **Modélisation UML**, Bonne maîtrise du XML et de XSL Bon niveau d'anglais requis Nous vous formerons aux... la conception et le développement d'évolutions techniques liées à une plate-forme de **modélisation** et de génération de documentation...



quoi

UML

où

métier, mots-clés ou entreprise

Rechercher

Astuce : indiquez une ville ou un code postal dans la barre "où" afin d'afficher des résultats localisés.

Emploi UML

Emplois recommandés - 48 nouveautés

Mes recherches récentes

Qt - Marseille (13) - 3 nouveautés

Python - Pertuis (84) - 3 nouveautés

Developpeur C - Pertuis (84) - 7 nouveautés

Ingénieur Modélisation Uml - 7 nouveautés

C++ - Aubagne (13) - 11 nouveautés

» effacer les recherches

Trier par : pertinence - date

Estimation du salaire

30 000 € (902)

35 000 € (759)

40 000 € (376)

45 000 € (130)

50 000 € (55)

Type de contrat

CDI (464)

Temps plein (229)

Stage (101)

Freelance / Indépendant (46)

CDD (24)

plus »

1 lien

↑ Publiez votre CV - Postulez à plus de 60 000 emplois depuis n'importe quel appareil

Emplois 1 à 10 sur 1 049

Afficher : tous les emplois - 67 nouveaux emplois

Product Definition Analyst

Amadeus - ★★★★★ 128 avis - Nice (06)

UML modelling etc.). Design the applications that will shape the travel experience of our customers....

Sponsorisée - sauvegarder

Ingénieur algorithmes H/F

HORIBA Médical - Montpellier (34)

Vous êtes autonome en programmation C et C#, et maitrisez UML. A la pointe de l'analyse médicale, à Montpellier, *HORIBA MEDICAL*....

Postuler directement

Sponsorisée - sauvegarder

Ingénieur Architecture Logiciel Embarqué (H/F) – [AUTOSAR,...

Renault - ★★★★★ 2 637 avis - Guyancourt (78)

Votre environnement de travail Le monde automobile évolue rapidement : la voiture s'électrifie, se connecte, s'automatise. Relever ces trois défis passe par...

Sponsorisée - sauvegarder

Ingénieur développement logiciel (C++/QT) F/H - nouveau

CELAD - ★★★★★ 5 avis - Val-de-Marne

Modélisation objet, langage UML, machines à états finis (statecharts). Dans le cadre d'un projet de Réalité Augmentée, notre client développe un véhicule...

Postuler directement

✉ Simplifiez-v
les nouveau
recherche

Mon email :

☒ Recevoir égale
recommandés

Valider

Entreprise qui rec

**GROUP
RENAULT**

Renault

Le Groupe Renau
innovants, passio
votre job de dem

Chef de projet Es
(H/F)

Chef de projet Int
(H/F)

Technicien(ne) M

2 - Les différents diagrammes



- 2.1 - Le diagramme de cas d'utilisation (diagramme de comportement)
- 2.2 - Le diagramme de classe (diagramme de structure)
- 2.3 - Le diagramme d'objet (diagramme de structure)
- 2.4 - Le diagramme de package (diagramme de structure)
- 2.5 - Le diagramme de séquence (diagramme d'interaction)
- 2.6 - Le diagramme d'états-transitions (diagramme de comportement)
- 2.7 - Le diagramme d'activité (diagramme de comportement)
- 2.8 - Le diagramme de composants (diagramme de structure)
- 2.9 - Le diagramme de déploiement (diagramme de structure)
- 2.10 - Le diagramme de Communication/Collaboration (diagramme d'interaction)
- 2.11 - Le diagramme de vue global des interactions (diagramme d'interaction)
- 2.12 - Le diagramme de structure composite (diagramme de structure)
- 2.13 - Le diagramme de temps (diagramme d'interaction)
- 2.14 - Langage d'expression de contraintes sur les objets : OCL

2.1 - Le diagramme de cas d'utilisation



- Fonctionnalité
- Notation
- Mise en œuvre
- Exemple



Les cas d'utilisation (*use cases*) permettent de :

- Spécifier ce qu'il sera possible de demander de l'extérieur à l'entité ainsi représentée
- Spécifier une fonctionnalité offerte par cette même entité
- Capturer les exigences fonctionnelles selon le point de vue des utilisateurs
- Structurer les besoins des utilisateurs et les objectifs correspondants d'un système
- Centrer l'expression des exigences du système sur ses utilisateurs
- Identifier les utilisateurs du système (acteurs) et leur interaction avec le système
- Classer les acteurs et structurer les objectifs du système
- Servir de base à la traçabilité des exigences d'un système dans un processus de développement intégrant UML (tests, cahier de recette, ...)

Cas d'utilisation = Description + Diagramme



- Description textuelle des cas d'utilisation
- Diagrammes cas d'utilisation
- Acteurs
- Relations
- Système
- Note et commentaire
- Package

2.1.2.1 - Description textuelle des cas d'utilisations



Titre : 1 – Nom du scénario

Début cas d'utilisation : descriptif

Fin cas d'utilisation : descriptif

Condition :

pré-condition : descriptif

post-condition : descriptif

Acteur :

- Acteurs principaux
- Acteurs secondaires
- Matériel externe
- Autre systèmes

Scénario principal :

1.1 - descriptif de l'action (boucles, situations optionnelles)

1.2 - descriptif de l'action

1.n - etc...

Extension :

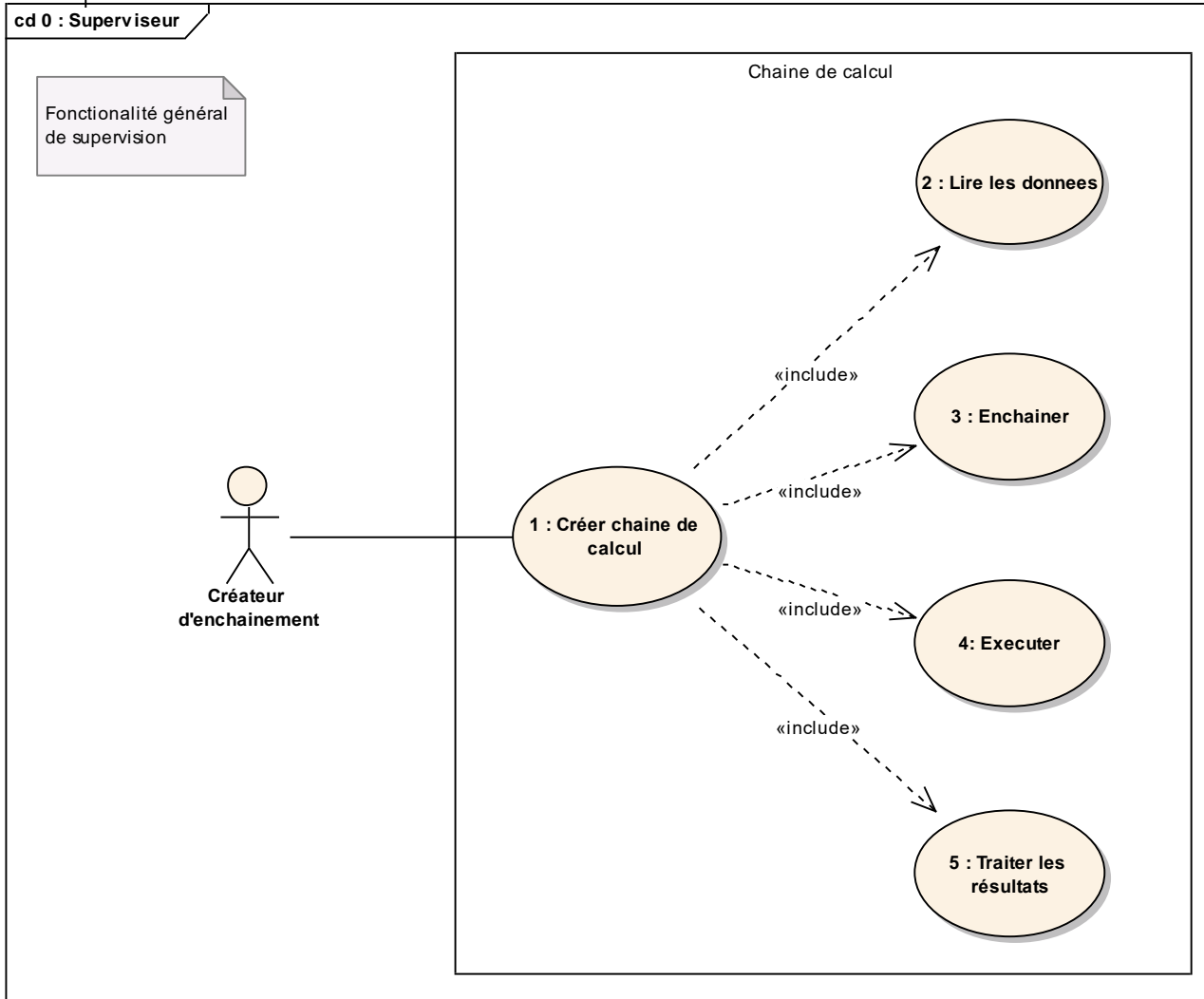
1.1 Créer la fiche modèle

1.1.1 - descriptif de l'action (boucles, situations optionnelles)

1.1.2 - descriptif de l'action

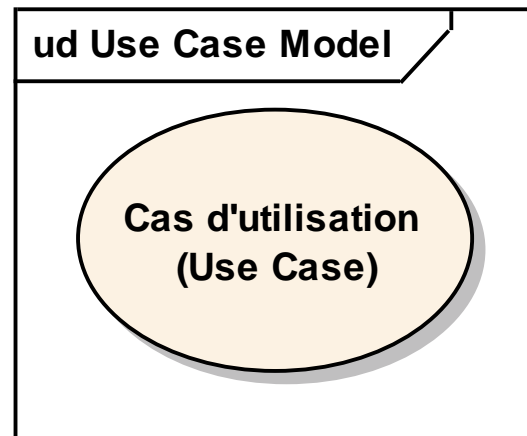
1.1.k - etc...

2.1.2.2 - Diagrammes cas d'utilisation



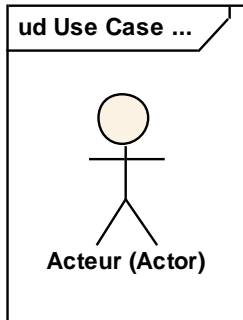


Les cas d'utilisation représentent un ensemble de scénarios d'interaction du système avec un acteur extérieur, reliés par un but commun





Les acteurs représentent le rôle d'un homme, d'une machine, d'un système extérieur ou du temps qui interagissent avec le système



4 sortes d'acteur :

- Acteurs principaux
- Acteurs secondaires
- Matériel externe
- Autres systèmes



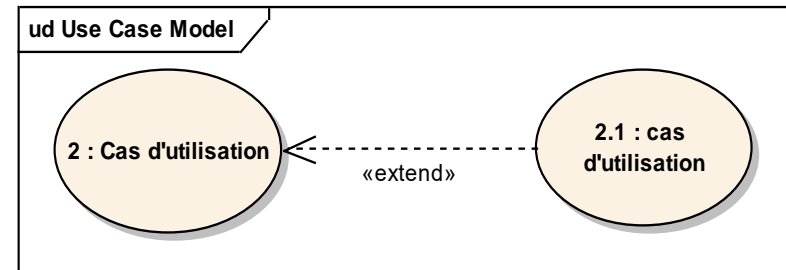
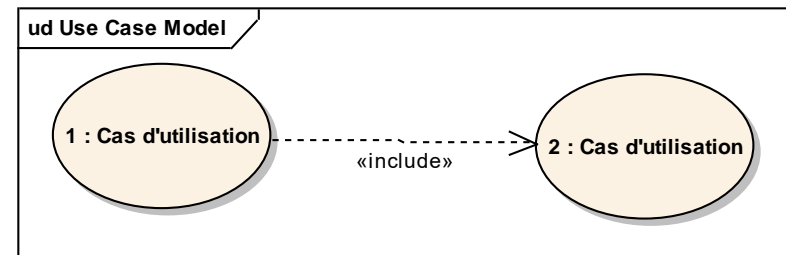
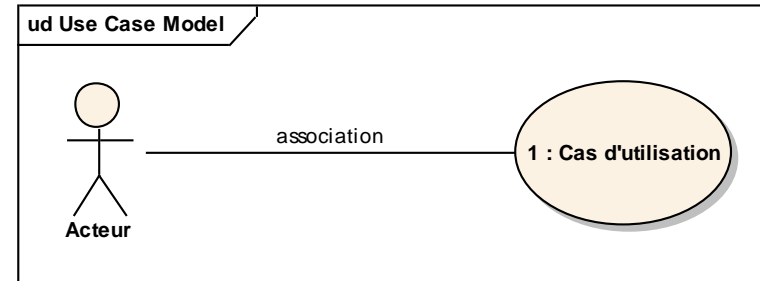
Un acteur représente le rôle d'une entité externe (*utilisateur humain ou non*) interagissant avec le système.

On représente généralement à gauche l'acteur principal, et à droite les acteurs secondaires.

Remarque : Un utilisateur peut amené à jouer plusieurs rôles vis-à-vis du système et à ce titre être modélisé par plusieurs acteurs.



- **Association entre acteur et cas**
 - un acteur interagit avec un cas d'utilisation en le déclenchant
- **Relation d'inclusion**
 - Le cas d'utilisation source (1) contient aussi le comportement décrit dans le cas d'utilisation destination (2) (*a besoin de*)
- **Relation d'extension**
 - Le cas d'utilisation source (2.1) précise le comportement du cas d'utilisation destination (2) (*peut avoir besoin de*)

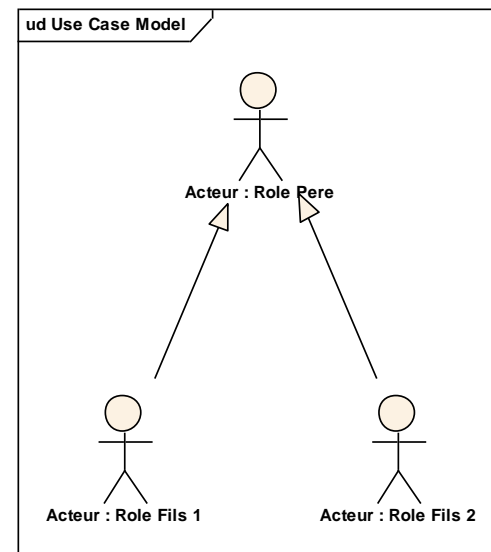
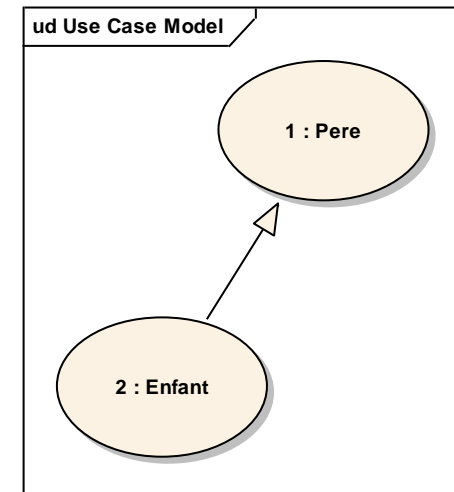




Quand un cas n'est pas directement relié à un acteur, mais qui est utiliser par un autre cas d'utilisation => il est qualifié de cas d'utilisation interne.

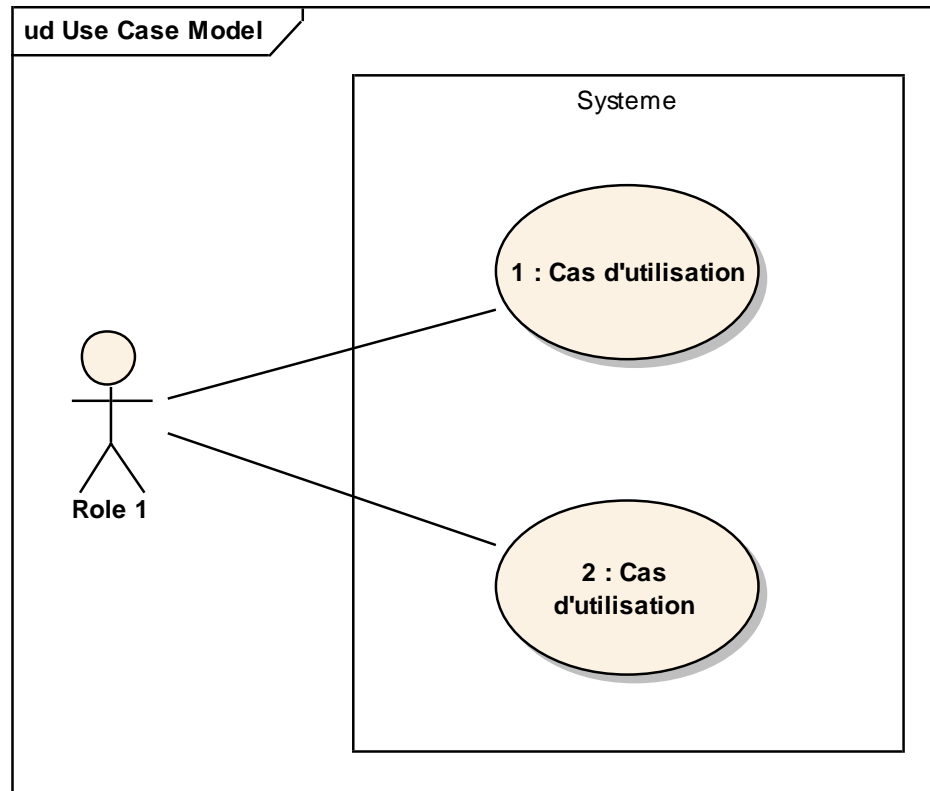


- **Relation de Généralisation entre cas d'utilisation**
 - Le cas d'utilisation enfant (5) est une spécialisation du cas d'utilisation père (4)
- **Relation de Généralisation entre acteurs**
 - Meilleure identification des rôles



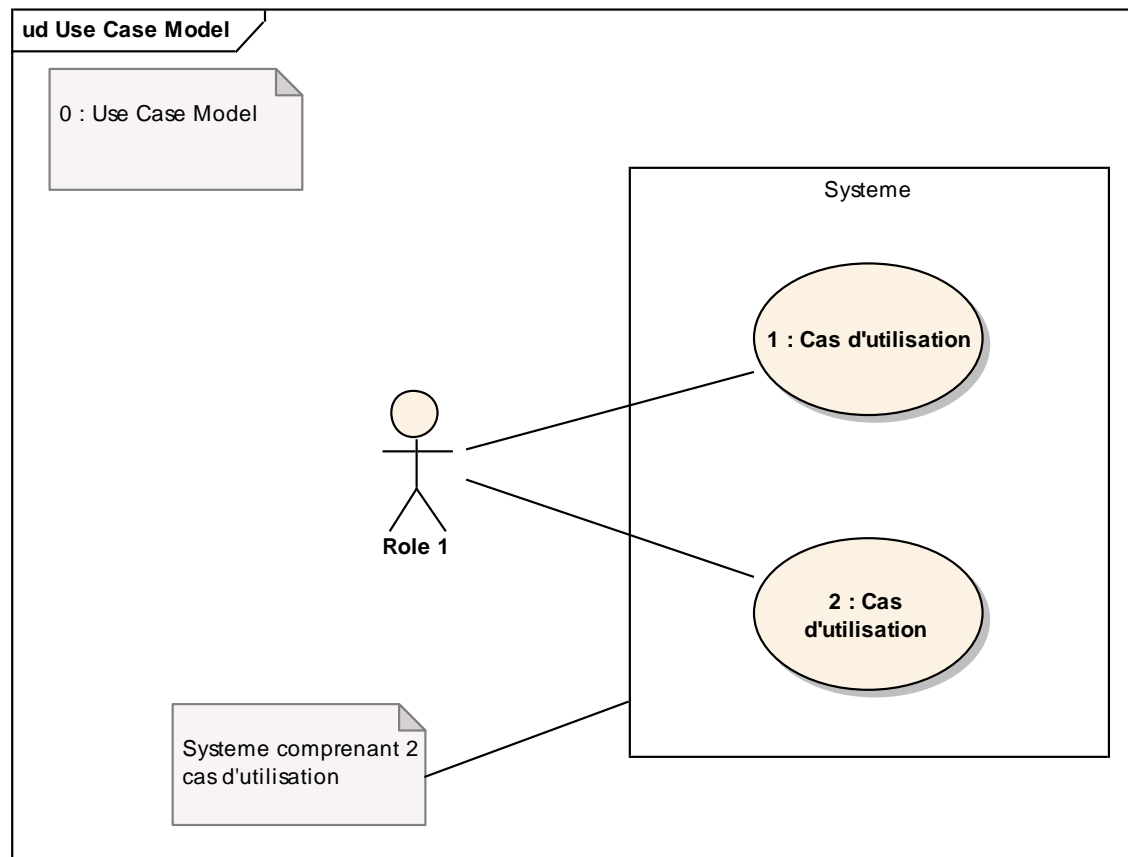


Les cas d'utilisation peuvent être contenus dans un rectangle représentant les limites du système





Une note est un symbole graphique qui contient des informations

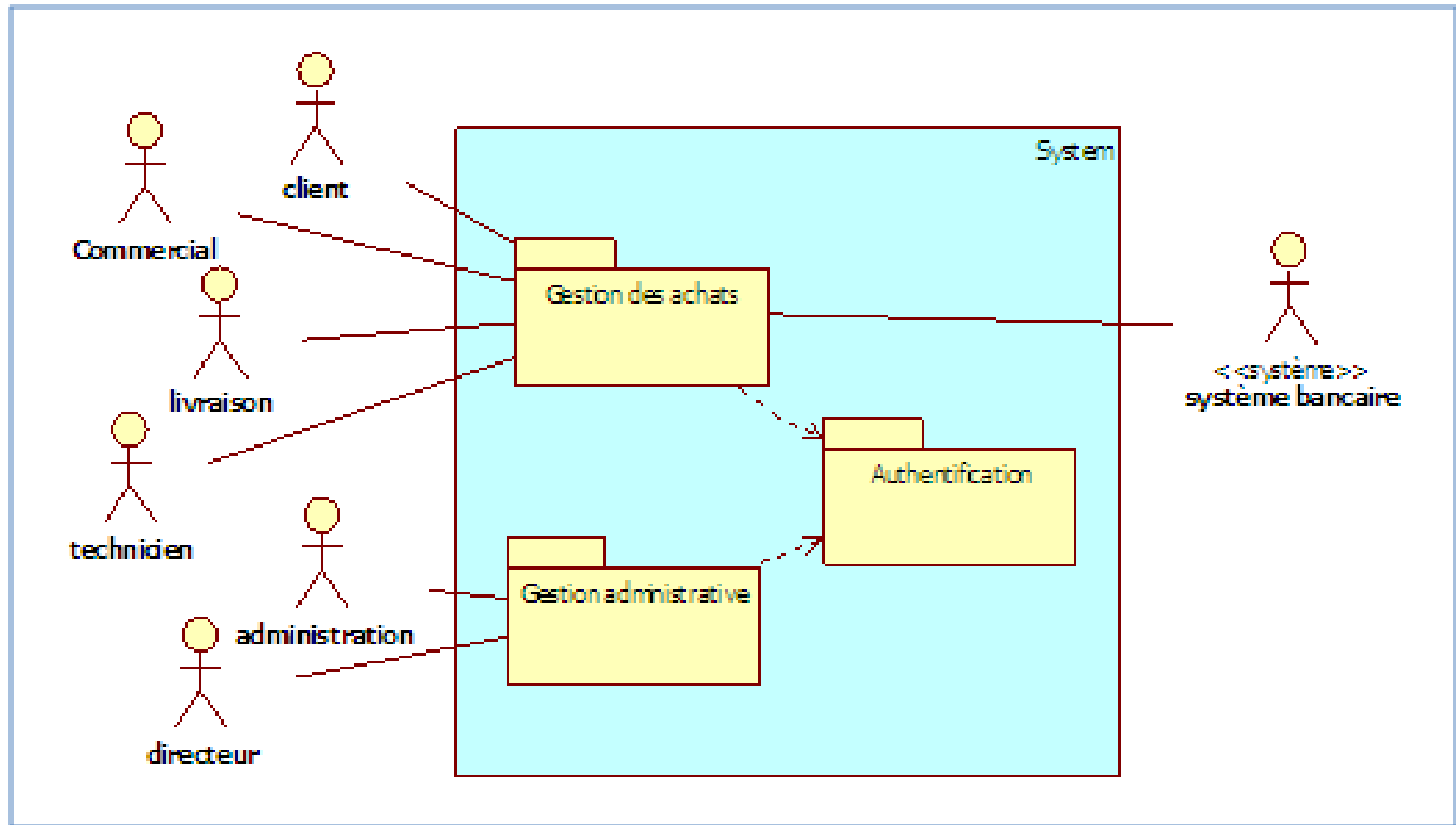




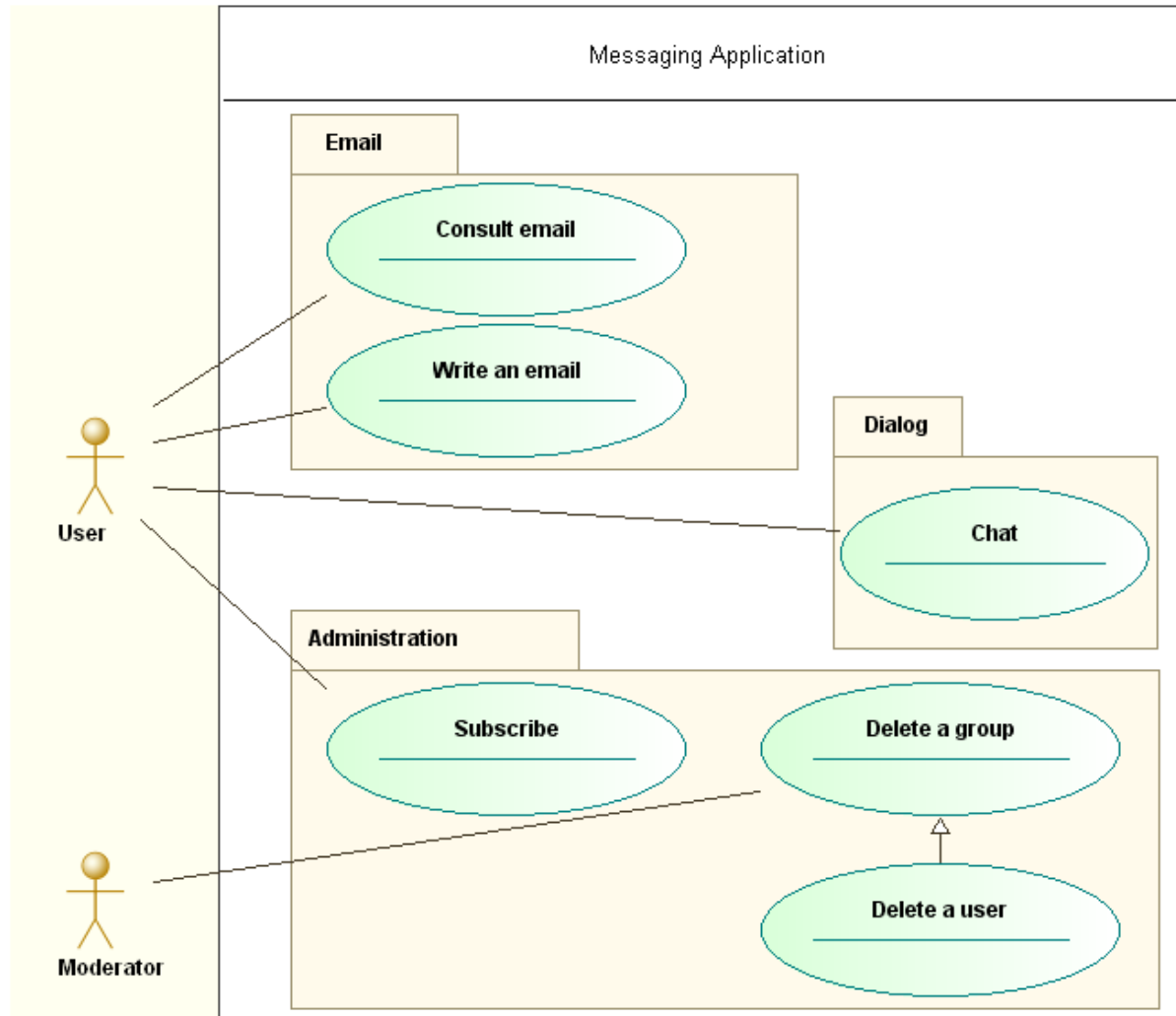
Le diagramme de packages pour les use case, permet de décomposer le système en catégories ou parties plus facilement observables, appelés « packages ».

Un package est constituer de plusieurs fonctionnalités qui forment une famille.

Cela permet également d'indiquer les acteurs qui interviennent dans chacun des packages.



2.1.2.8 - Package



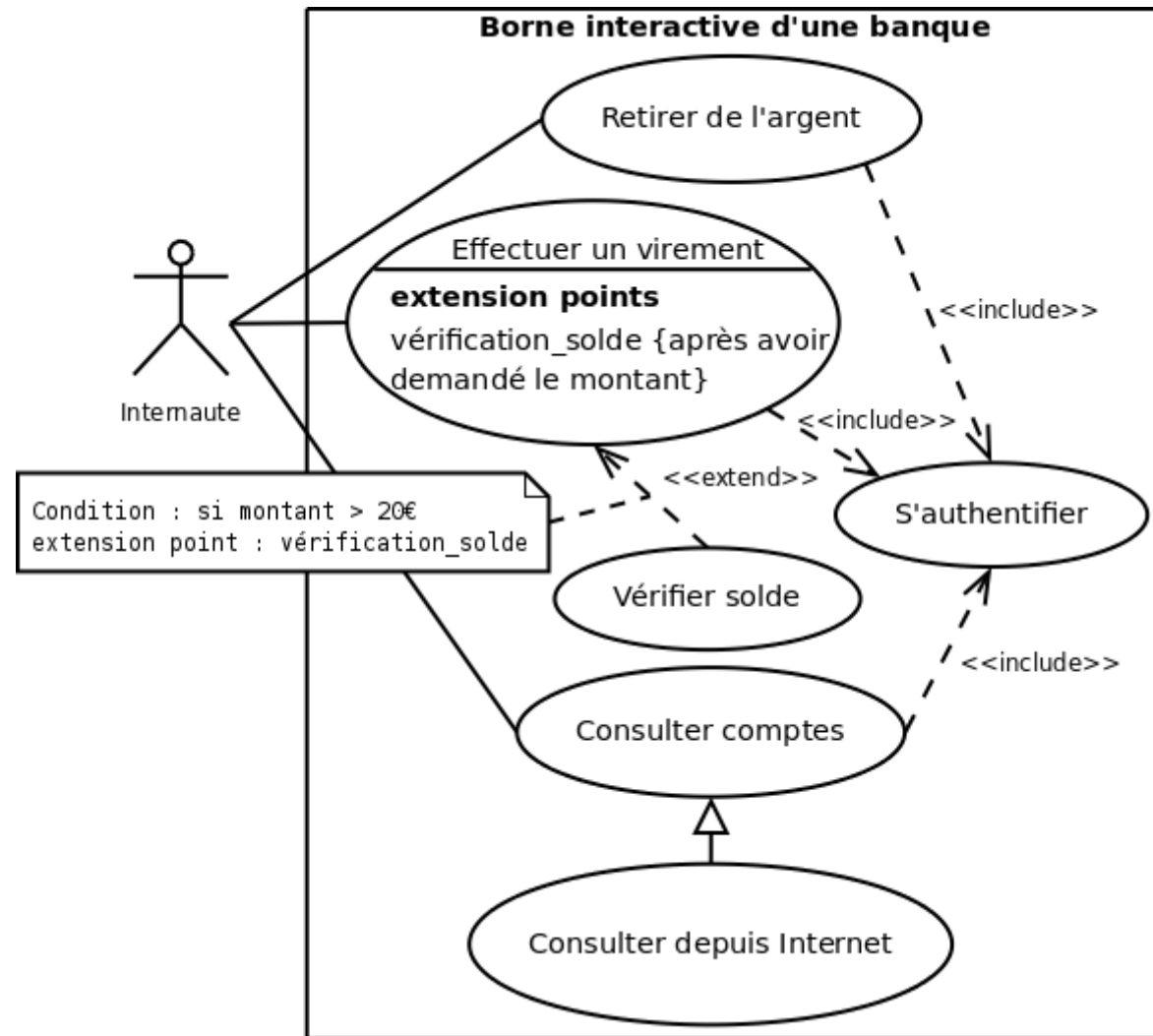


- **Écrire les premiers cas d'utilisation globaux**
- **Détailler les parties principales**
- **Développer les parties**

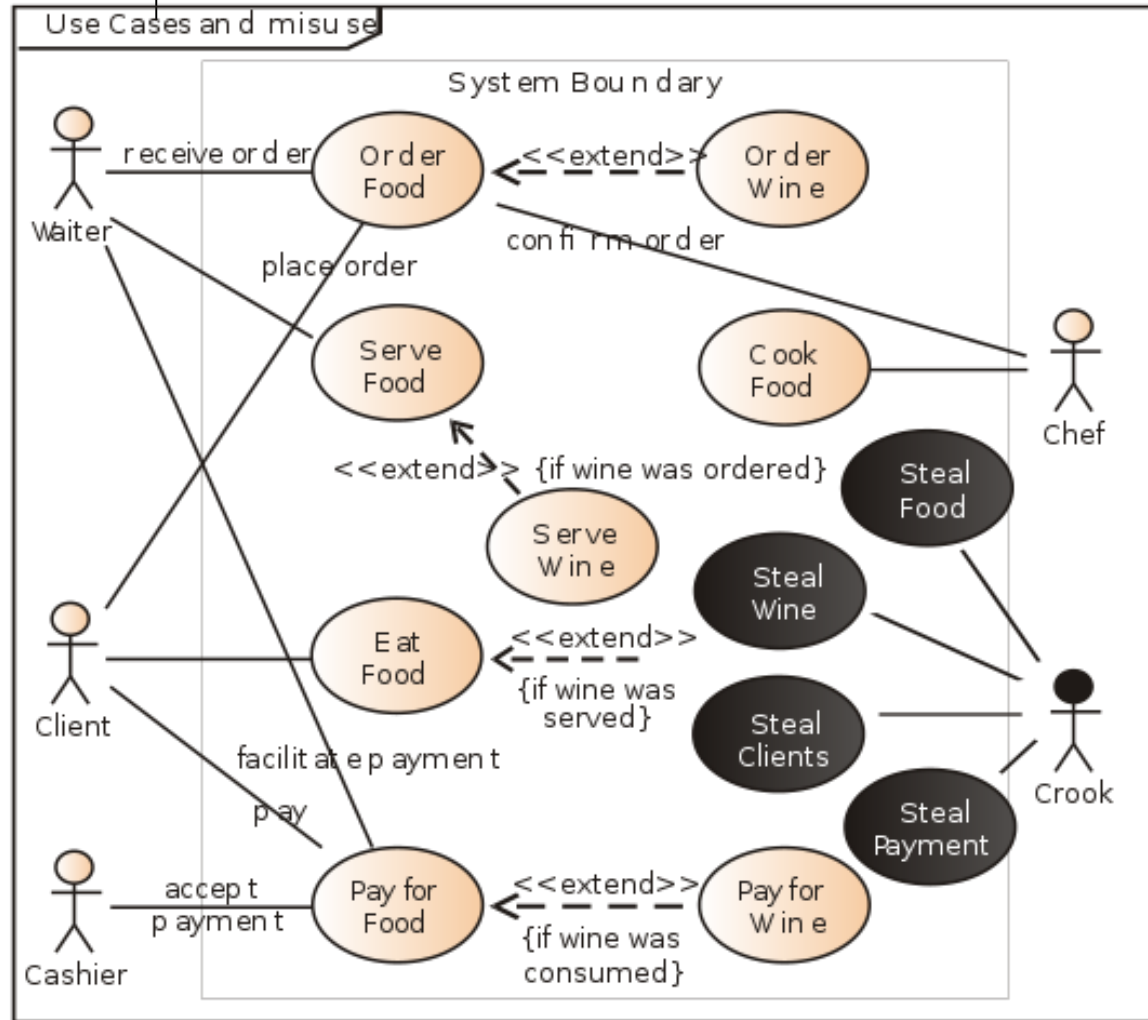
Attention :

- **Limiter les cas d'utilisation : 10-20 maximum pour un projet moyen**
- **Faire des cas d'utilisation simples**

2.1.4 - Exemple



2.1.4 - Exemple



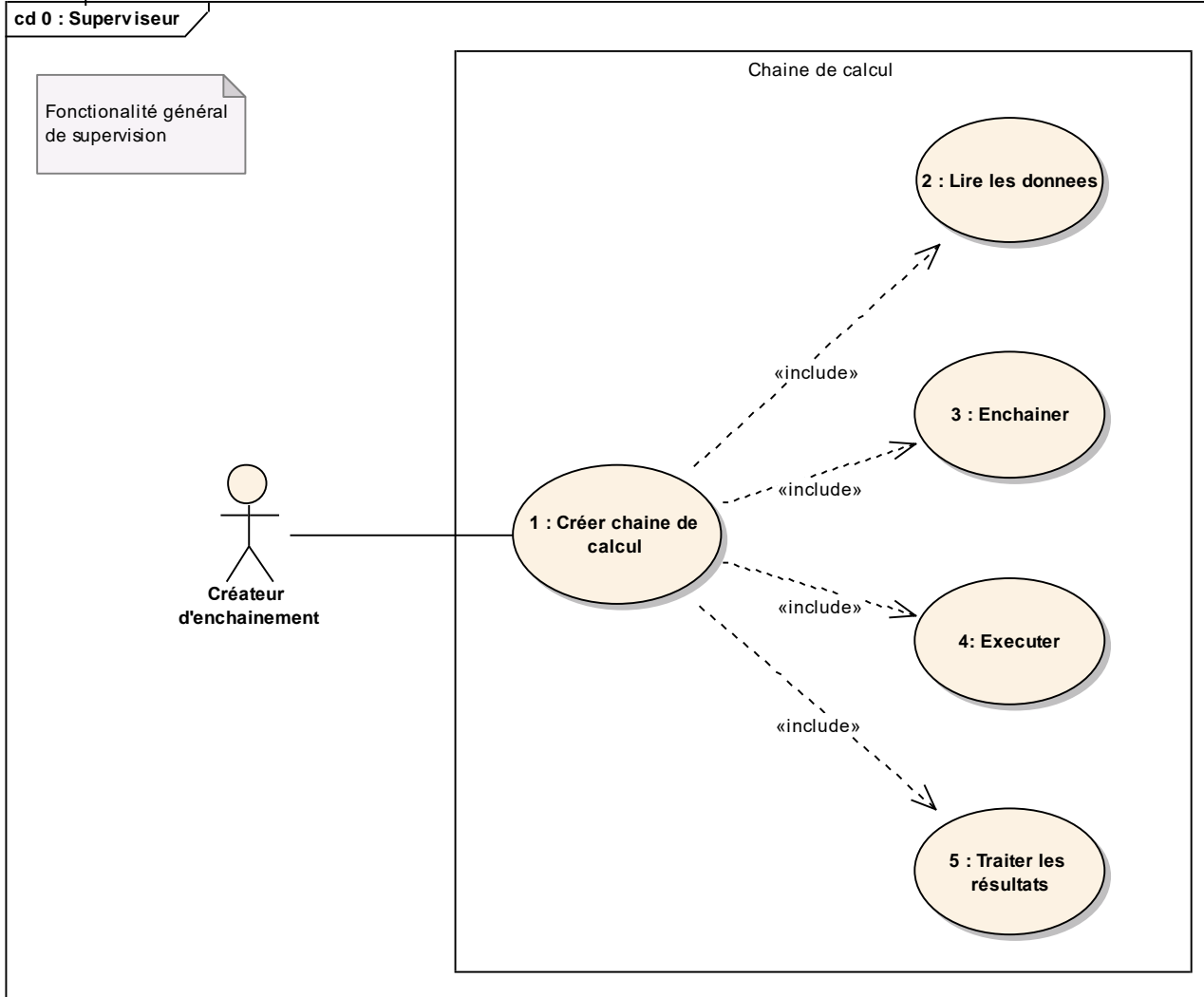
https://en.wikipedia.org/wiki/Misuse_case#From_use_to_misuse_case

2.1.4 - Exemple



- Exemple : Chaîne de calcul
- Exemple : Lecture des données
- Exemple : Enchaînement de code

2.1.4.1 – Exemple : Chaîne de calcul



2.1.4.2 – Exemple : Lecture des données



ud 2 : Lire les donnees

2 : Lire les données

:Créateur
d'enchainement

Lecture données

2.1 : Rentrer les
données

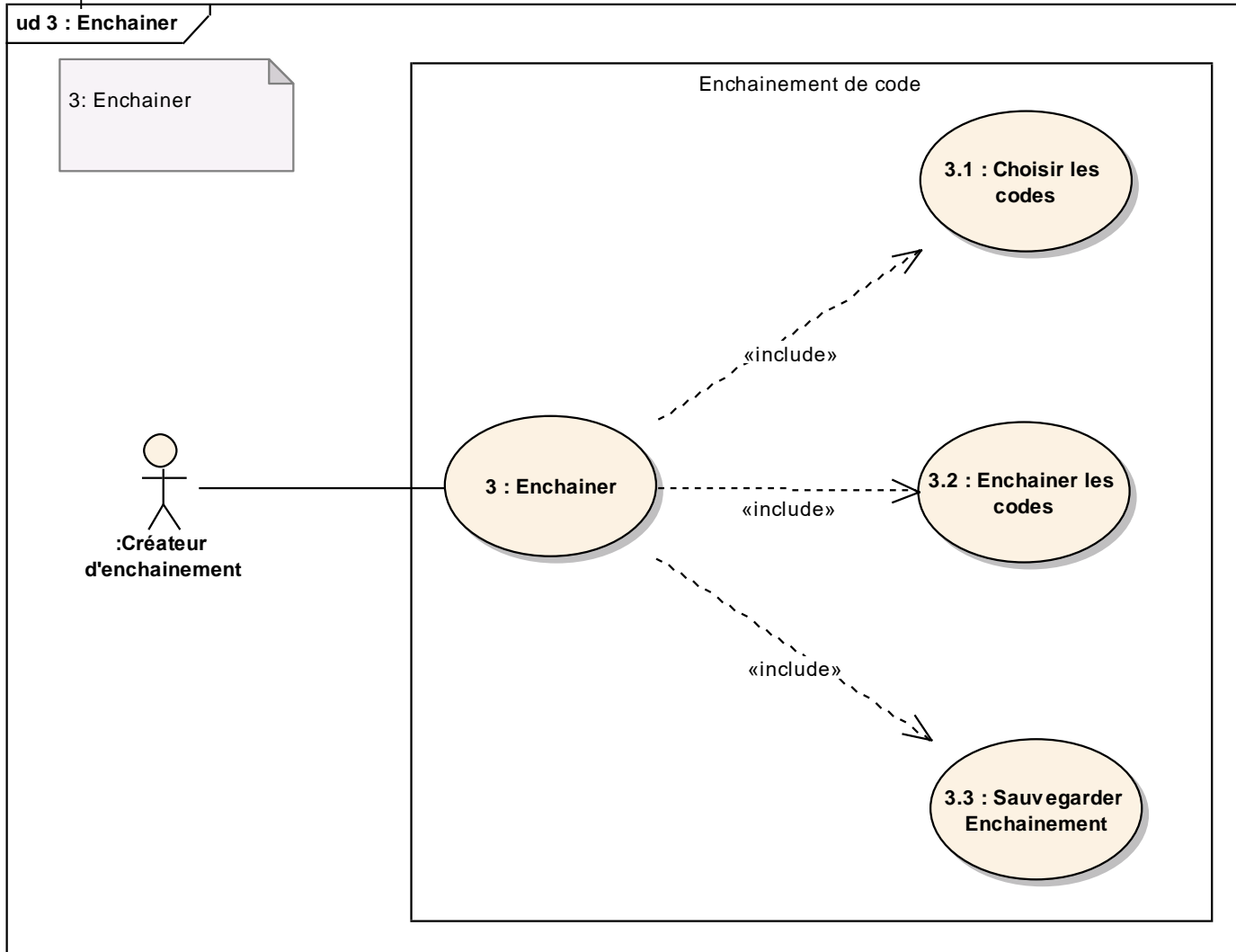
«include»

2 : Lire les
donnees

«include»

2.2 : Mettre en
forme les données

2.1.4.3 – Exemple : Enchaînement de code



2.2 - Le diagramme de classe



- Fonctionnalité
- Notation
- Mise en œuvre
- Exemple



Un diagramme de classe est une collection d'éléments de modélisation statiques (classes, paquetage, ...) qui permet de:

- Montrer la structure statique d'un problème en termes de classes et de relations entre ces classes
- Faire abstraction des aspects dynamiques et temporels
- Représenter un contexte précis : un diagramme de classes peut être instancié en diagramme d'objets
- Définir la structure statique d'un problème
- D'organiser l'espace du problème (analyse : catalogue les objets) ou de la solution (conception : organise le système)

2.2.2 - Notation

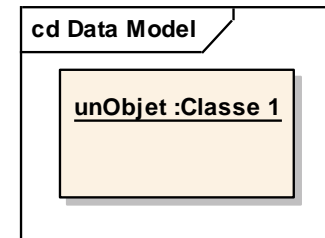
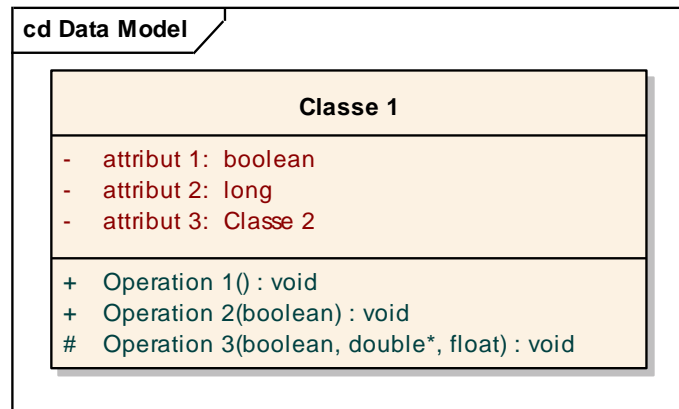


- **Classes**
- **Attributs**
- **Opérations**
- **Attributs et Opérations statiques**
- **Association**
- **Multiplicité**
- **Navigabilité**
- **Agrégations**
- **Compositions**
- **Associations qualifiées**
- **Classes associations**
- **Propriétés dérivées**
- **Dépendance**
- **Généralisation/classification statique**
- **Héritage multiple (A éviter en conception)**
- **Classification dynamique**
- **Interface et classes abstraites**
- **Classes actives**
- **Classes paramétrables**
- **Énumérations**
- **Visibilité**



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

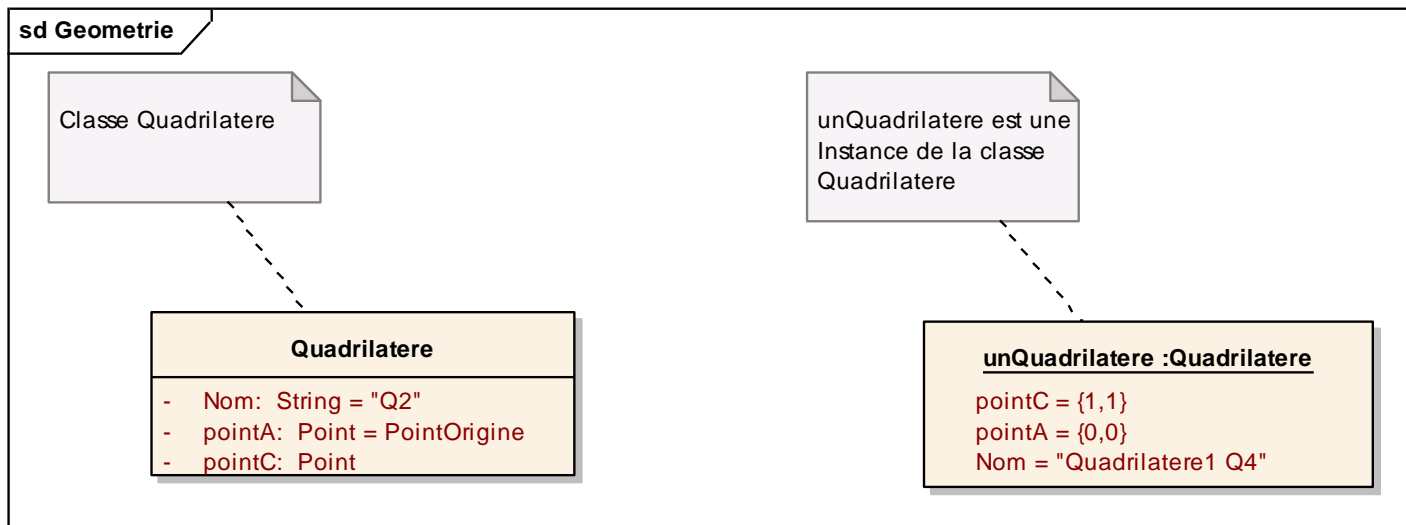
Classe = attributs + opération + instantiation





Les **attributs** correspondent aux propriétés de la classe, définis par

- Un **nom** unique
- Un **type**
- Une **valeur** initiale (*optionnel*)



2.2.2.3 - Opérations

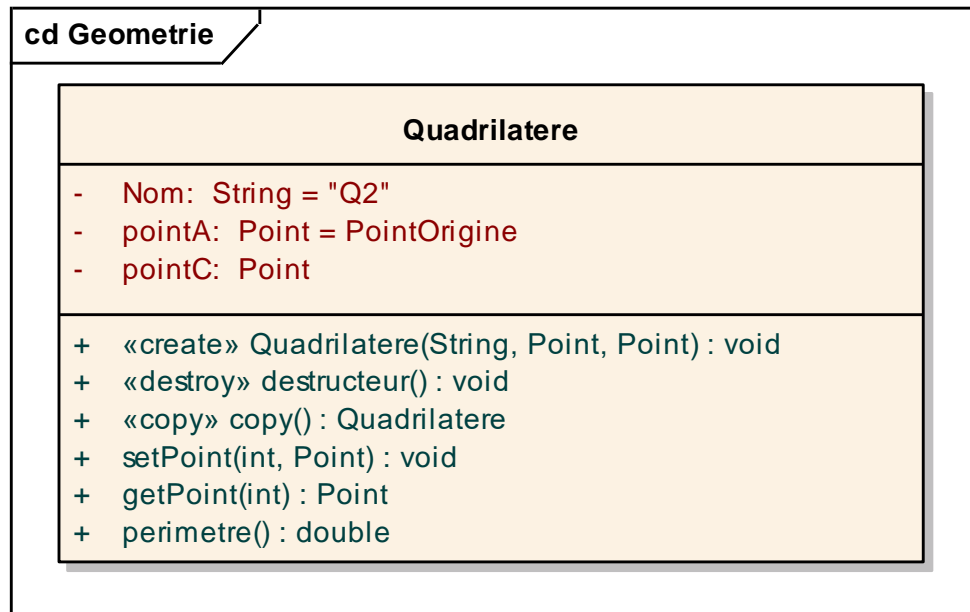


Les opérations spécifient le comportement d'un objet.
La réalisation du comportement est exprimée dans les méthodes

- Une opération est un service offert par les instances de la classe
- Une méthode est l'implémentation d'une opération

5 principaux types d'opération :

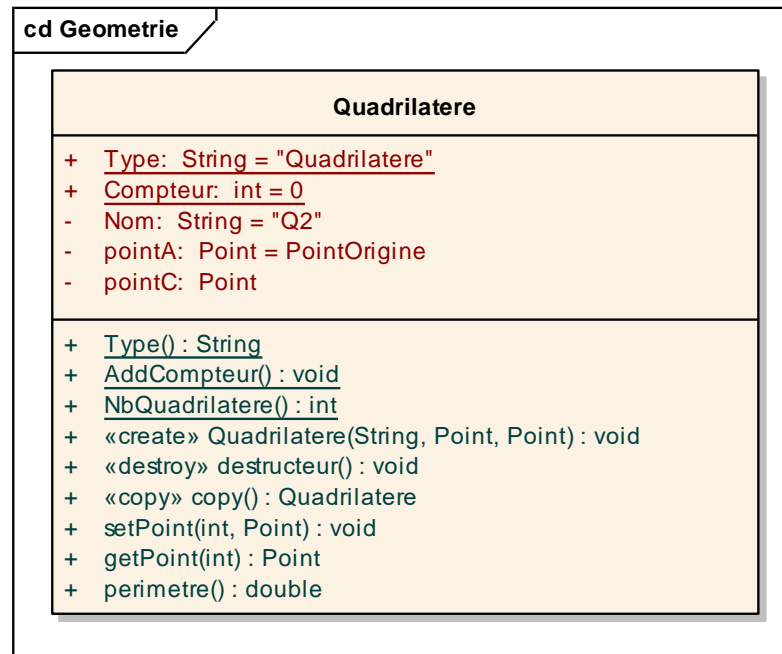
- Constructeurs
- Destructeurs
- Sélecteurs
- Modificateurs
- Itérateurs



2.2.2.4 - Attributs et Opérations statiques



Les attributs ou les opérations statiques s'appliquent à une classe et nom à ses instances

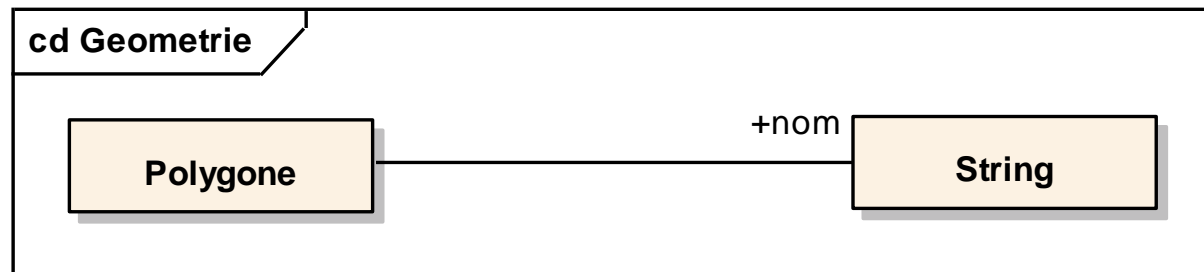


2.2.2.4 - Association



Une association représente des relations structurelles. Elle exprime une connexion sémantique bidirectionnelle entre deux classes .

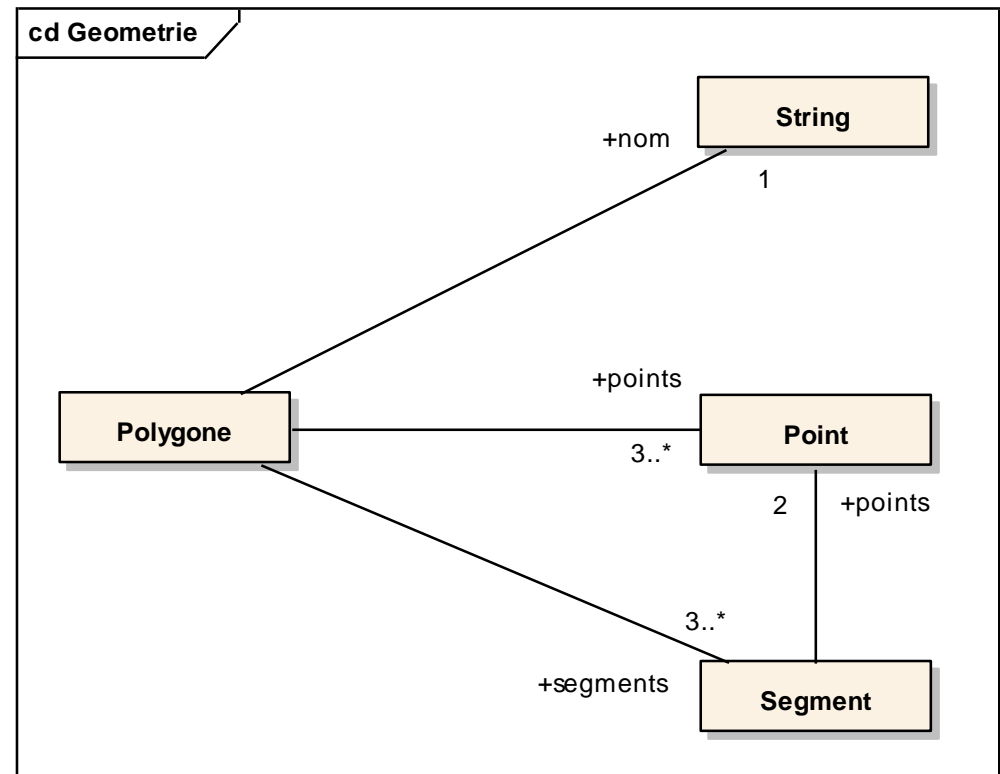
L'association est instanciable dans un diagramme d'objet ou de collaboration sous forme de liens entre objets issus de classes associées.





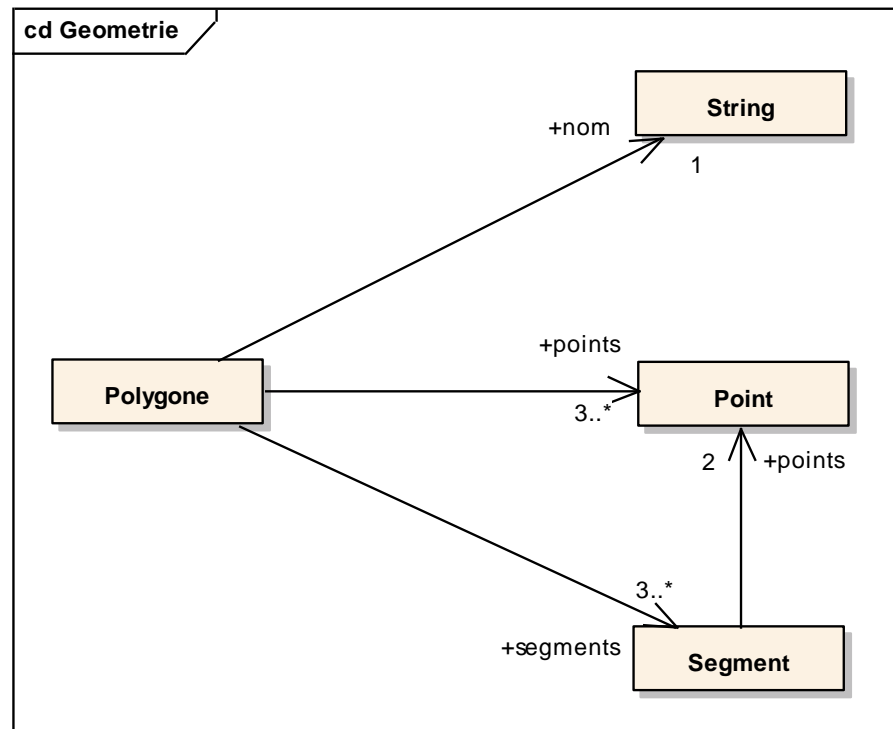
La **multiplicité** spécifie le **nombre d'instances** d'une classe **en relation** avec une instance de la classe associée

- 0..1
- 0..*
- 1..*
- 1
- *
- Autre : 4..6, 5, ...





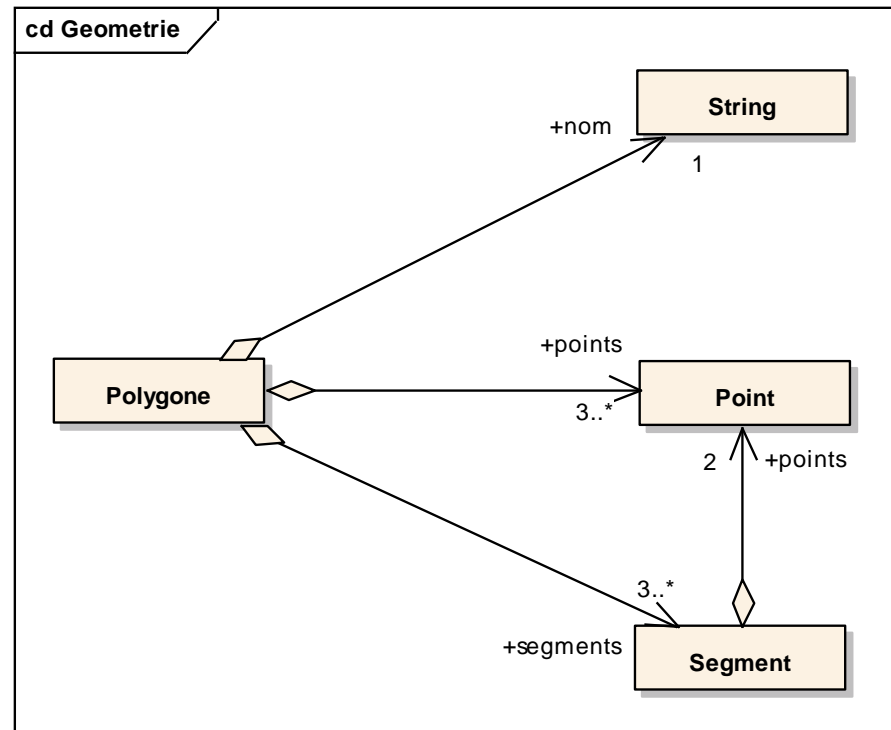
Une association représente des relations structurelles.
Par défaut une association entre deux classes est
bidirectionnelle (faux dans certain cas)





L'agrégation est un cas particulier d'association asymétrique.

Elle permet de modéliser une contrainte d'intégrité.

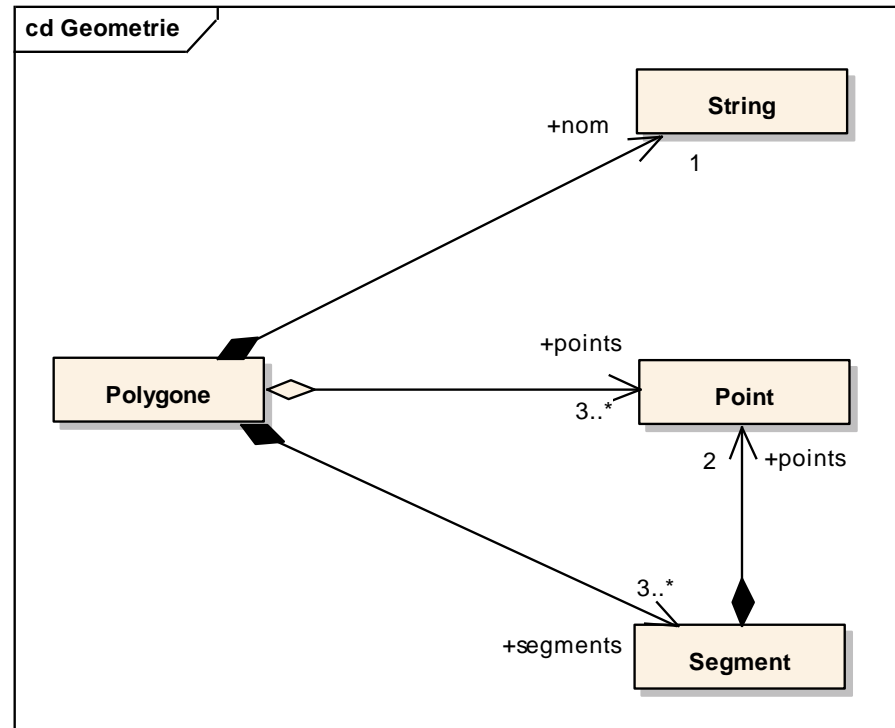


2.2.2.8 - Compositions

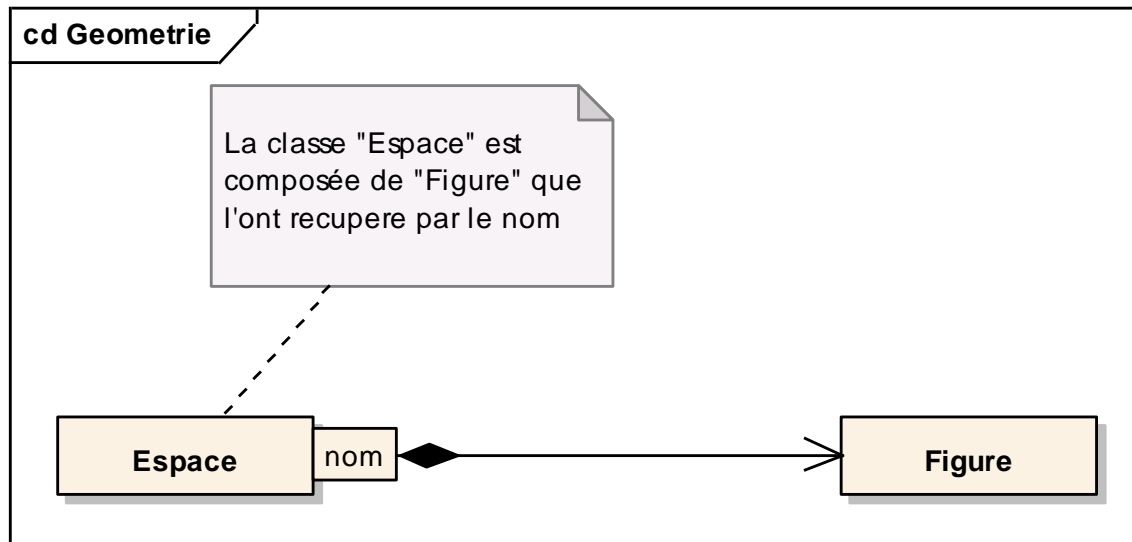


La composition est un cas particulier d'agrégation avec un couplage plus fort.

Elle implique une coïncidence des durées de vie du composant et du composite.

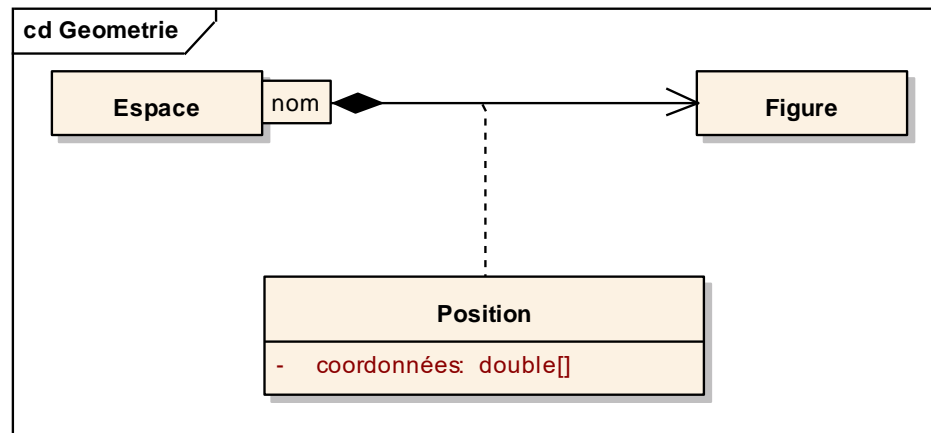


2.2.2.9 - Associations qualifiées





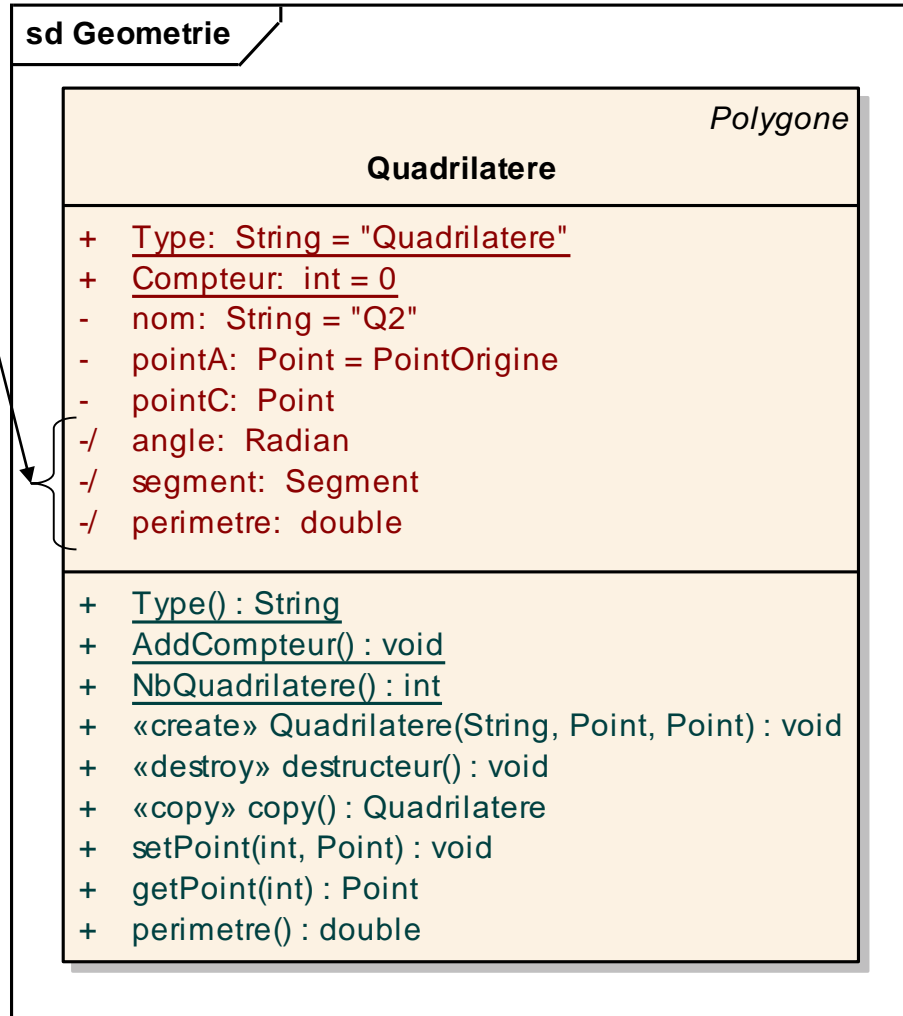
Les classes associations permettent d'ajouter des attributs, des opérations et d'autres fonctionnalités aux associations.



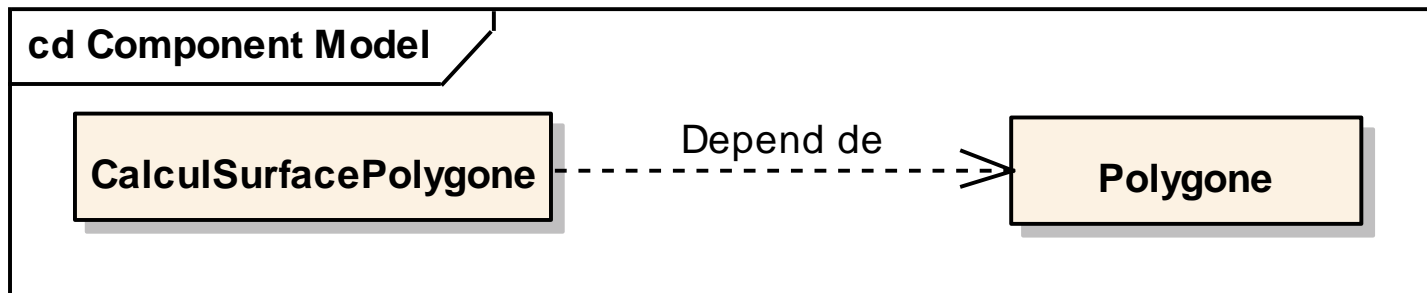
2.2.2.11 - Propriétés dérivées



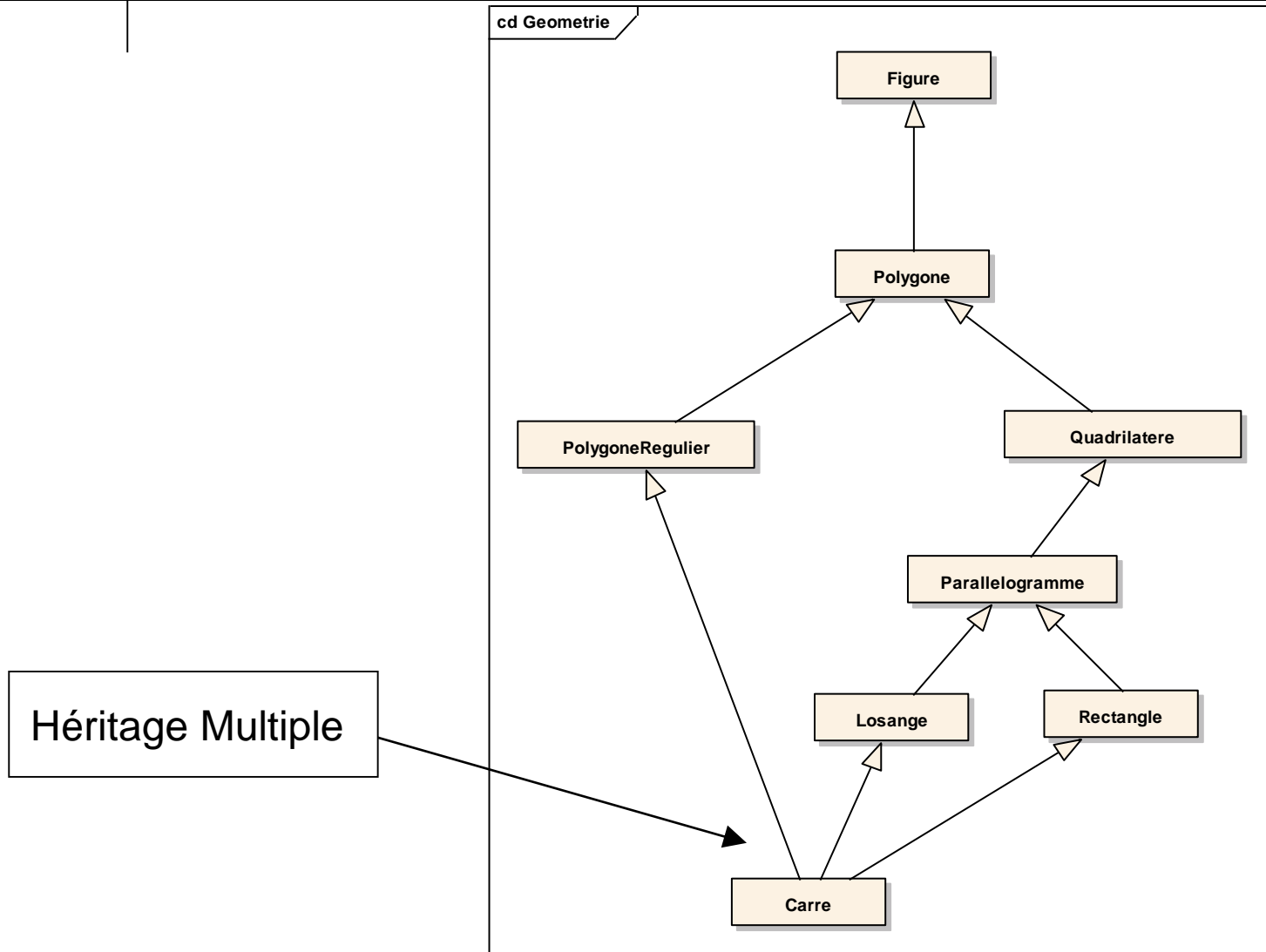
Propriétés dérivées



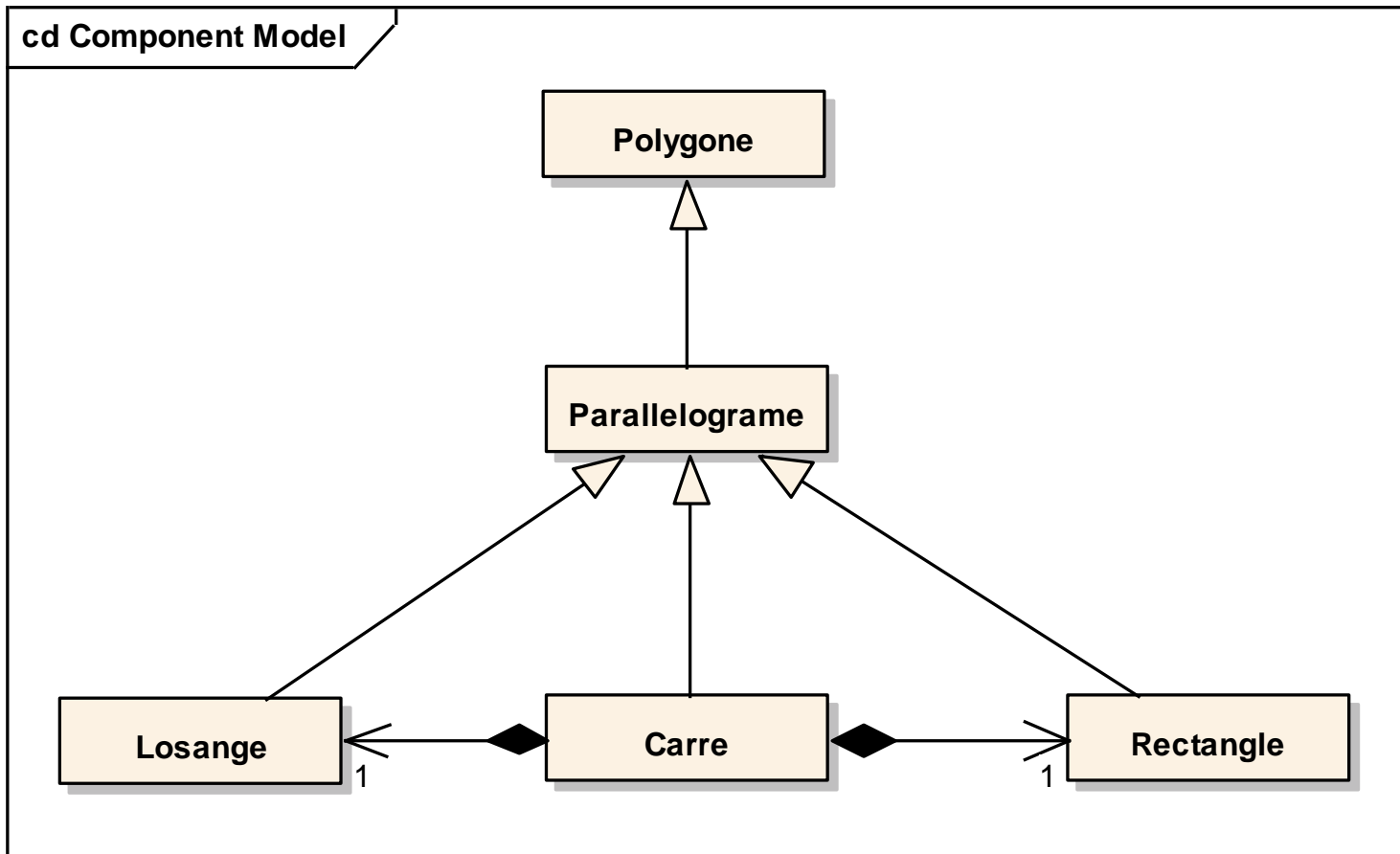
2.2.2.12 - Dépendance



2.2.2.13 - Généralisations/classification statique

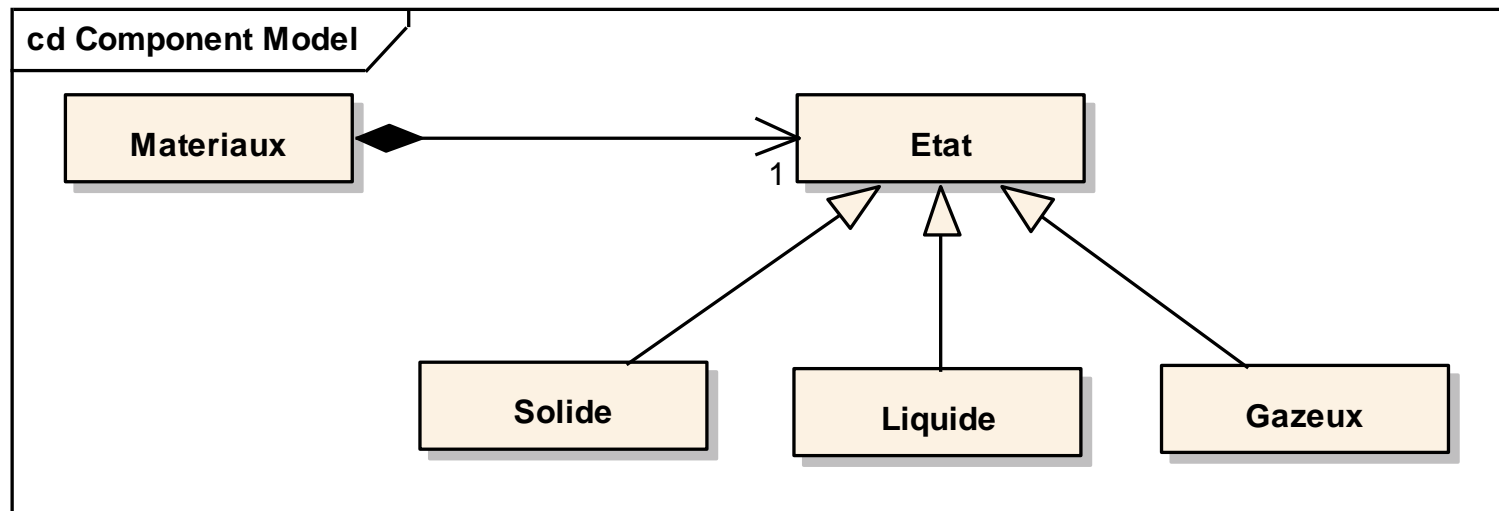


2.2.2.14 - Héritage Multiple (A éviter en conception)





Un objet peut-il changer de classe pendant l'exécution ?





Classe Abstraite :

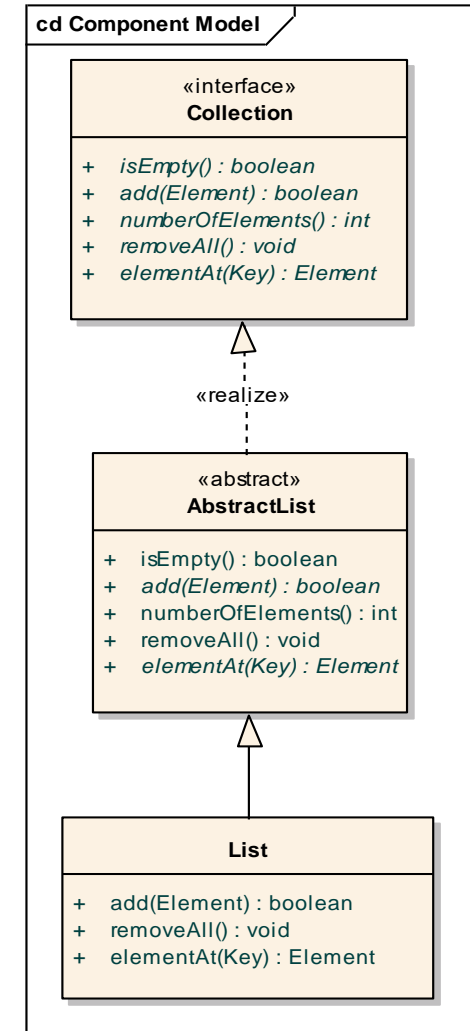
Une classe abstraite est une classe qui ne peut être instanciée directement.

Une opération abstraite n'a pas d'implémentation

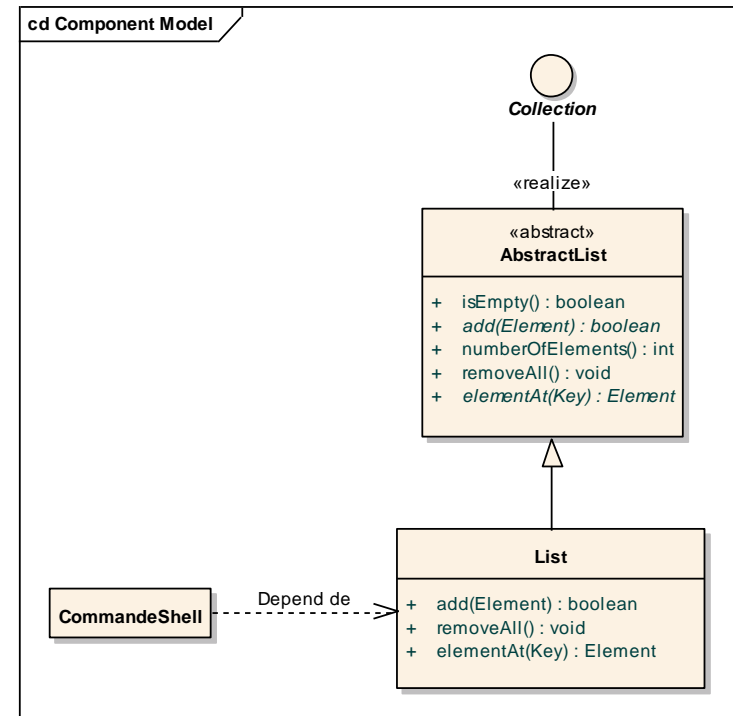
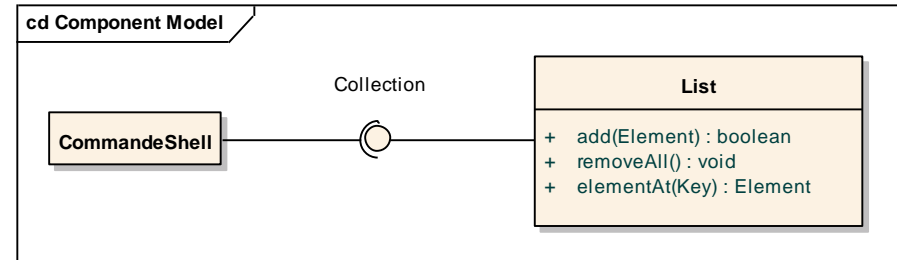
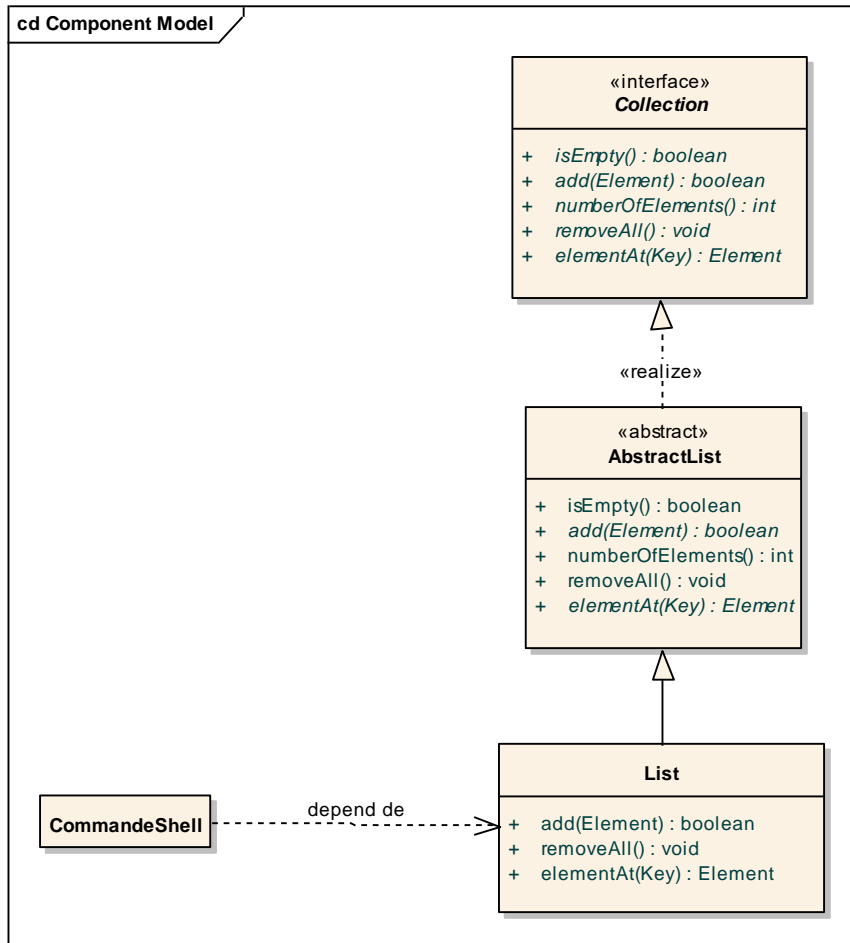
Classe Interface

Une interface est une classe qui n'a pas d'implémentation.

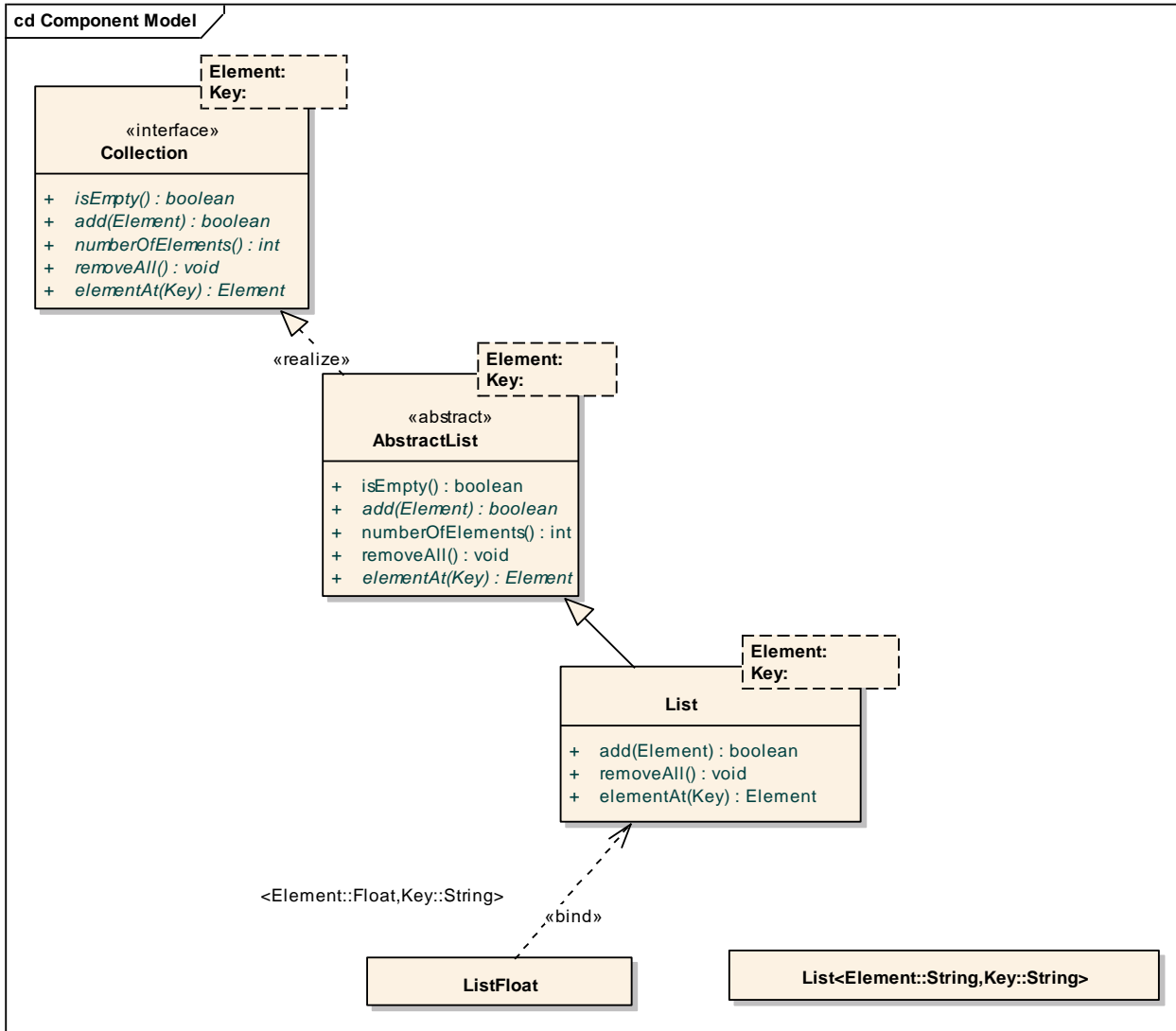
Toutes ses caractéristiques sont abstraites



2.2.2.16a - Interface et classes abstraites

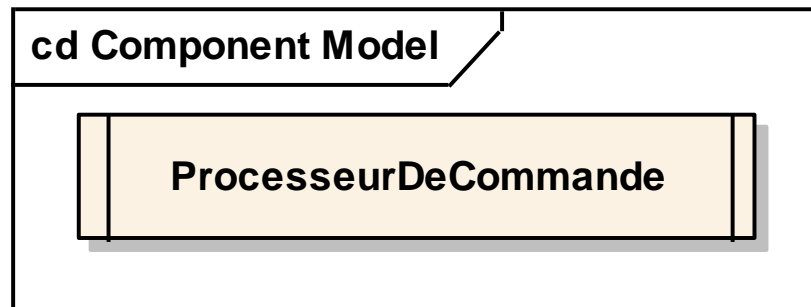


2.2.2.17 - Classes paramétrables



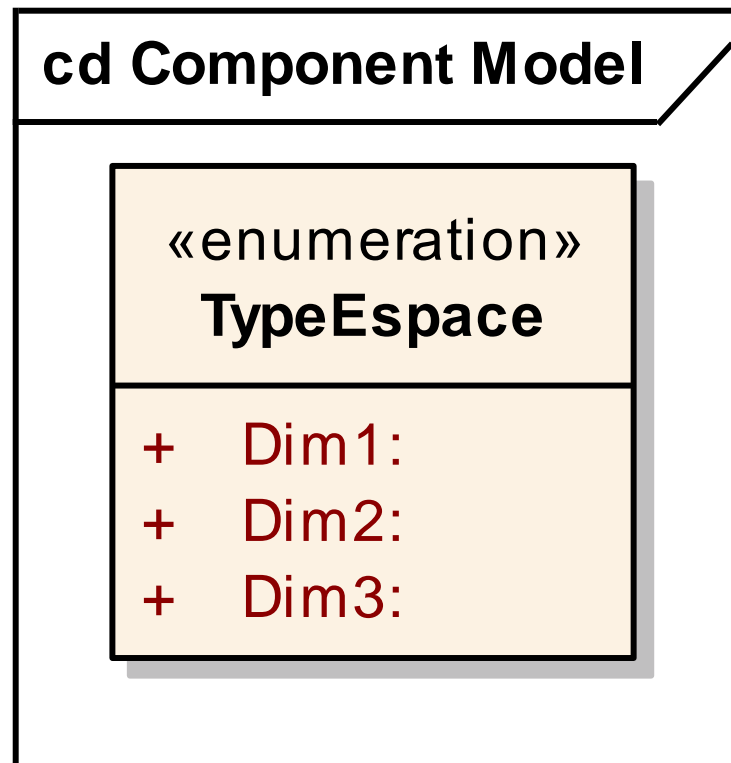


Une classe active est une classe qui a des instance exécutant et contrôlant son propre thread





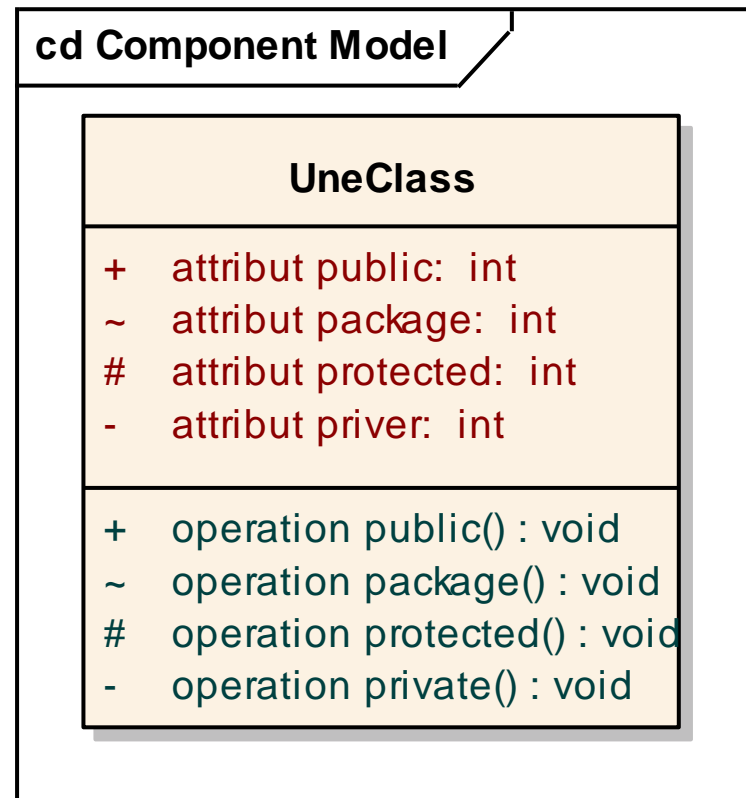
Les énumérations servent à indiquer un ensemble fixé de valeurs qui n'ont aucune propriété





Toute classe a des éléments publics et privés.
UML fournit une représentation de ces différences à travers 4 abréviations :

- privé
- + public
- # protégé
- ~package



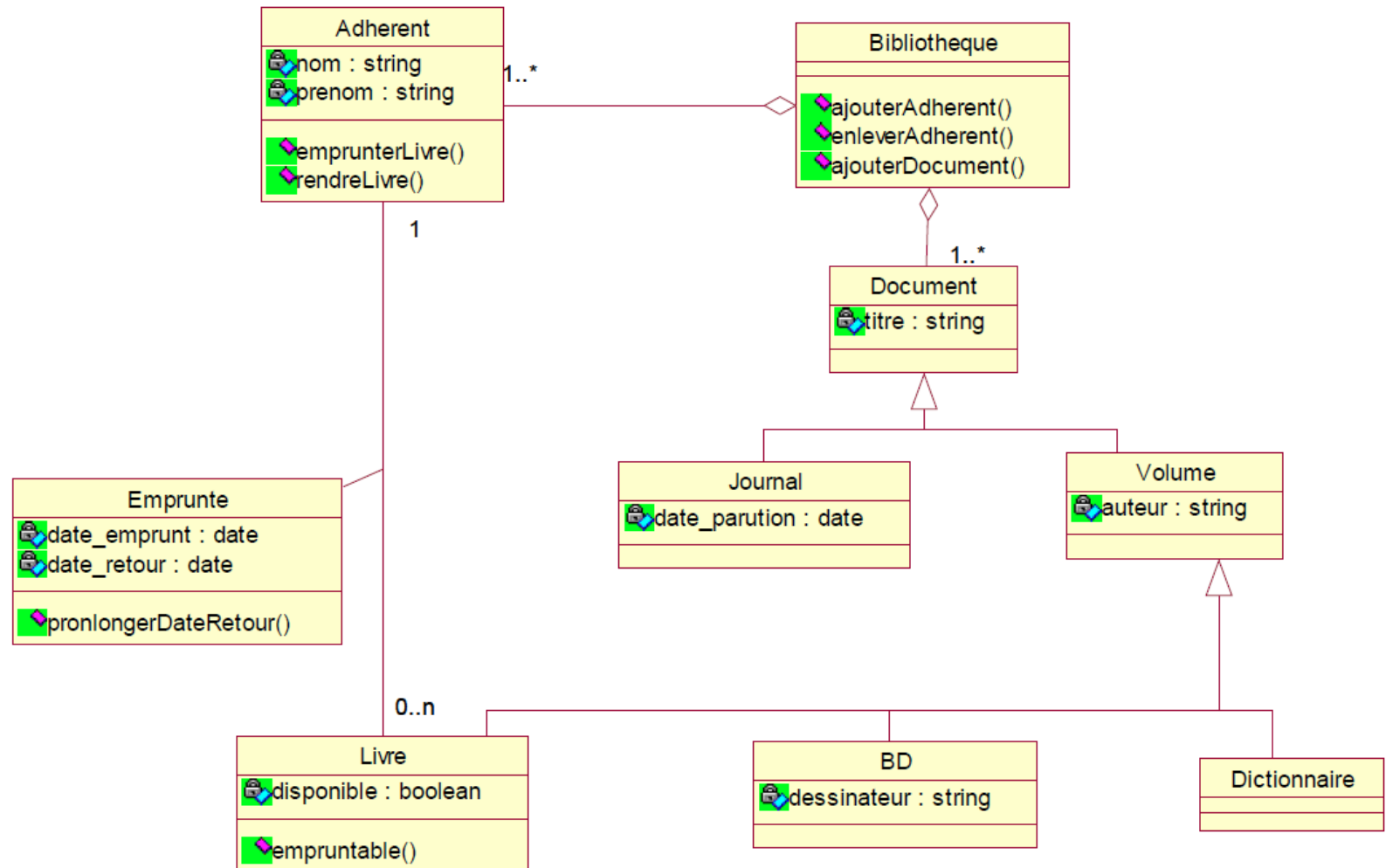


Les diagrammes de classe sont utilisés en permanence

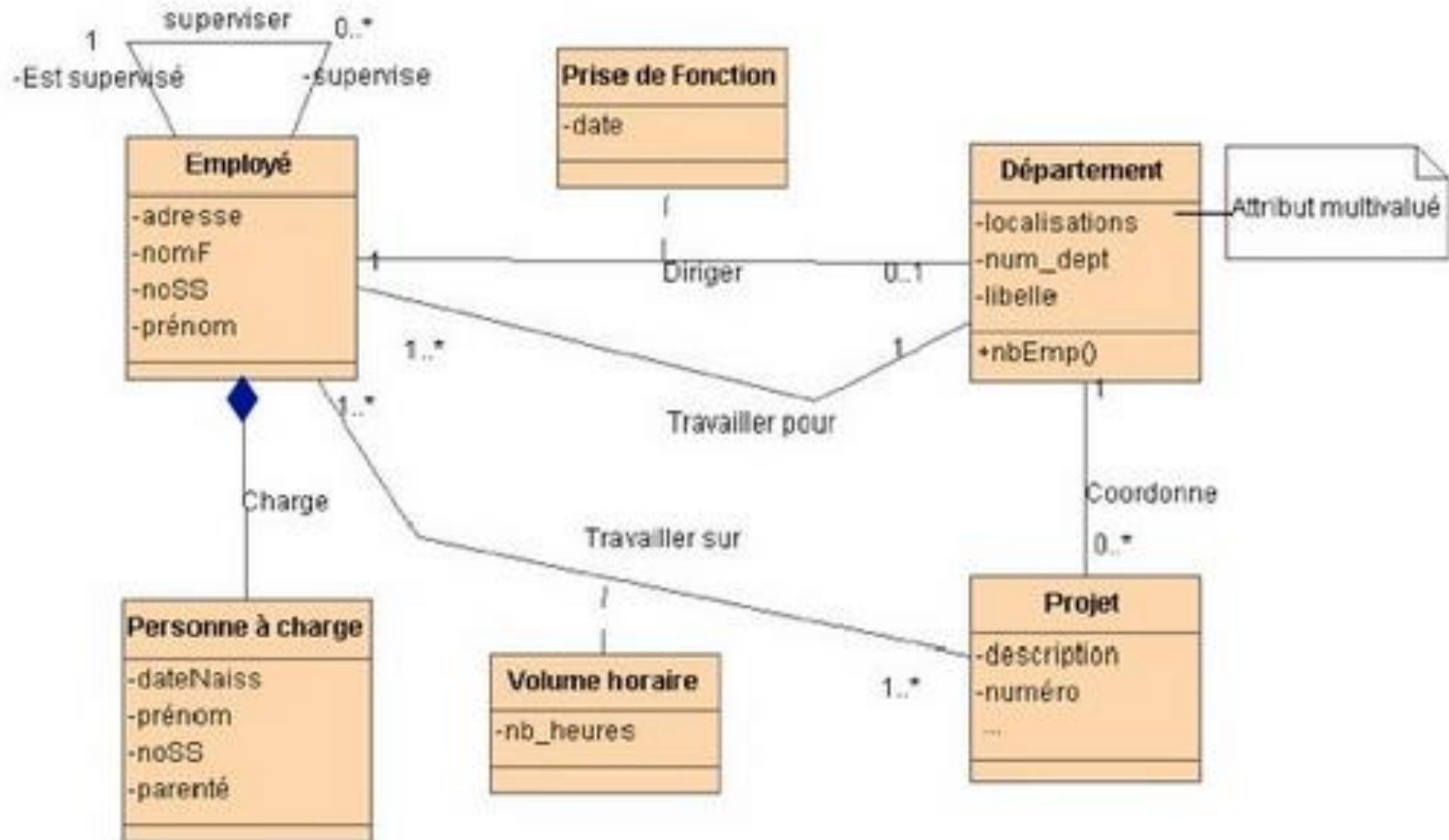
Conseils de mise en œuvre :

- Ne pas essayer d'utiliser tous les concepts,
- commencer par des notions simples : classe, association attribut, généralisation, contraintes
- Ne pas créer systématiquement des diagrammes
- le diagramme de classe conceptuelle peut servir à identifier le langage métier

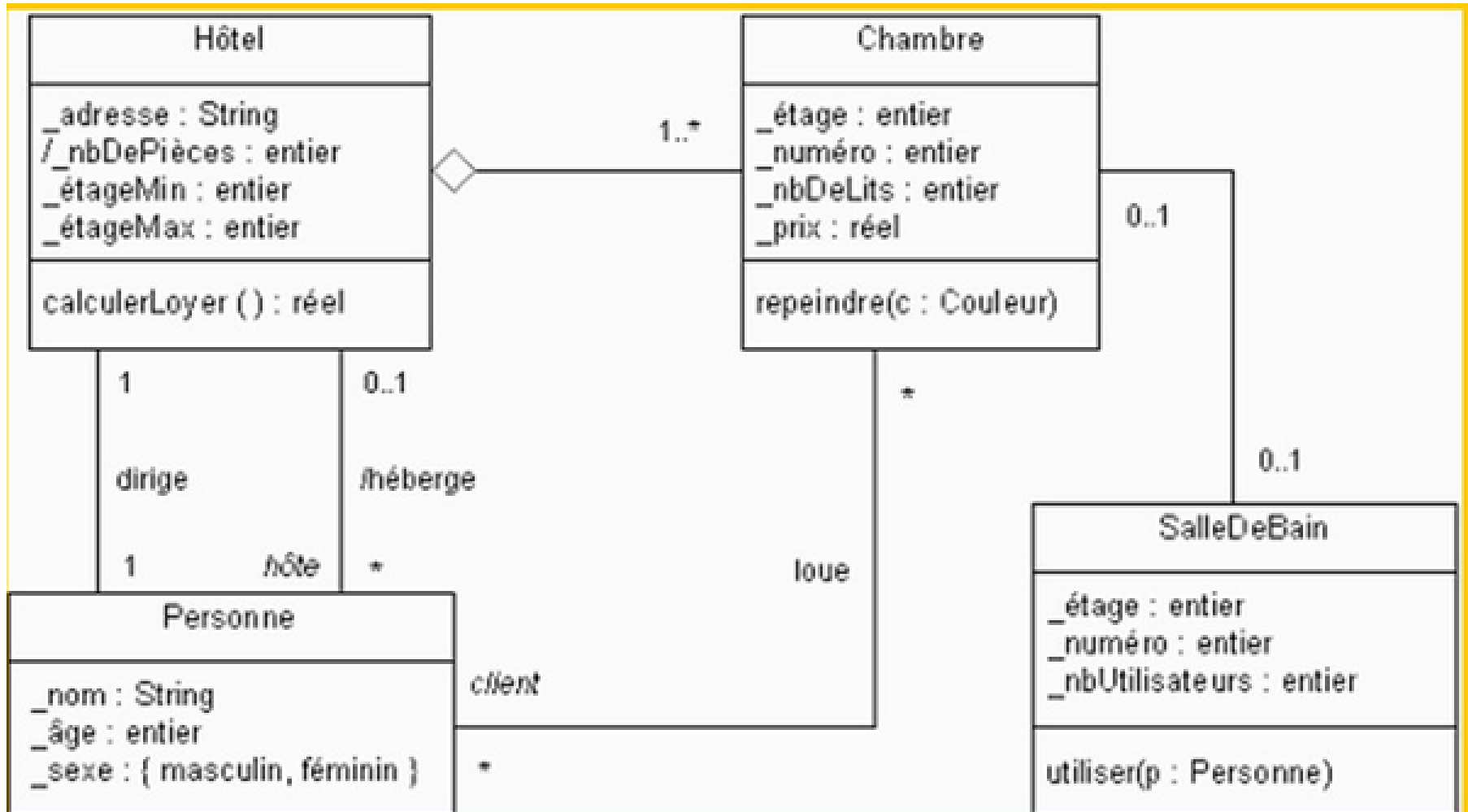
2.2.4 – Exemple



2.2.4 – Exemple



2.2.4 – Exemple

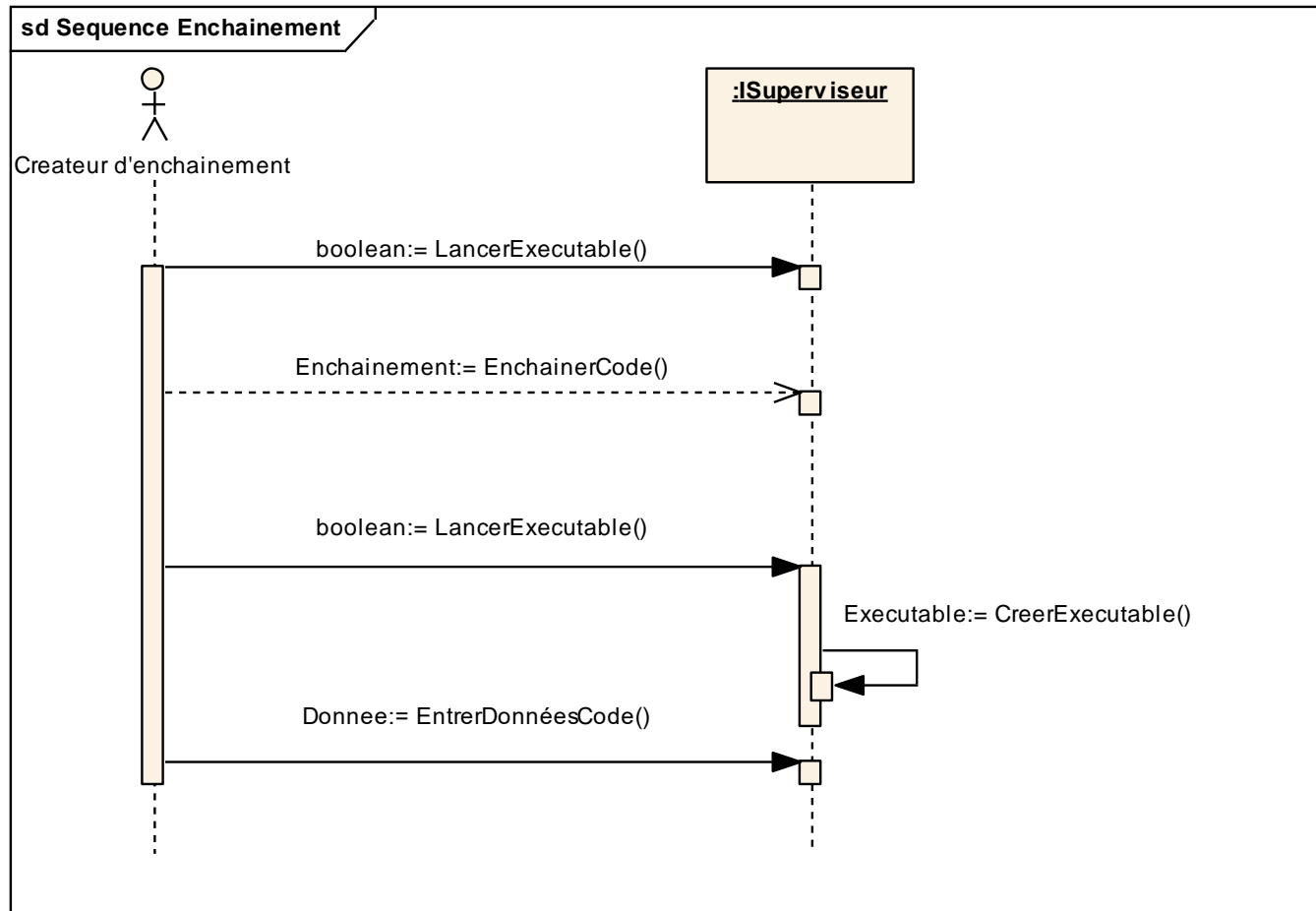


2.2.4 – Exemple

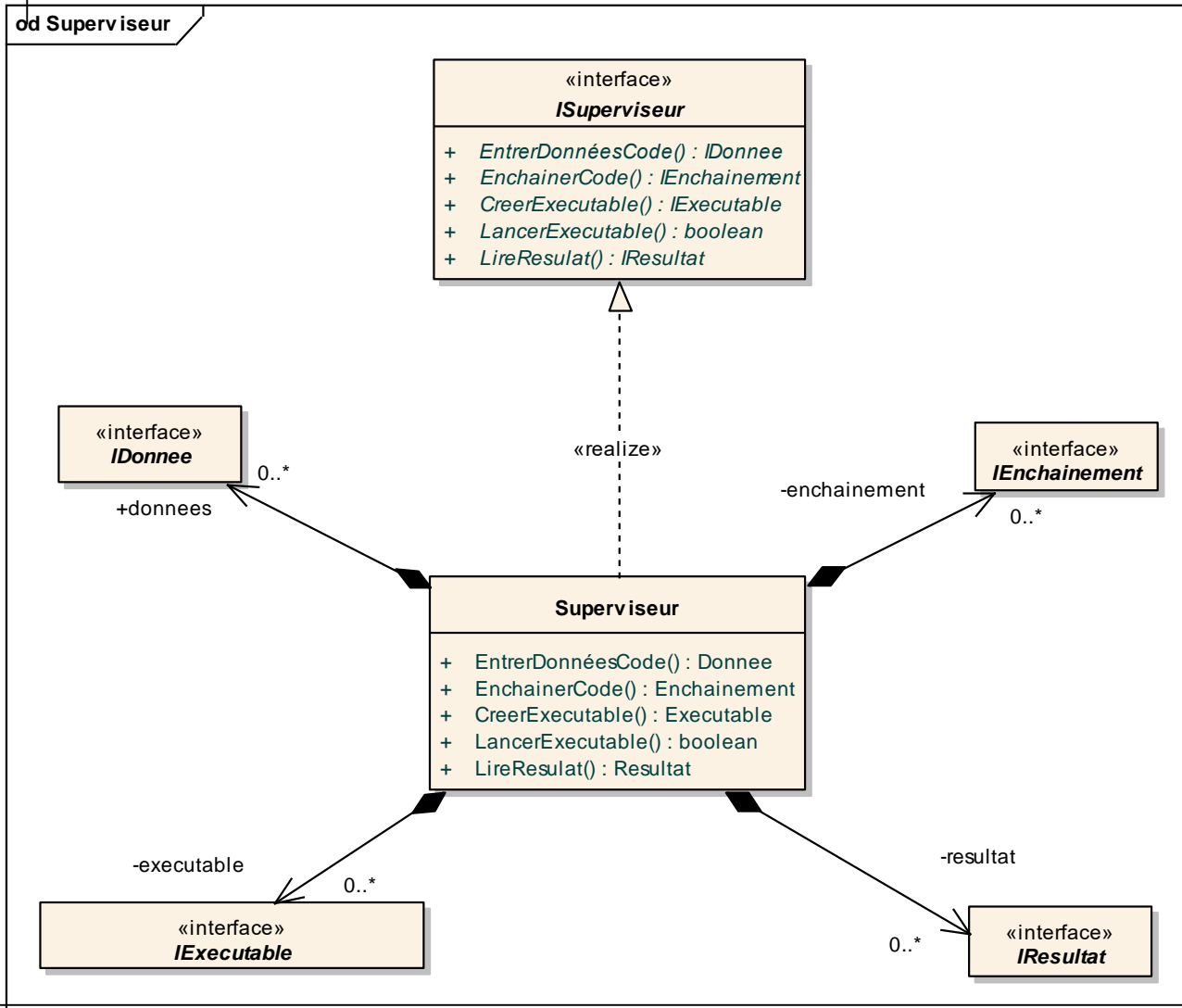


- Exemple : Séquence d'enchaînement
- Exemple : Classe Superviseur
- Exemple : Séquence de lancement d'exécutable
- Exemple : Classe Executable

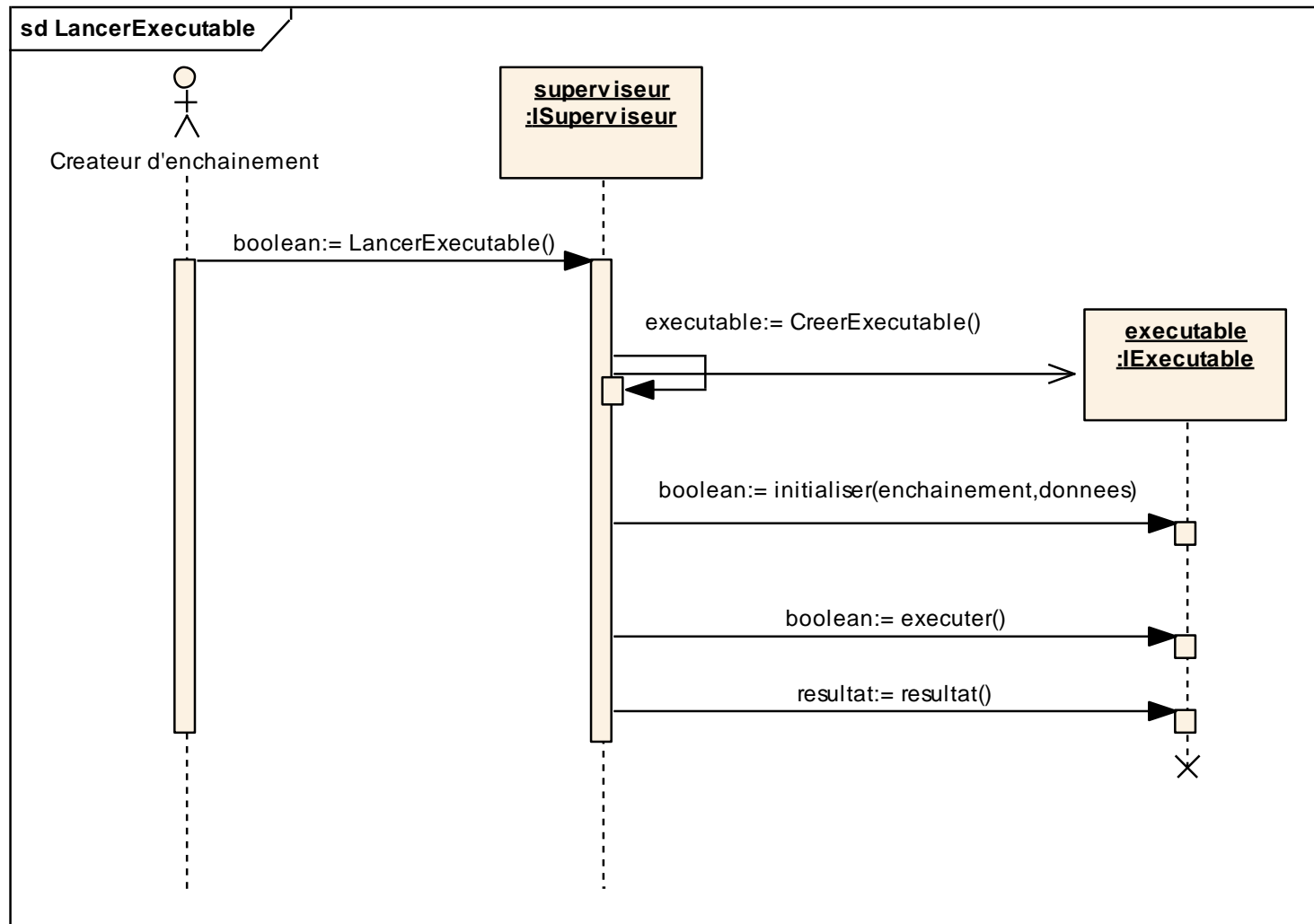
2.2.4.1 – Exemple : Séquence d'enchaînement



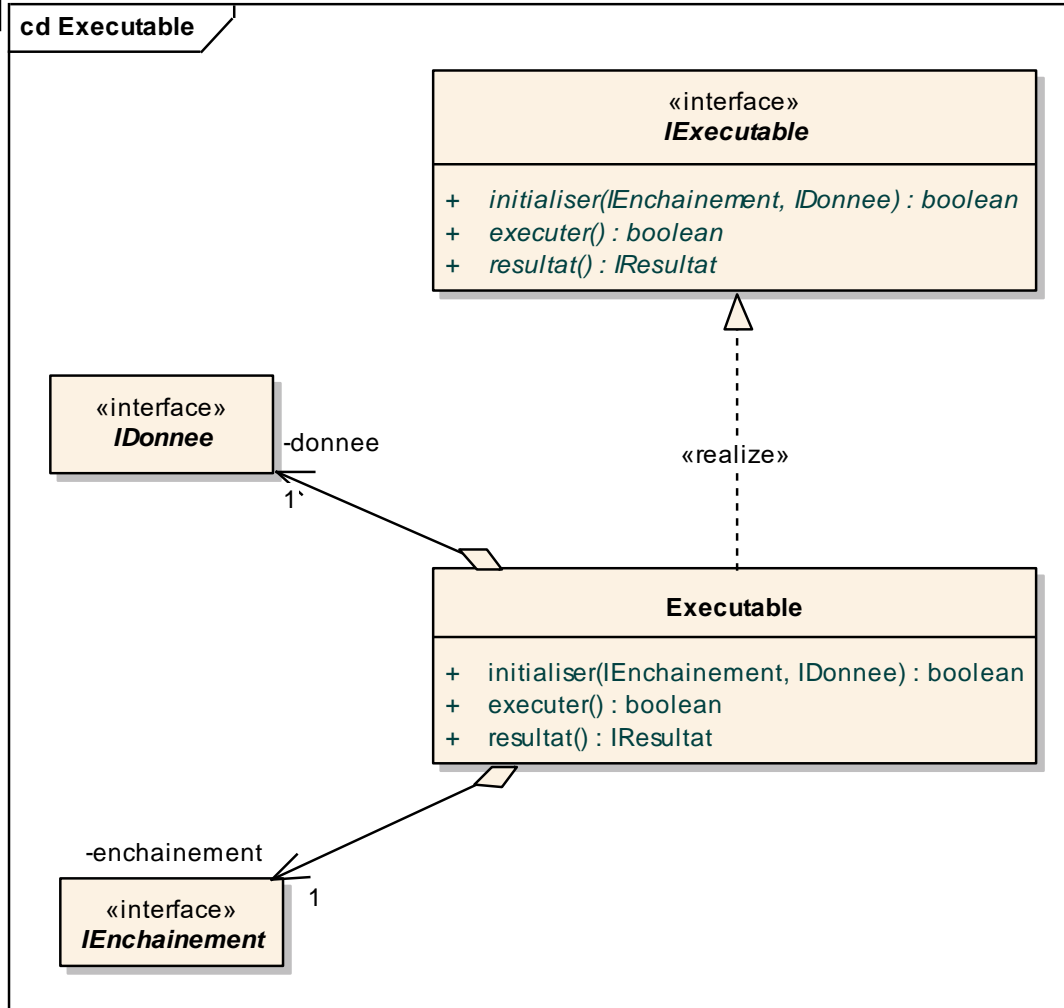
2.2.4.2 – Exemple : Classe Superviseur



2.2.4.3 – Exemple : Séquence de lancement d'exécutable



2.2.4.4 – Exemple : Classe Executable



2.3 - Le diagramme d'objet



- Fonctionnalité
- Exemple



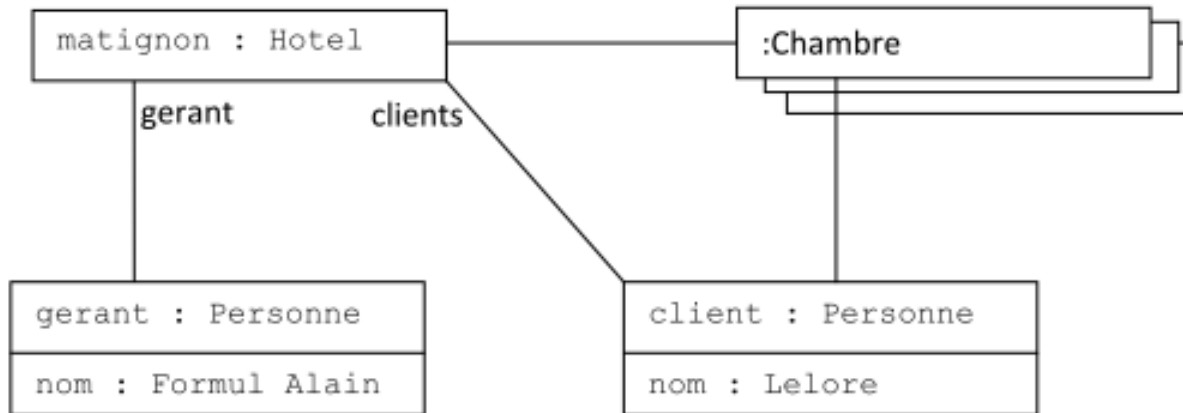
Le diagramme d'objet (d'instance) est un instantané des objets du système à un moment donné.

Il permet de :

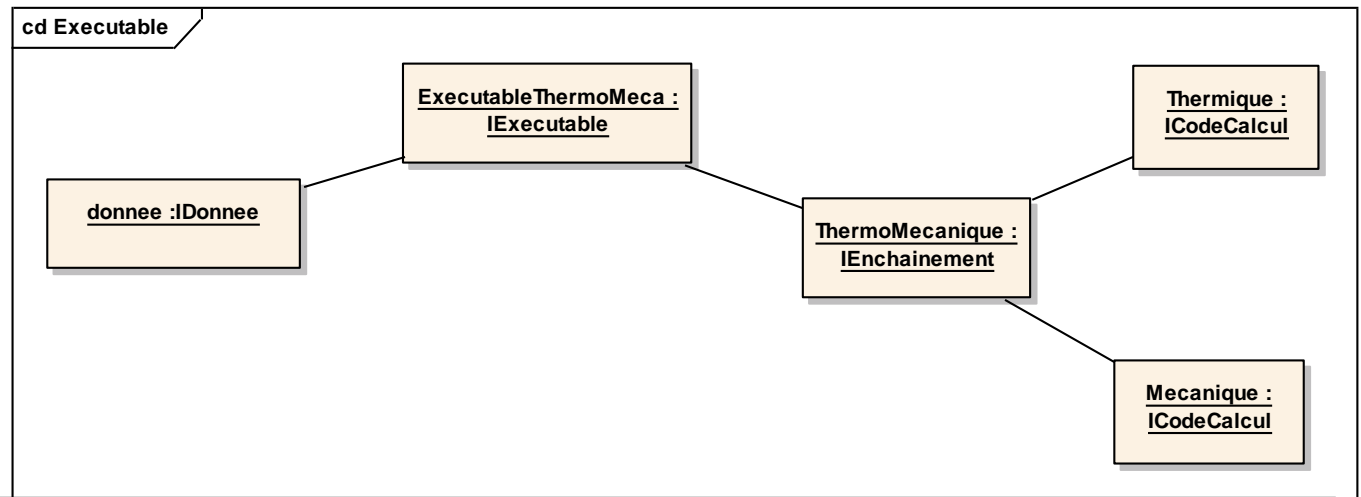
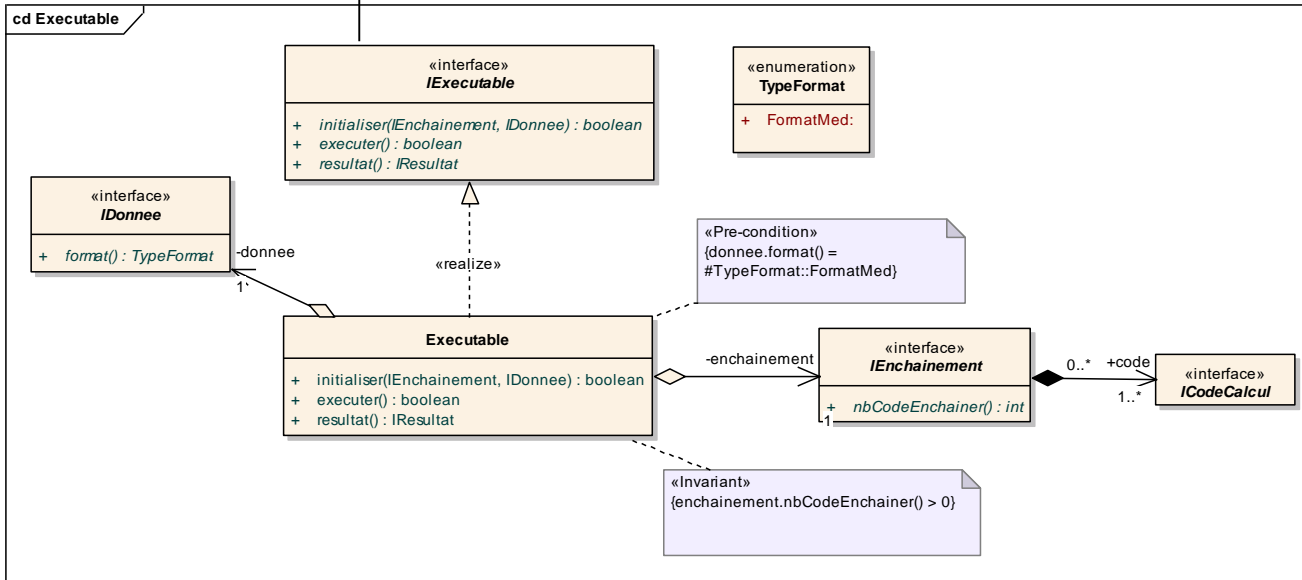
- Montrer des **objets** (instances de classes dans un état particulier) et des **liens** (relations sémantiques) entre ces objets.
- Montrer un **contexte** (avant ou après une interaction entre objets par exemple).
- Explorer des solutions dans les phases exploratoires du projet

Le diagramme d'objet permet d'illustrer un diagramme de classe par des exemples.

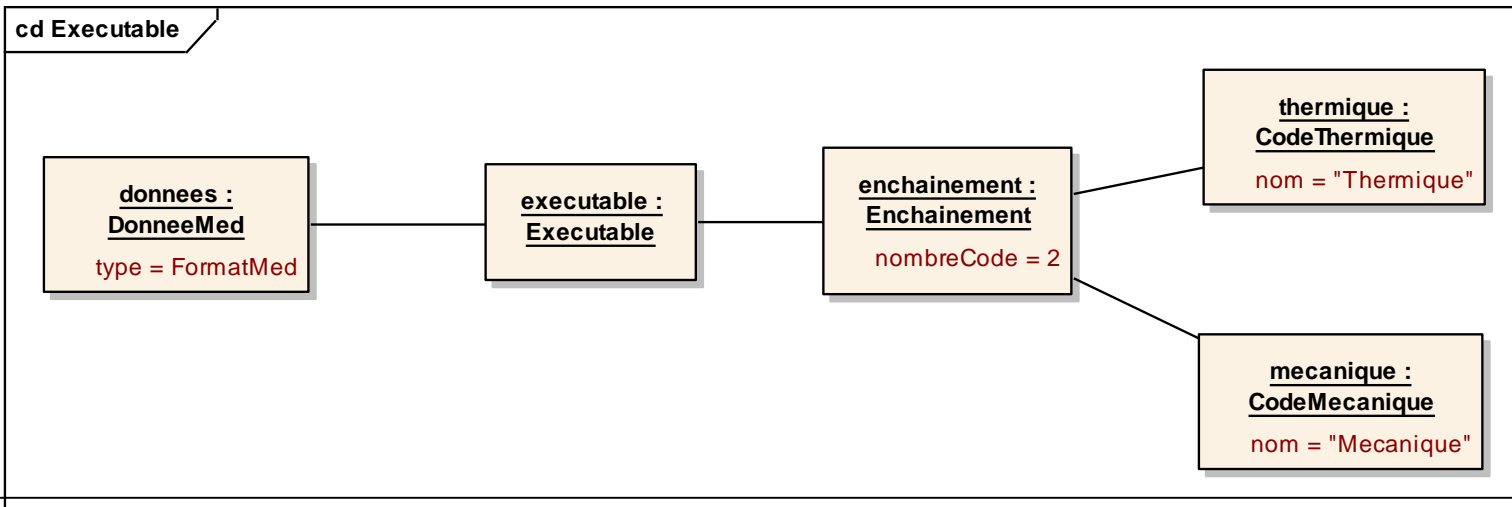
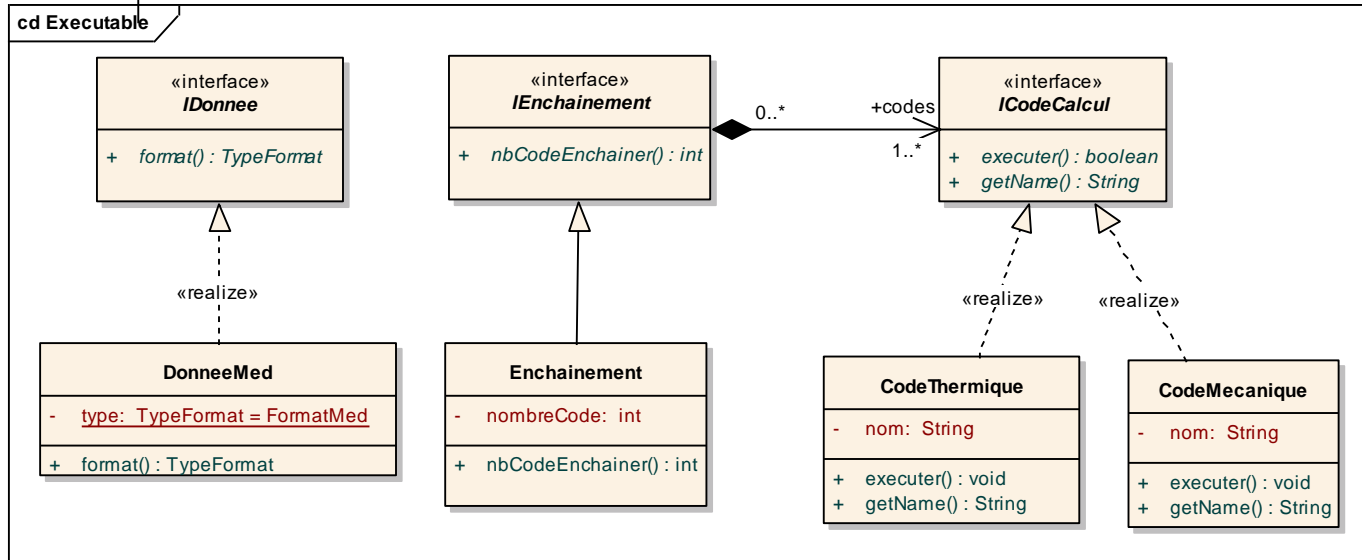
2.3.4.1 – Exemple



2.3.4.1 – Exemple : Exécutable (interface)



2.3.4.2 – Exemple : Exécutable



2.4 - Le diagramme de package



- Fonctionnalité
- Notation
- Mise en œuvre
- Exemple



Les **paquetages** sont des **éléments d'organisation** des modèles et servent de "briques" de base dans la construction d'une architecture.

Ils permettent de :

- Regrouper des éléments de n'importe quelle construction UML, selon des critères purement logiques
- Encapsuler des éléments de modélisation
- Structurer un système en catégories et sous-systèmes
- Représenter des espaces de nommage
- Voir ce qui est réutilisable

Un package en analyse = une catégorie

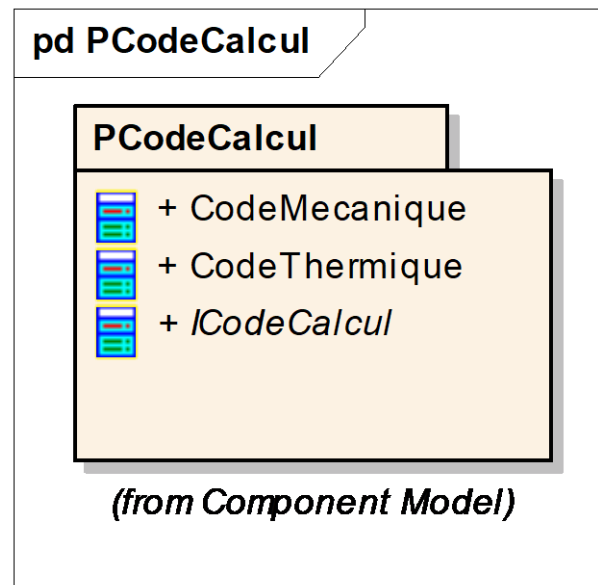
Un package en conception d'architecture = un sous-système



- Espaces de nommage
- Dépendance entre packages
- Généralisation



Tout élément contenu dans un package se distingue par son appartenance au package englobant



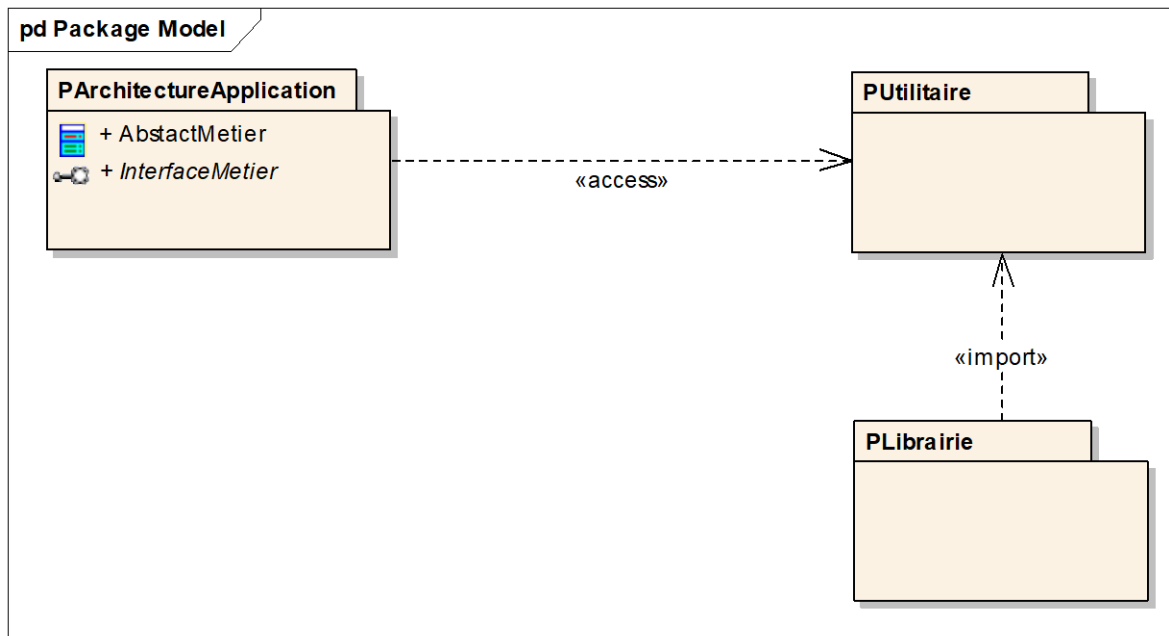
2.4.2.2 - Dépendance entre package



Les dépendances entre package représentent des relations entre packages et non entre éléments individuels

Stéréotypes standards associés :

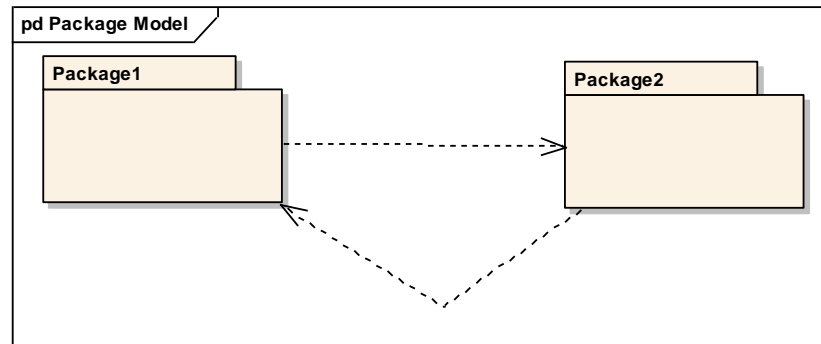
- « importe » : ajoute les éléments du package au package source
- « accede » : permet d'accéder à des éléments du package



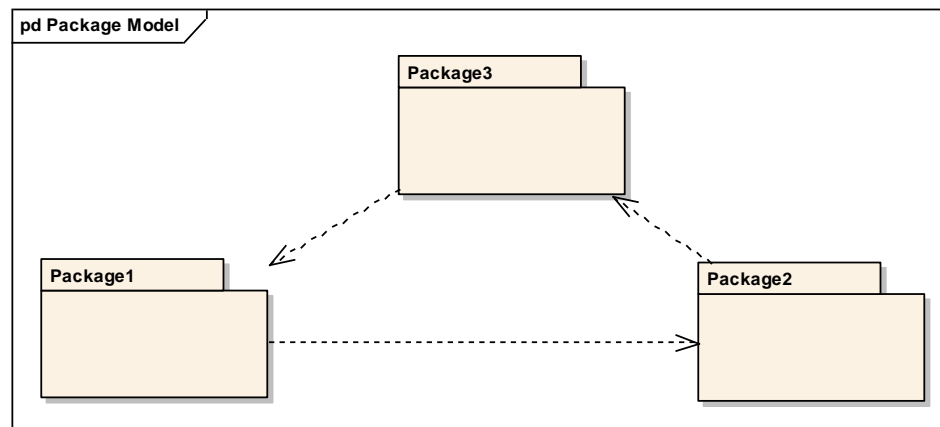


A éviter :

- Les dépendances circulaires

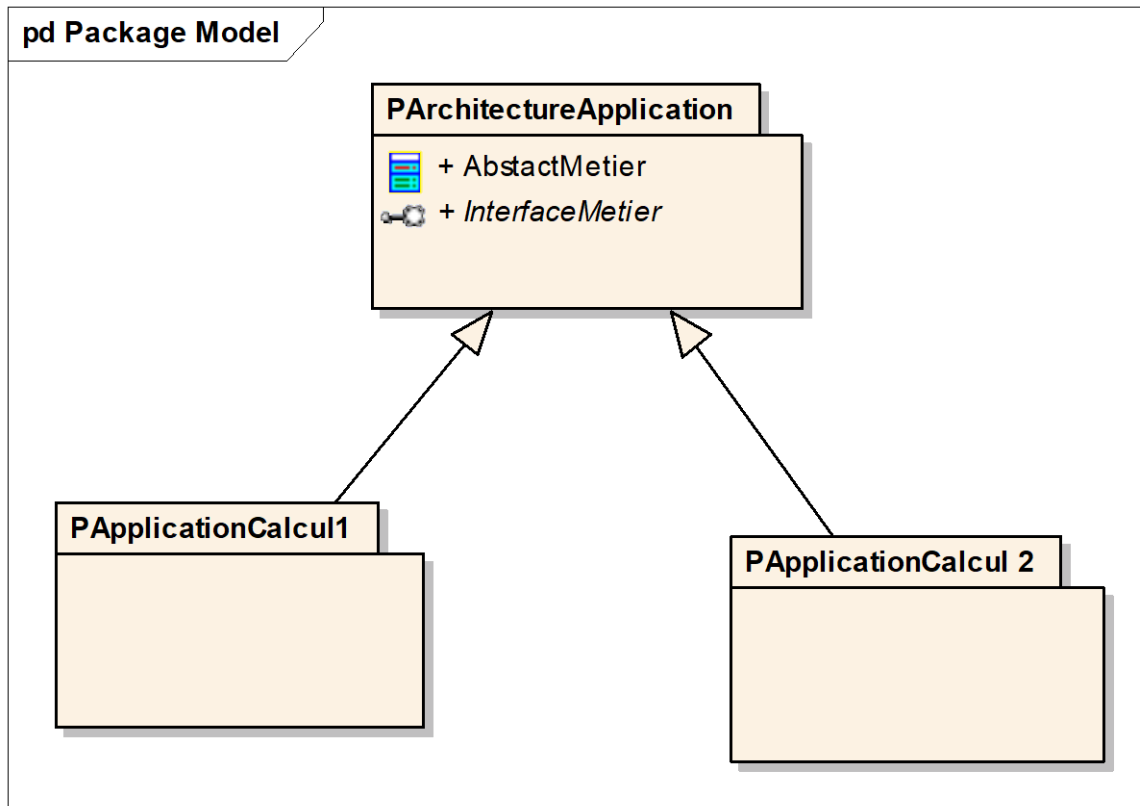


- Les dépendances circulaires transitives





La généralisation des packages est équivalente à la généralisation des classes





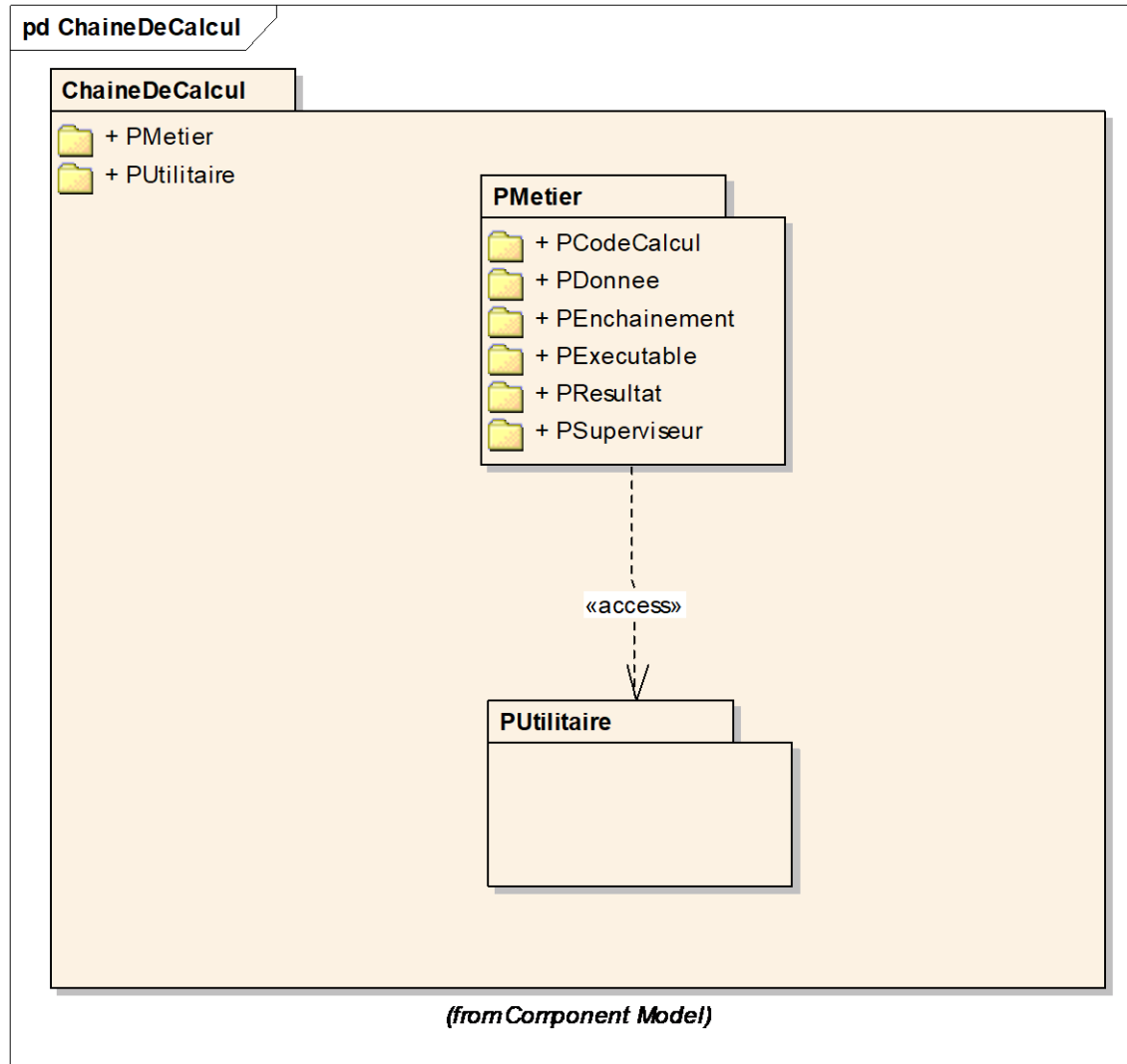
A utiliser pour regrouper les entités de modélisation afin d'avoir une image des dépendances, de les contrôler et les réduire

2.4.4 - Exemple

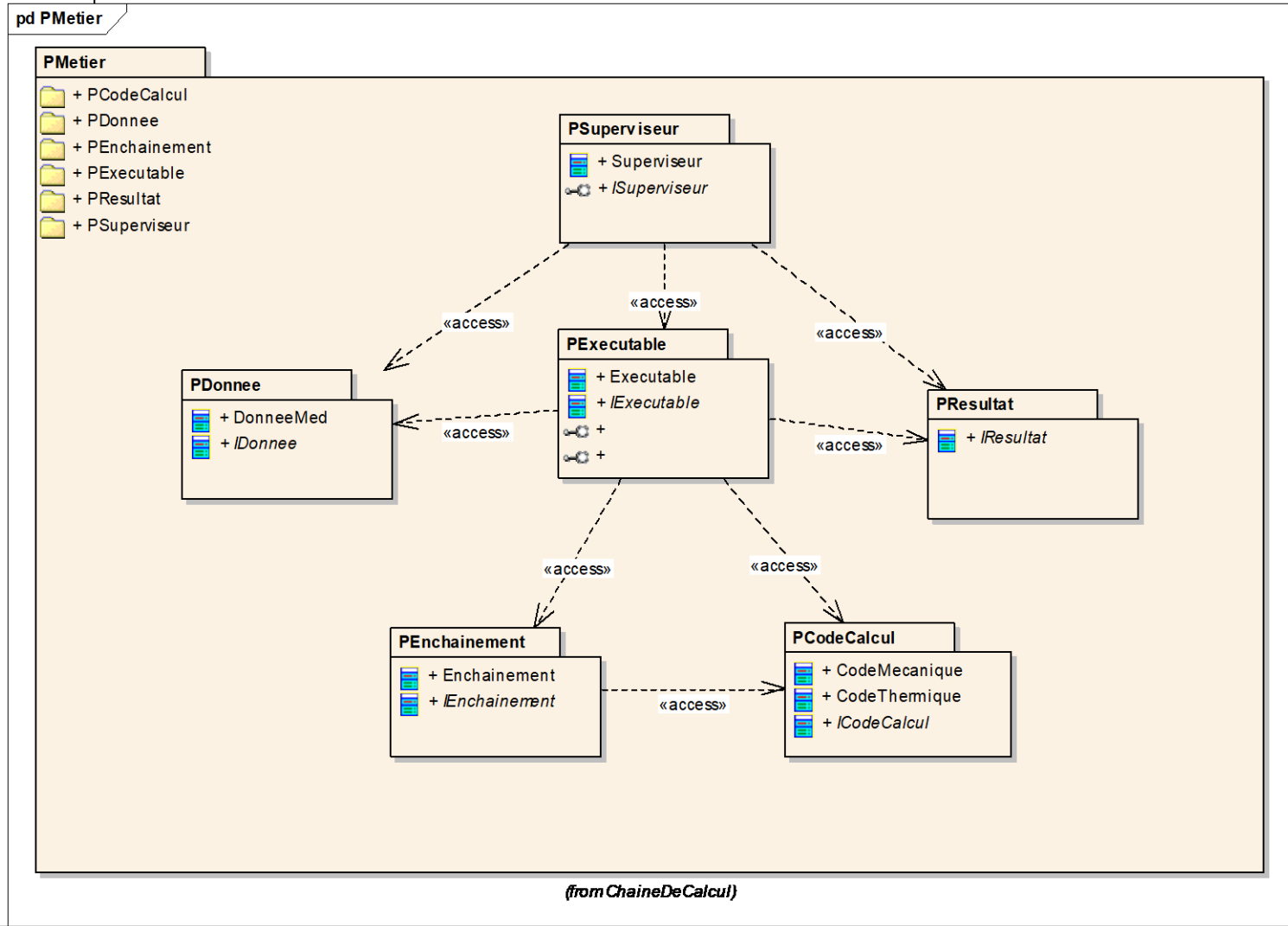


- Exemple : Package chaîne de calcul
- Exemple : Package métier

2.4.4.1 – Exemple : Package Chaîne de calcul



2.4.4.2 – Exemple : Package métier



2.5 - Le diagramme de séquence



- Fonctionnalité
- Notation
- Mise en œuvre
- Exemple



Le diagramme de séquence est une vue dynamique qui permet de :

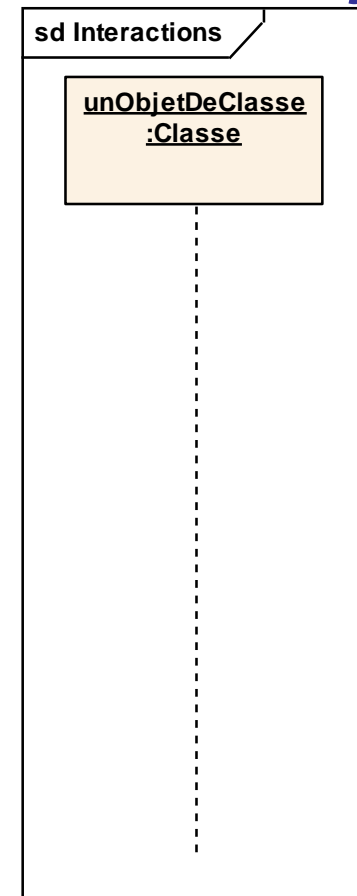
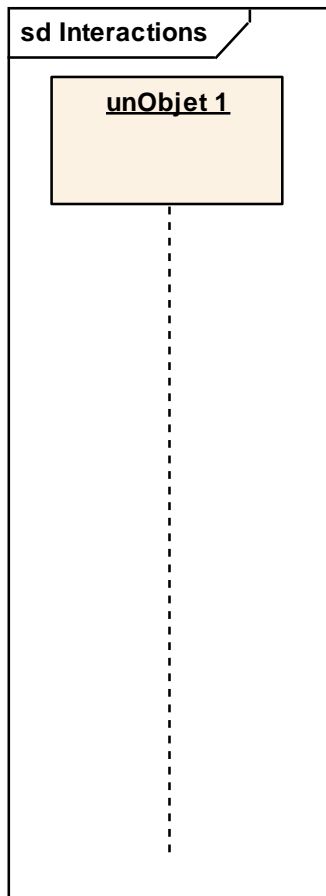
- Décrire la succession chronologique d'opérations réalisées par un acteur
- Décrire le contexte ou l'état des objets
- Décrire les différents scénarios d'un cas d'utilisation
- Exposer le comportement complexe d'une interaction (classe ou processus)
- Mettre en avant les contraintes temporelles
- Indiquer les objets que l'acteur va manipuler, et les opérations qui font passer d'un objet à un autre



- Interaction
- Activations et envois de message
- Contrainte temporelle
- Ligne de vie des objets
- Cadre d'interaction
- Opérateurs courants des cadres d'interaction
- Appel synchrone et asynchrone

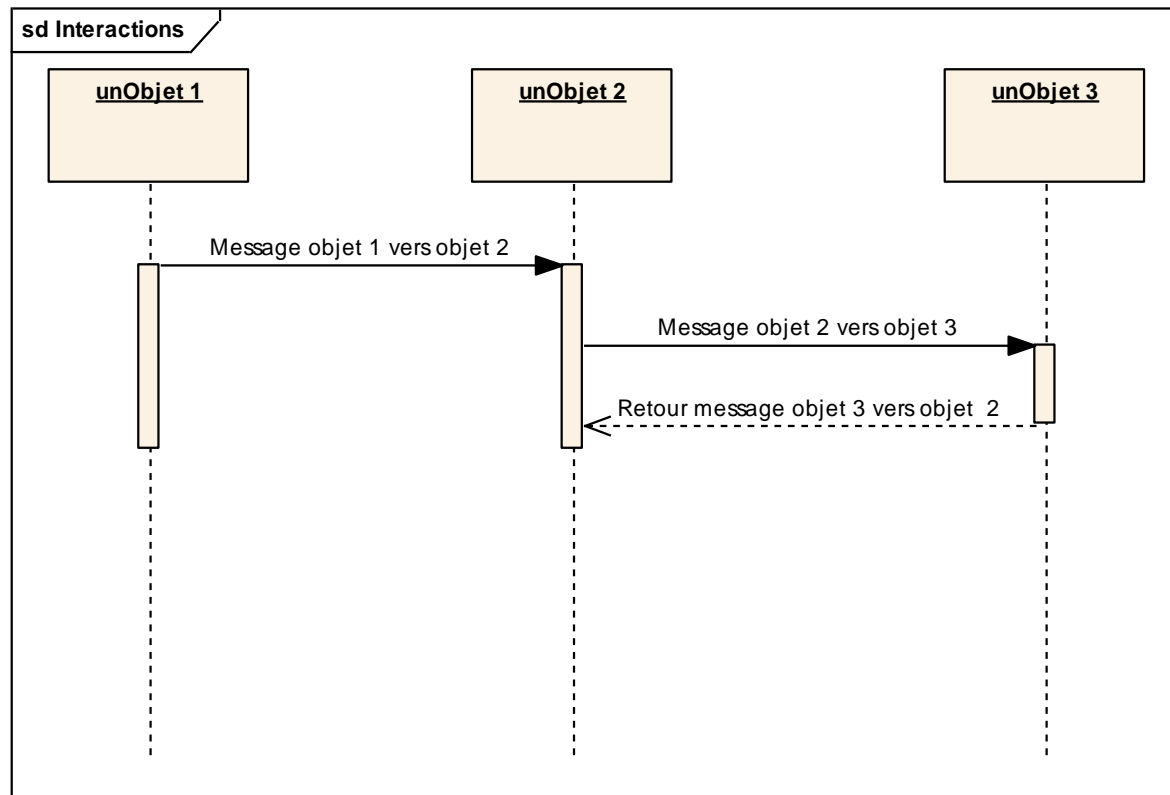


Représentation graphique d'un Objet

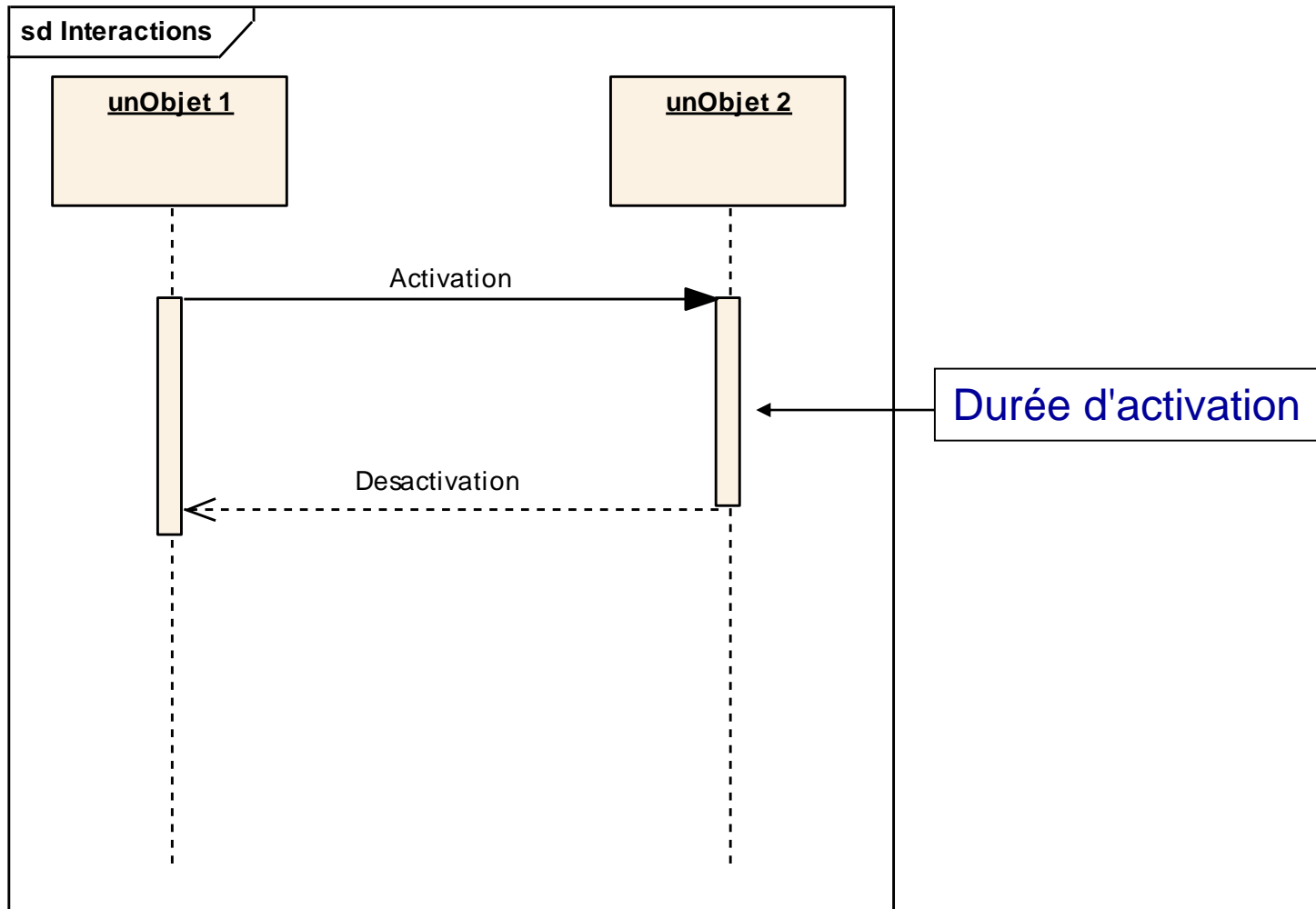




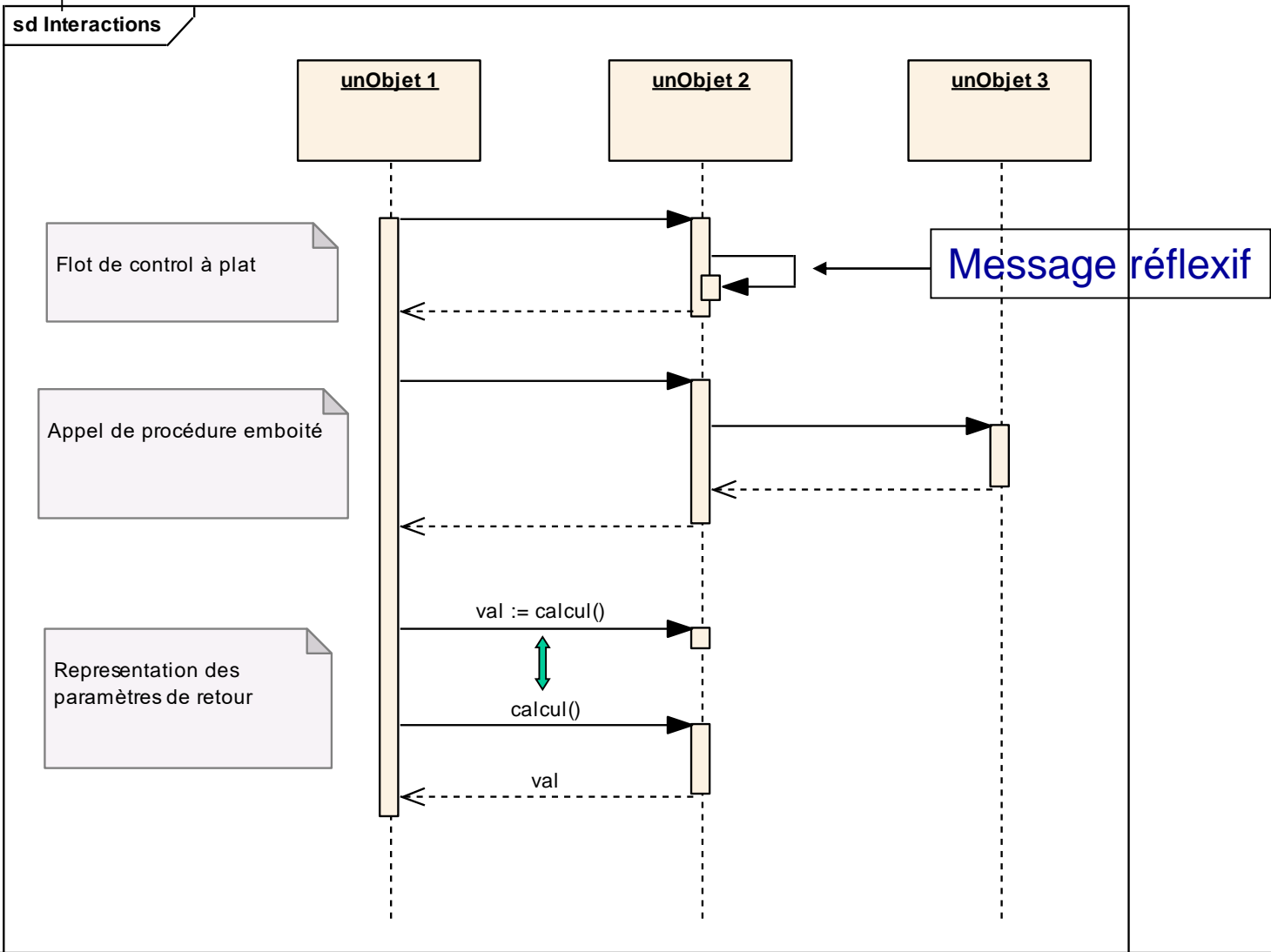
Exemple de diagramme de séquence



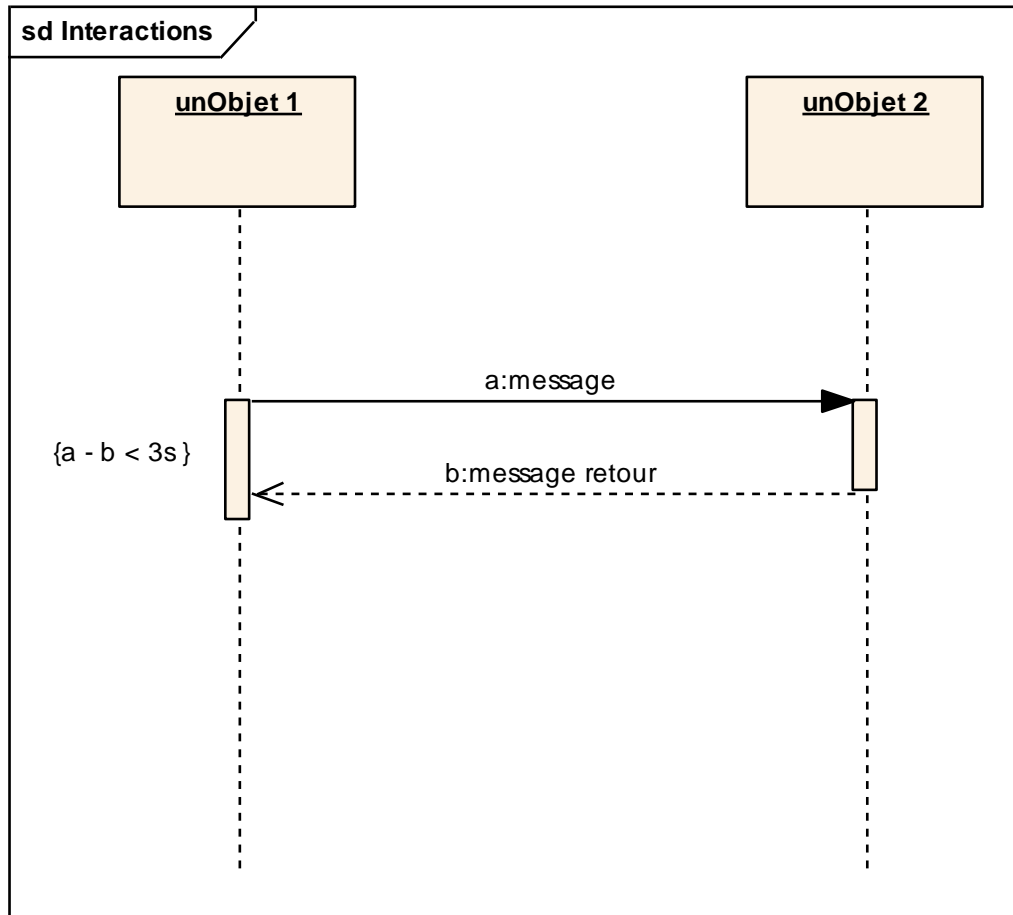
2.5.2.2 - Activations et envois de message



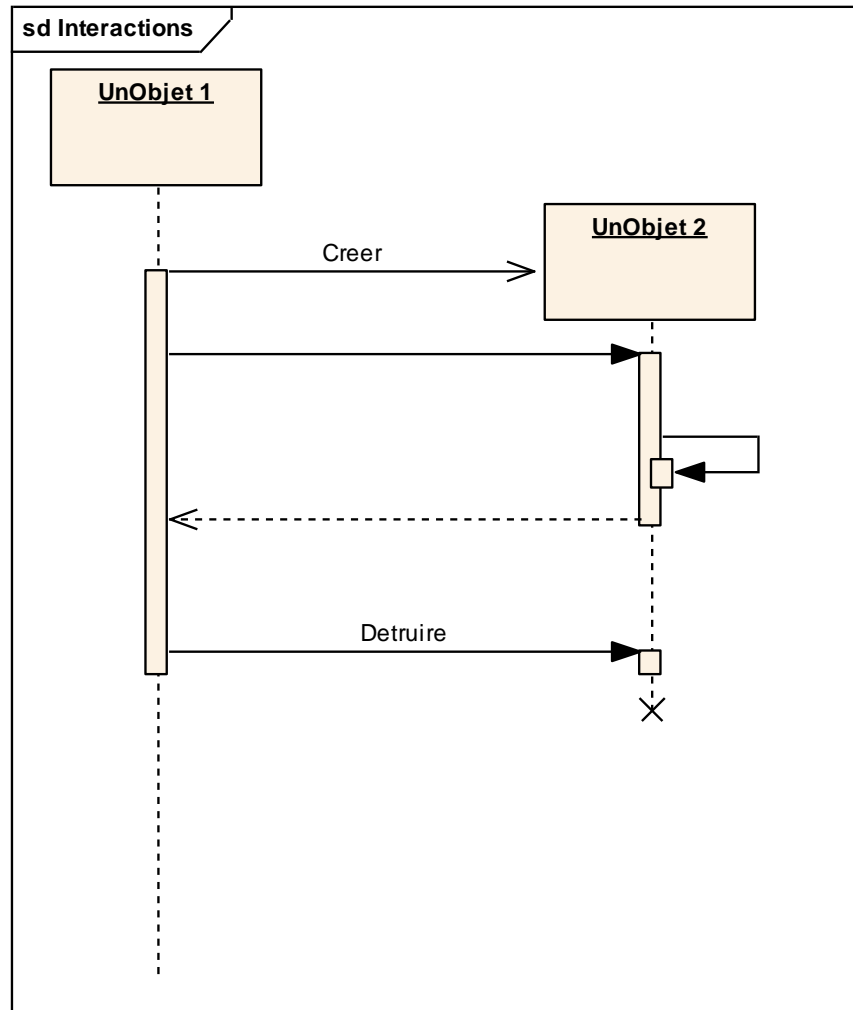
2.5.2.2a - Activations et envois de message



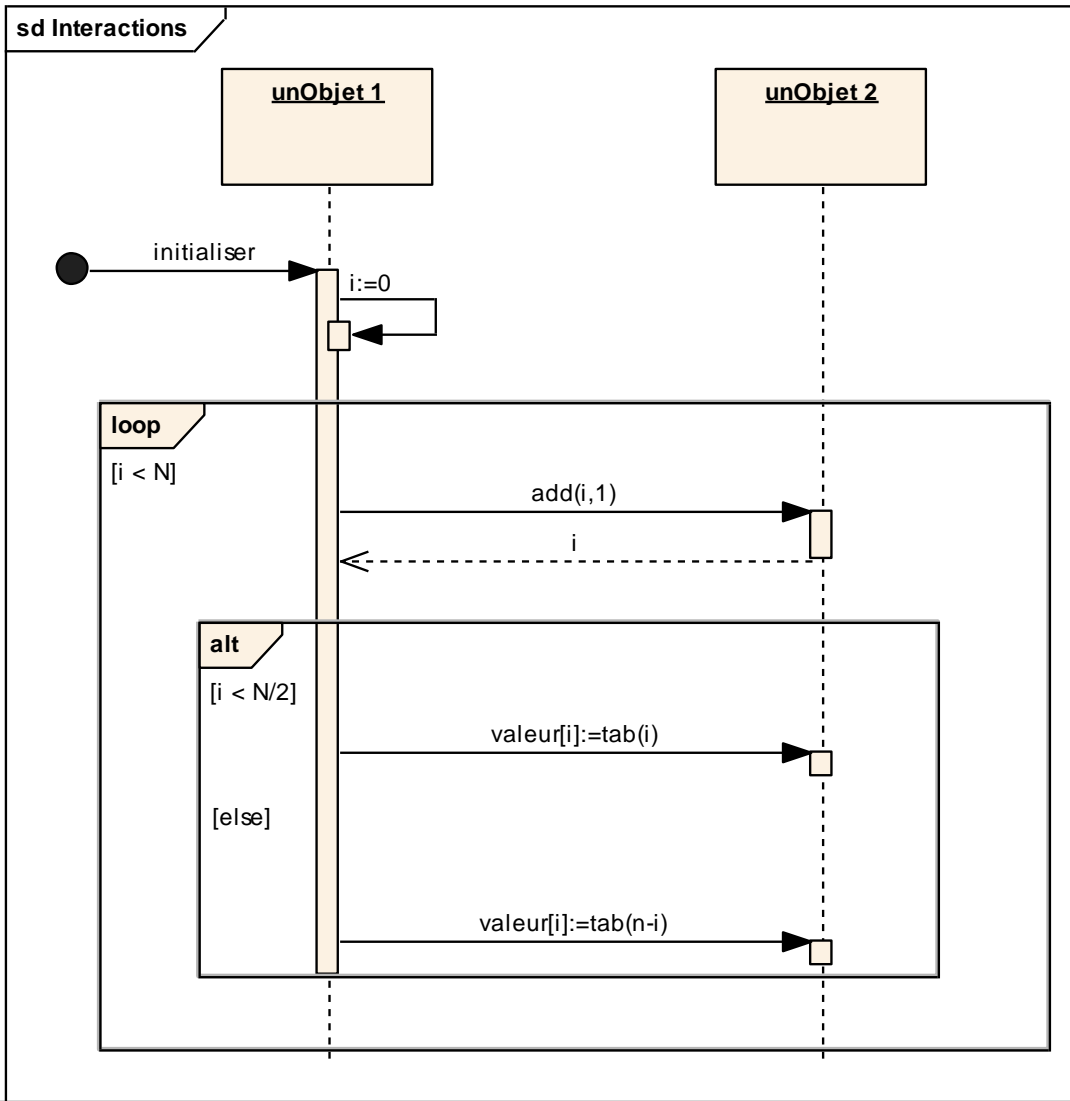
2.5.2.3 - Contrainte temporelle



2.5.2.4 - Ligne de vie des objets



2.5.2.5 - Cadre d'interaction



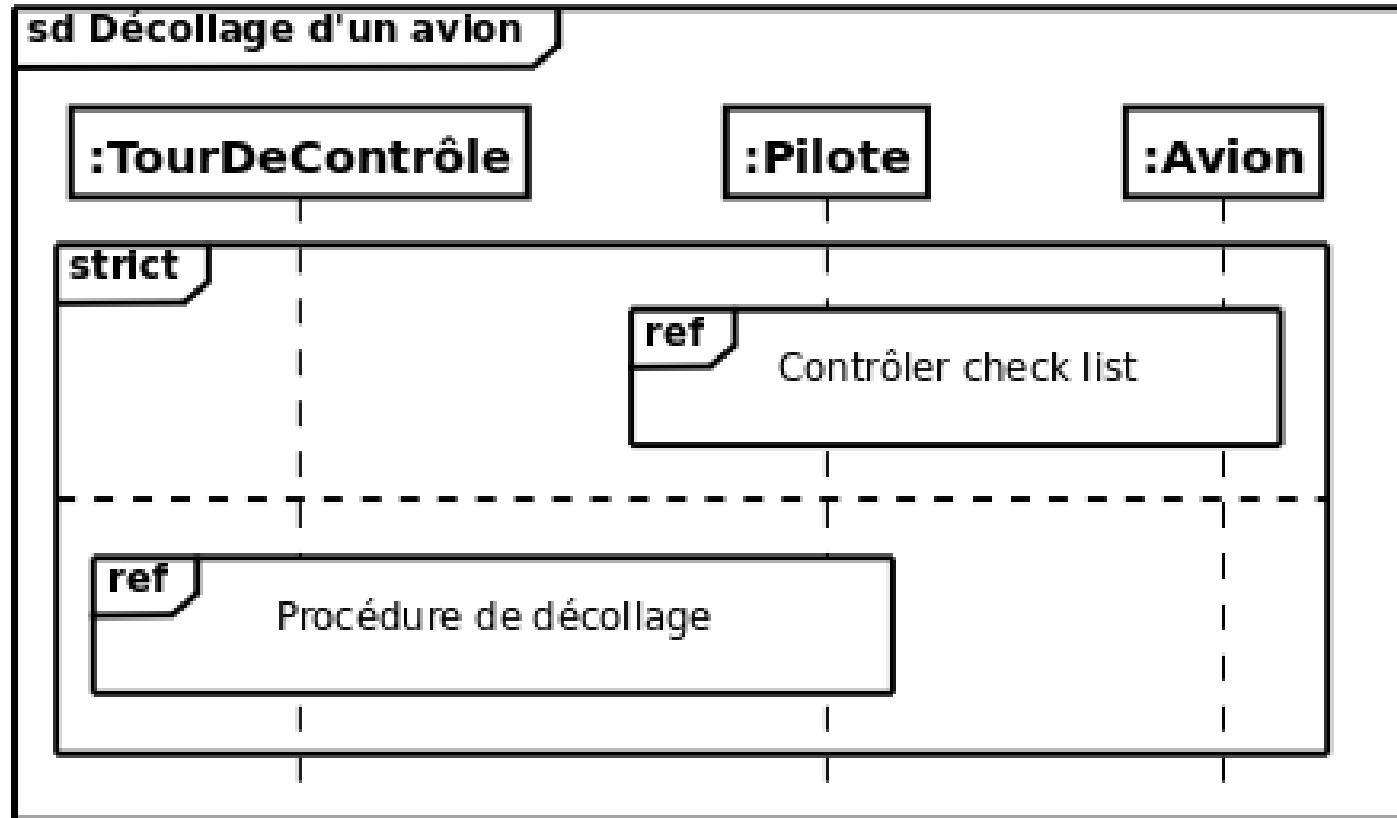
2.5.2.6 - Opérateurs courants des cadres d'interaction



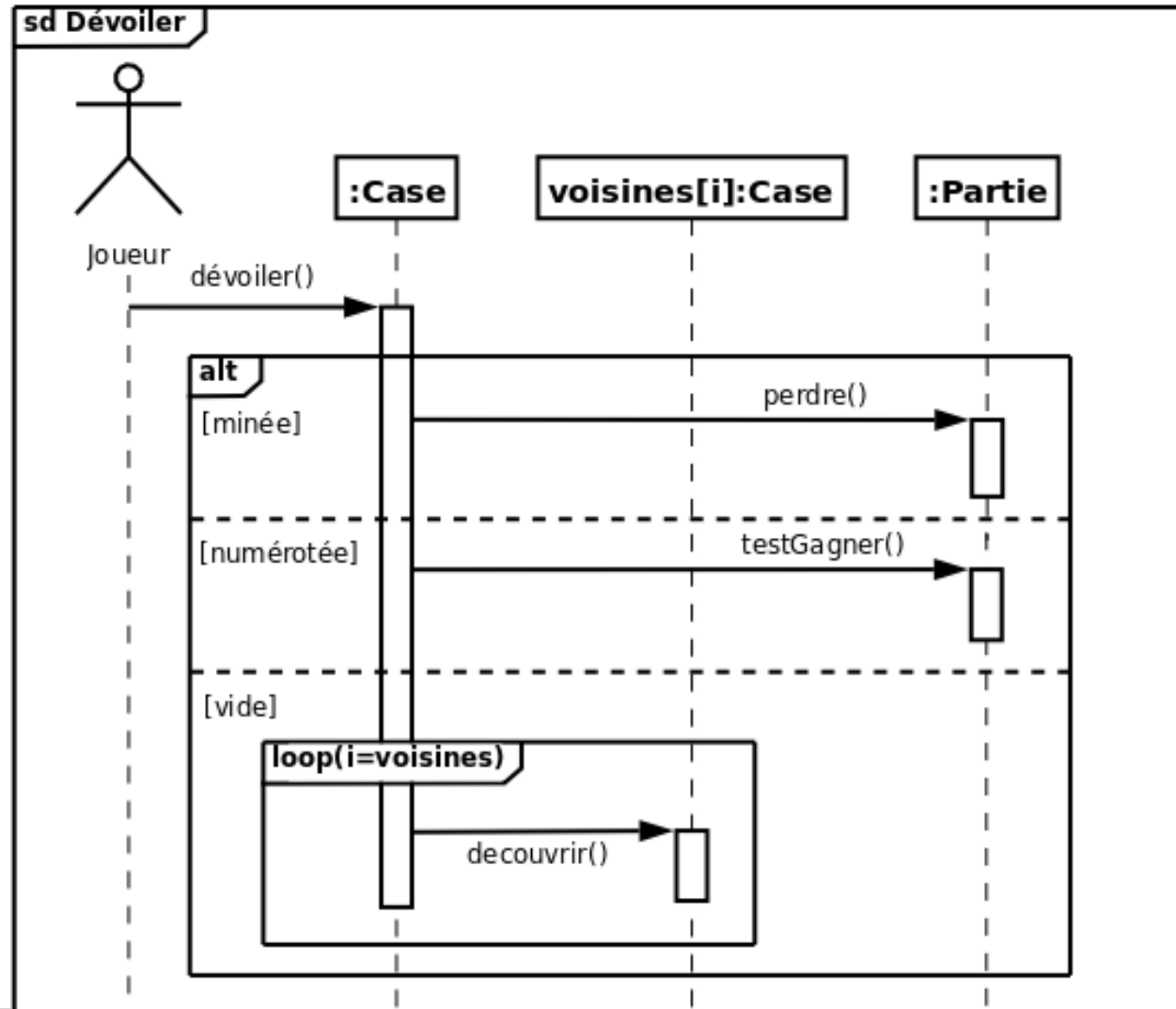
alt	Alternative
opt	Optionnel
par	Parallèle
loop	boucle
region	Région critique
neg	Négatif
ref	Référence à un autre diagramme
sd	Diagramme de séquence
break	Exception
critical	Section critique

- les opérateurs de choix et de boucle : **alternative**, **option**, **break** et **loop** ;
- les opérateurs contrôlant l'envoi en parallèle de messages : **parallel** et **critical region** ;
- les opérateurs contrôlant l'envoi de messages : **ignore**, **consider**, **assertion** et **negative** ;
- les opérateurs fixant l'ordre d'envoi des messages : **weak sequencing** , **strict sequencing**.

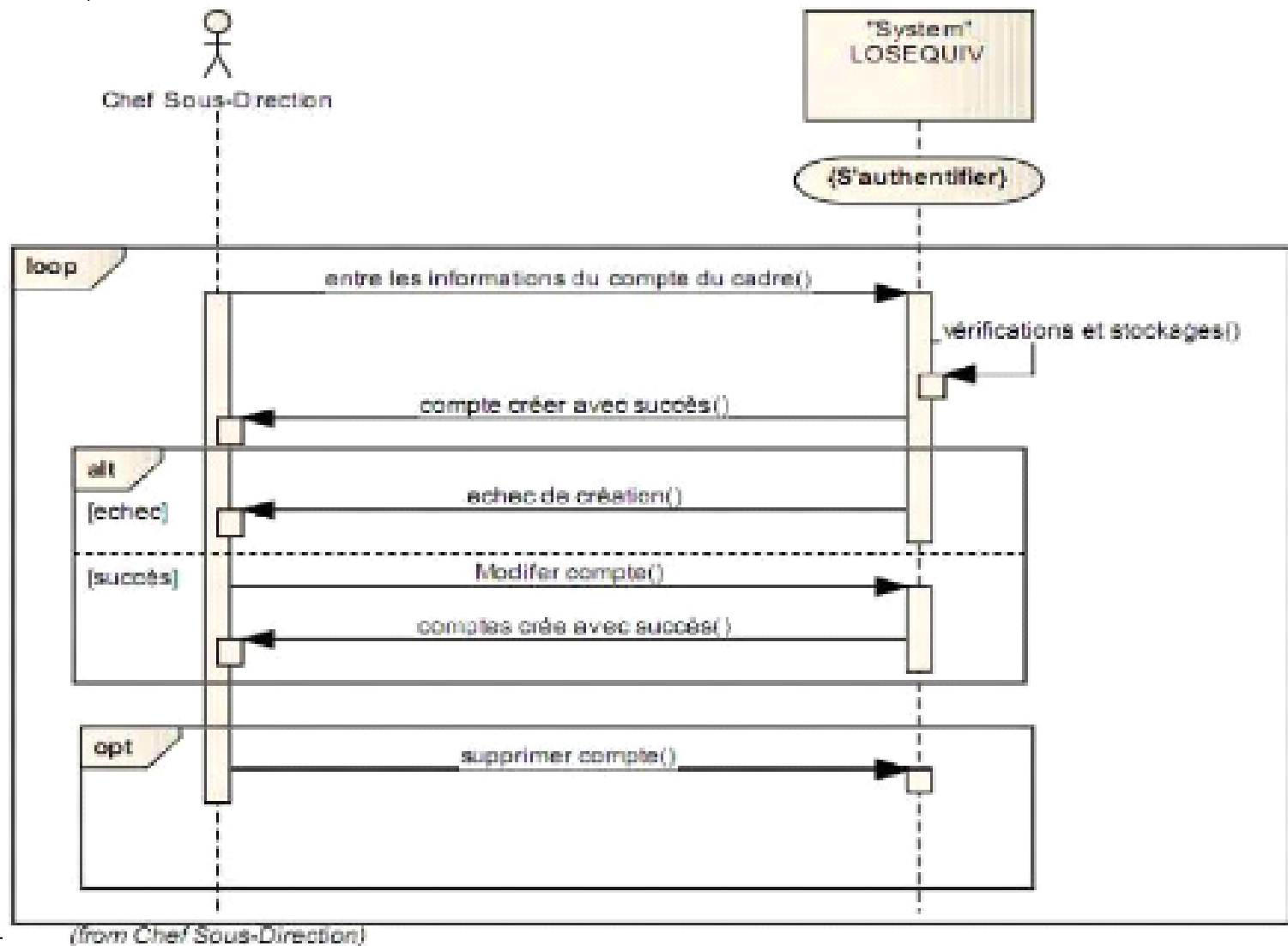
2.5.2.6 - Opérateurs courants des cadres d'interaction



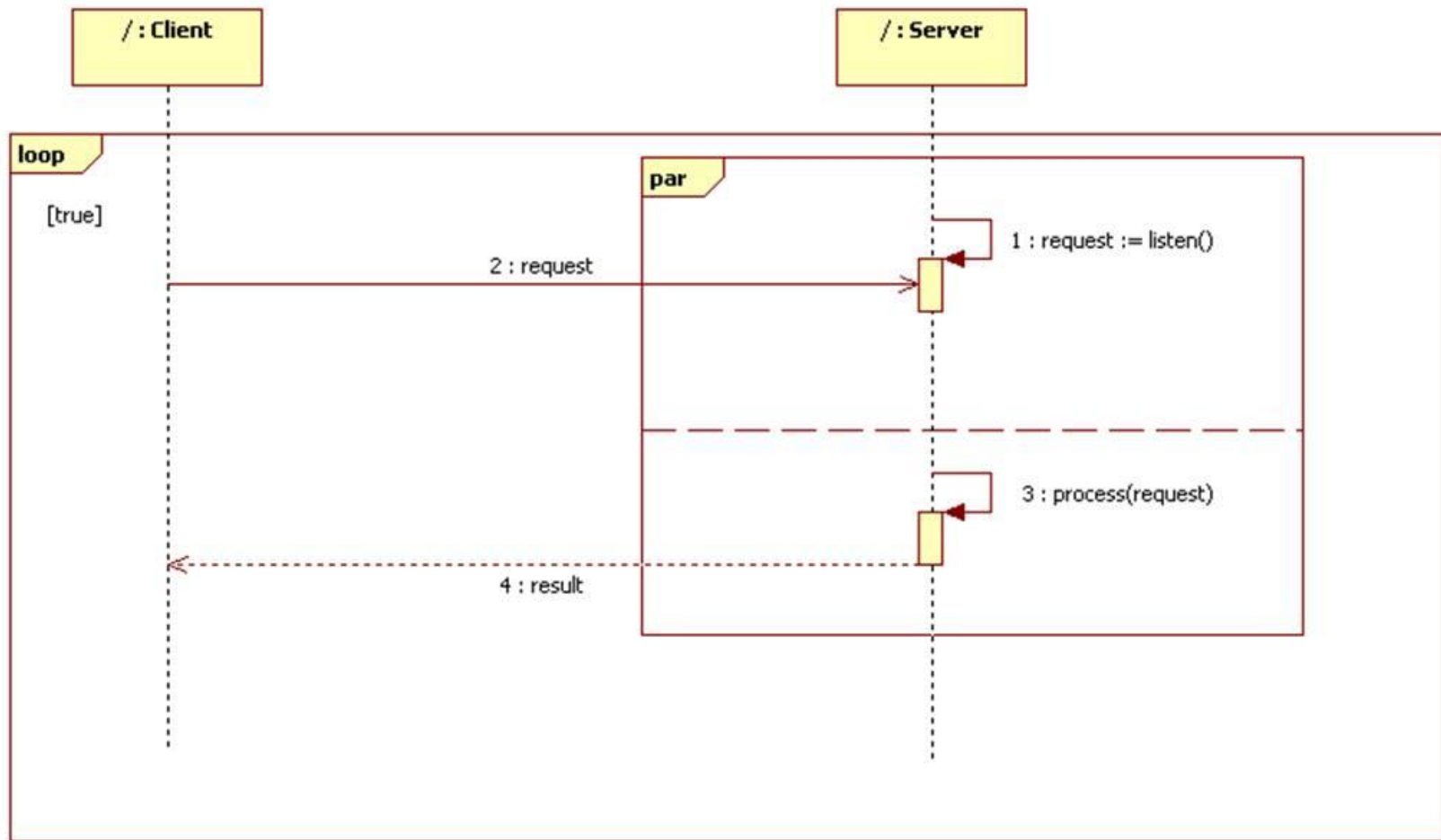
2.5.2.6 - Opérateurs courants des cadres d'interaction



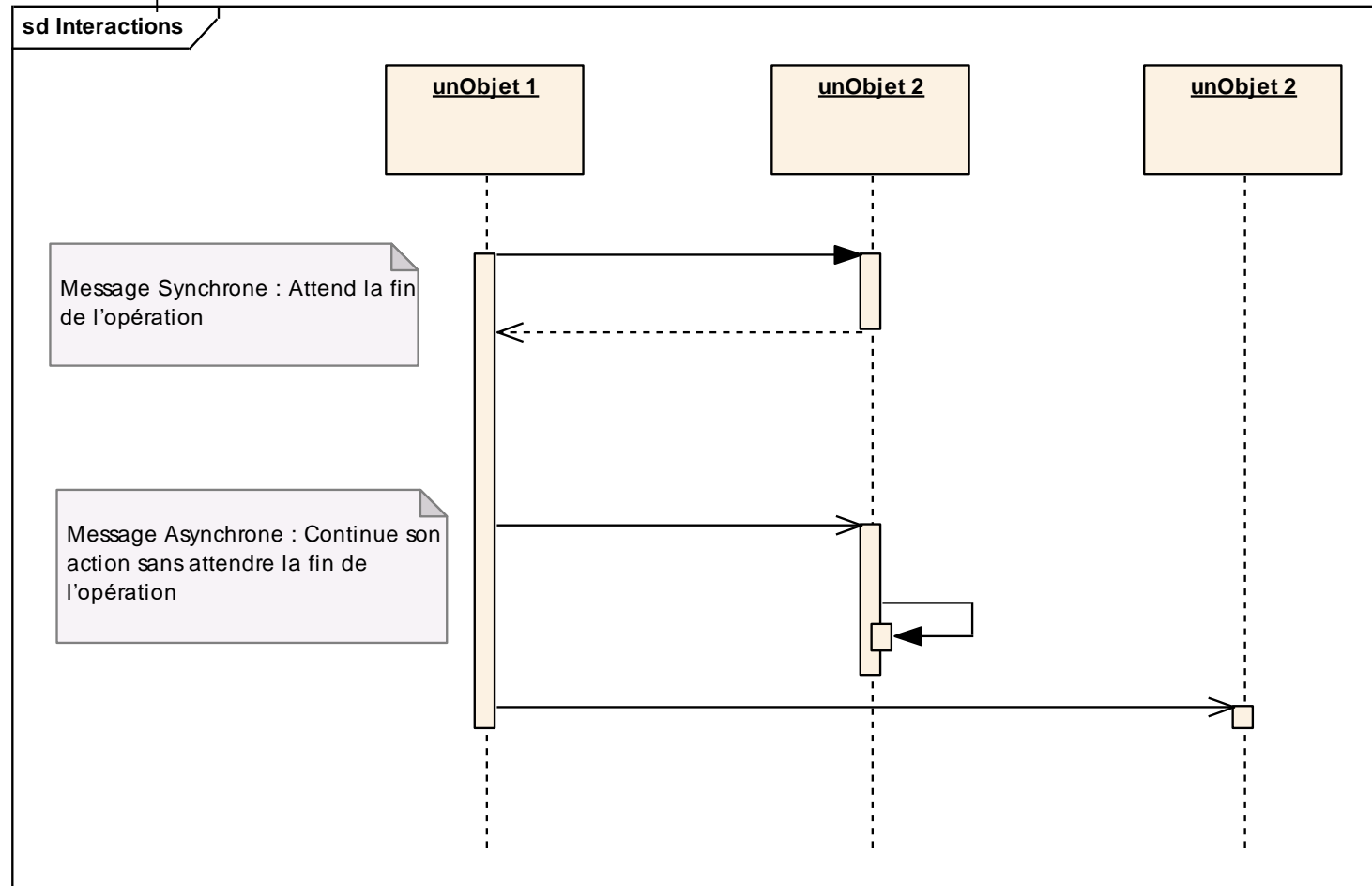
2.5.2.6 - Opérateurs courants des cadres d'interaction



2.5.2.6 - Opérateurs courants des cadres d'interaction



2.5.2.7 - Appel synchrone et asynchrone



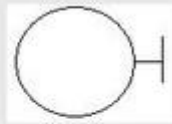
2.5.2.8 – Interface/Contrôleur/Entité



• Inclus dans UML :



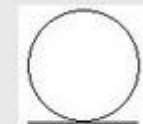
Acteur



Interface
boundary



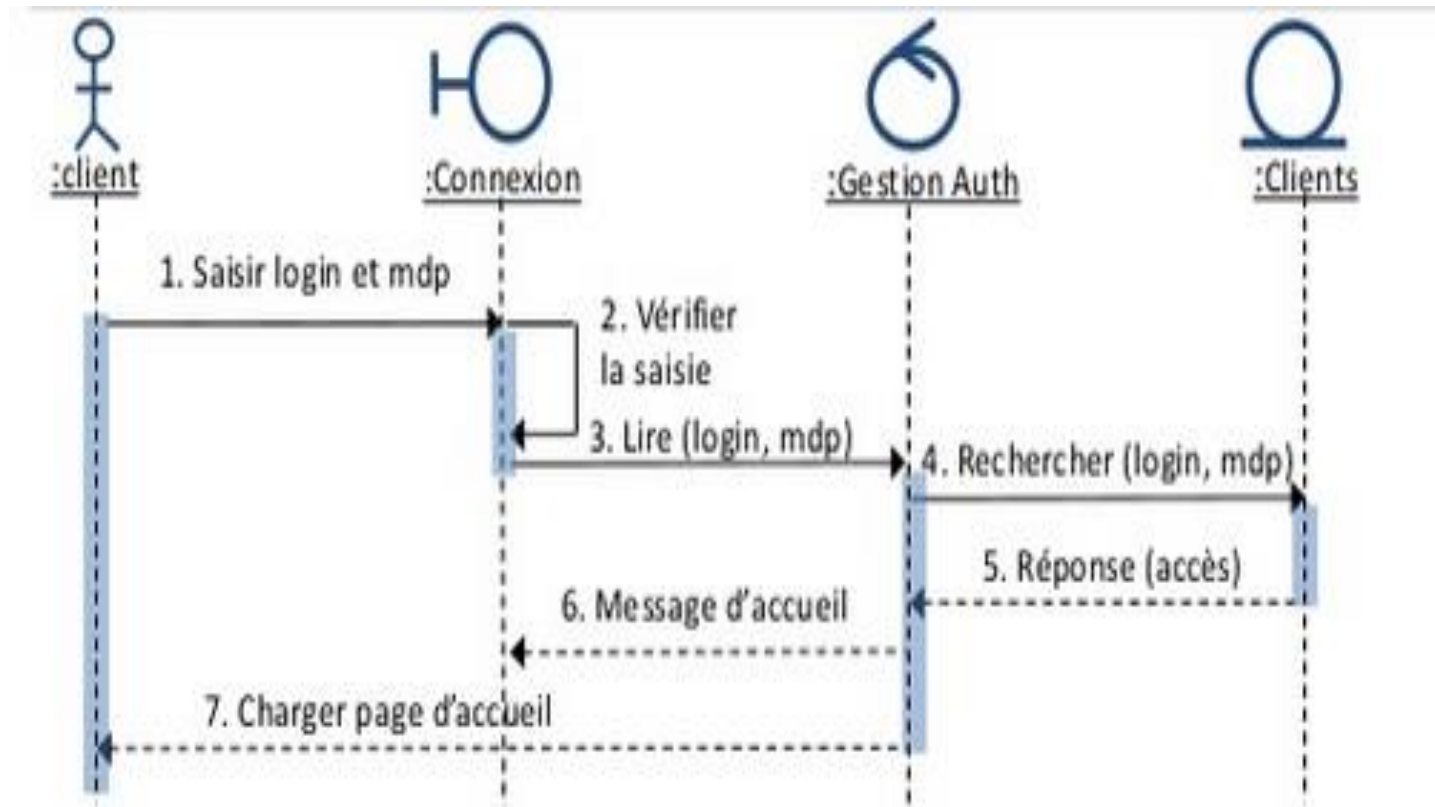
Contrôleur
Controller



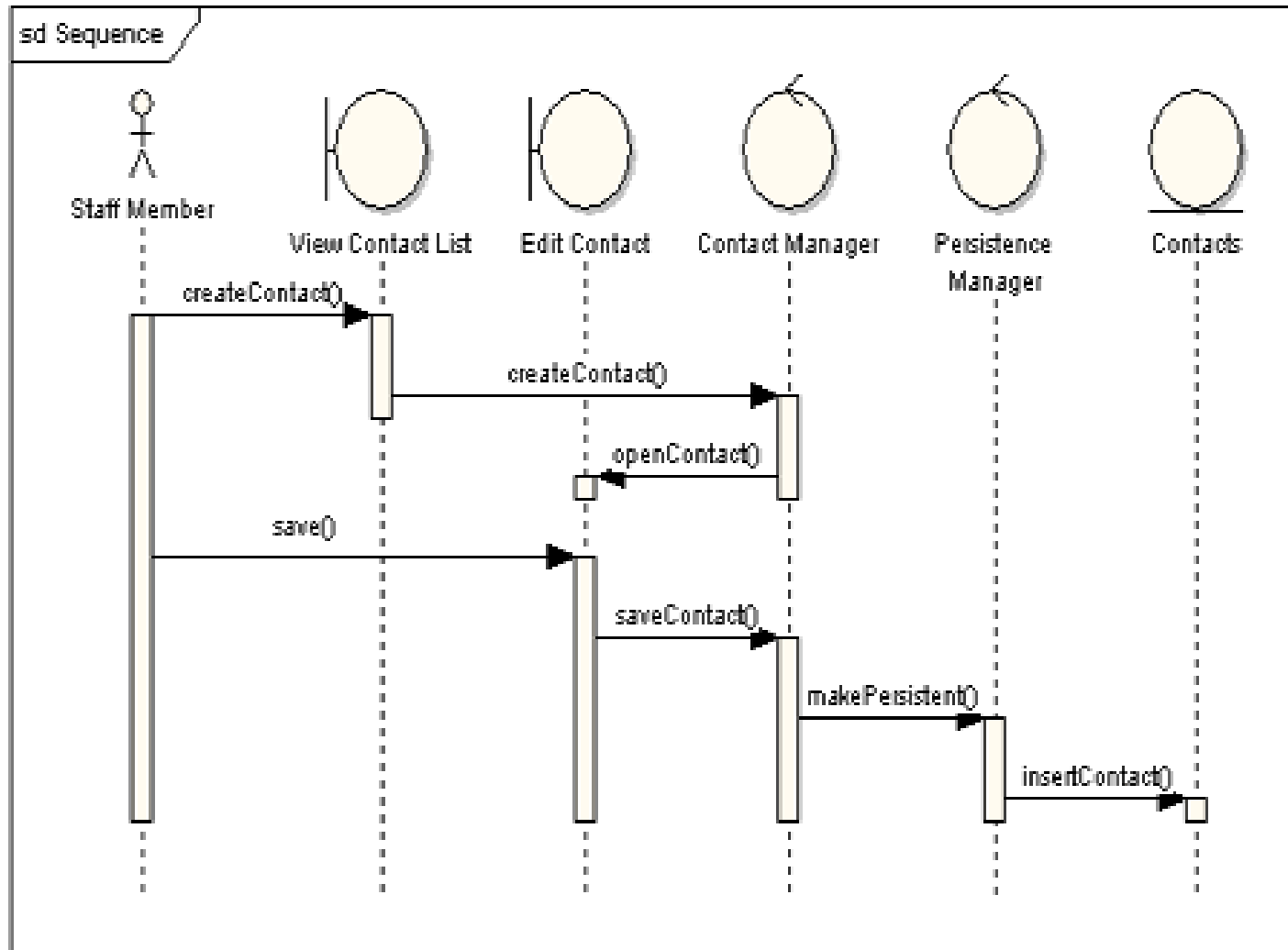
Entité
persistante
entity

- **Entities** (*modèle*)
Objets représentant des données système, souvent issues du modèle de domaine.
- **Boundaries** (*vue / service collaborateur*)
Objets qui s'interfaçent avec les acteurs du système (par exemple un **utilisateur** ou un **service externe**). Les fenêtres, les écrans et les menus sont des exemples de limites qui s'interfaçent avec les utilisateurs.
- **Contrôle**(*contrôleur*)
Objets qui interviennent entre les limites et les entités. Ceux-ci servent de lien entre les éléments de frontière et les éléments d'entité, mettant en œuvre la logique requise pour gérer les différents éléments et leurs interactions. Il est important de comprendre que vous pouvez décider d'implémenter des contrôleurs dans votre conception comme autre chose que des objets - de nombreux contrôleurs sont assez simples pour être implémentés en tant que méthode d'une entité ou d'une classe de limites par exemple

2.5.2.8 – Interface/Contrôleur/Entité



2.5.2.8 – Interface/Controleur/Entite

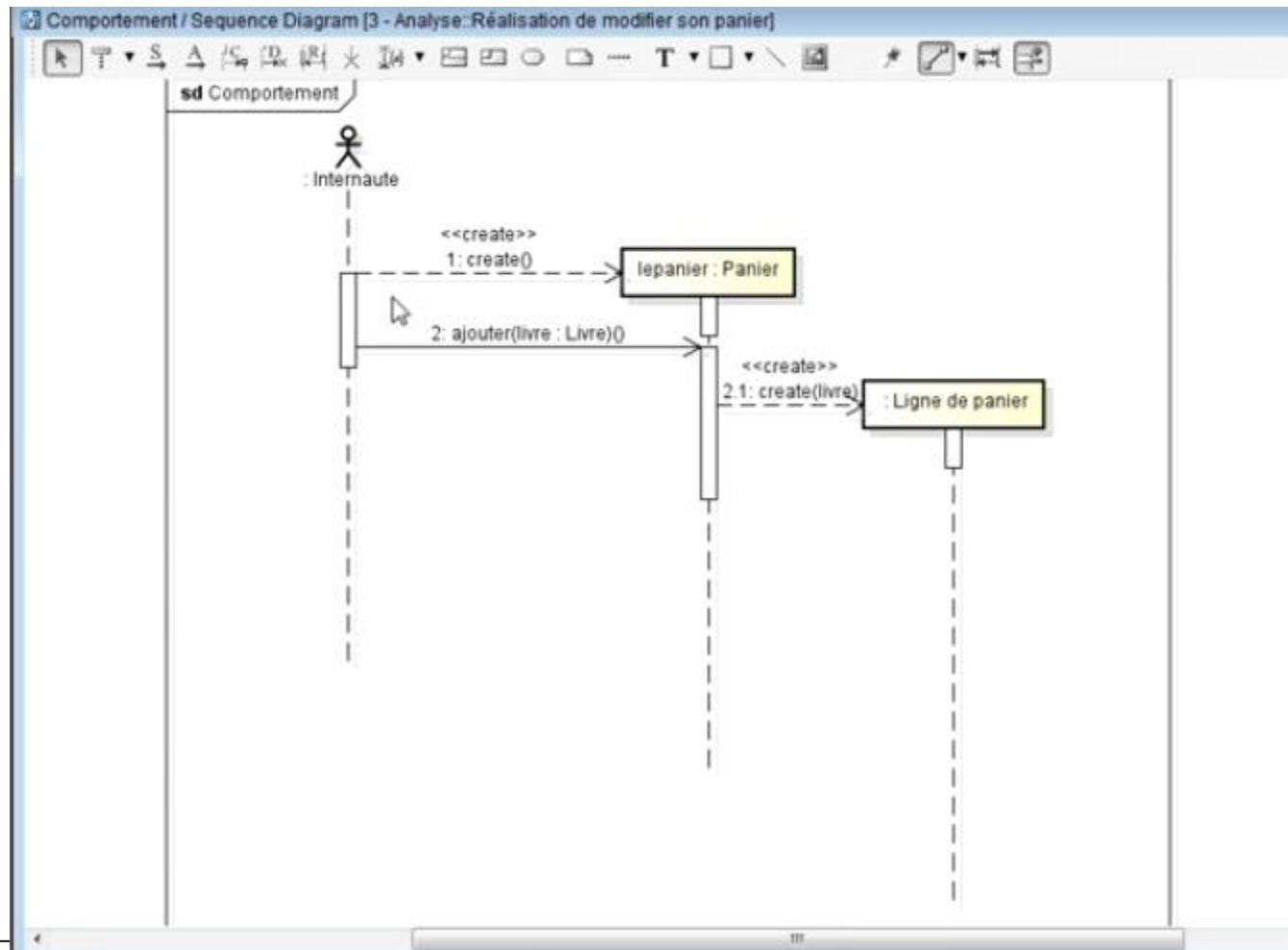




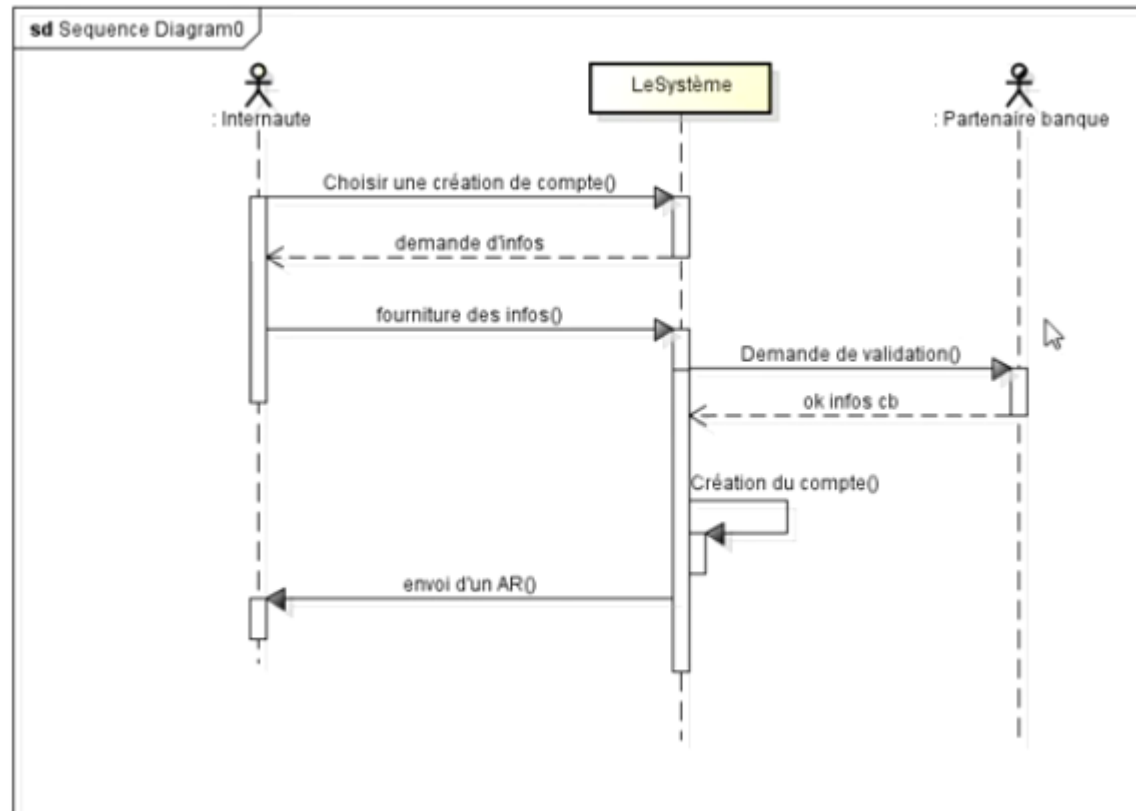
Les diagrammes de séquences peuvent servir à illustrer

- Les cas d'utilisation
 - instance du cas d'utilisation,
 - présente un scénario d'utilisation particulier du système,
 - met en œuvre les acteurs et le système.
- En analyse/Conception
 - les interactions temporelles des objets,
 - la description ou le résultat de cas test (unitaire, ..., recette)

2.5.4 - Exemple



2.5.4 - Exemple

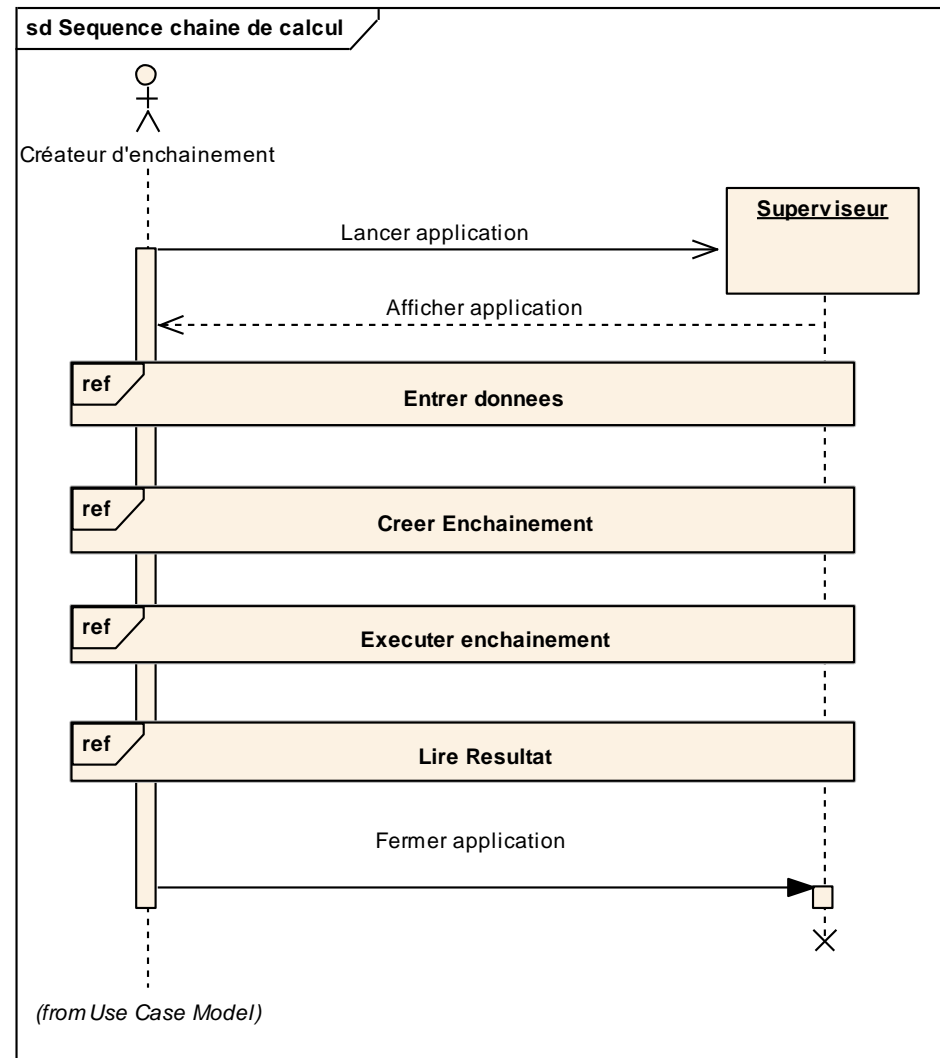


2.5.4 - Exemple

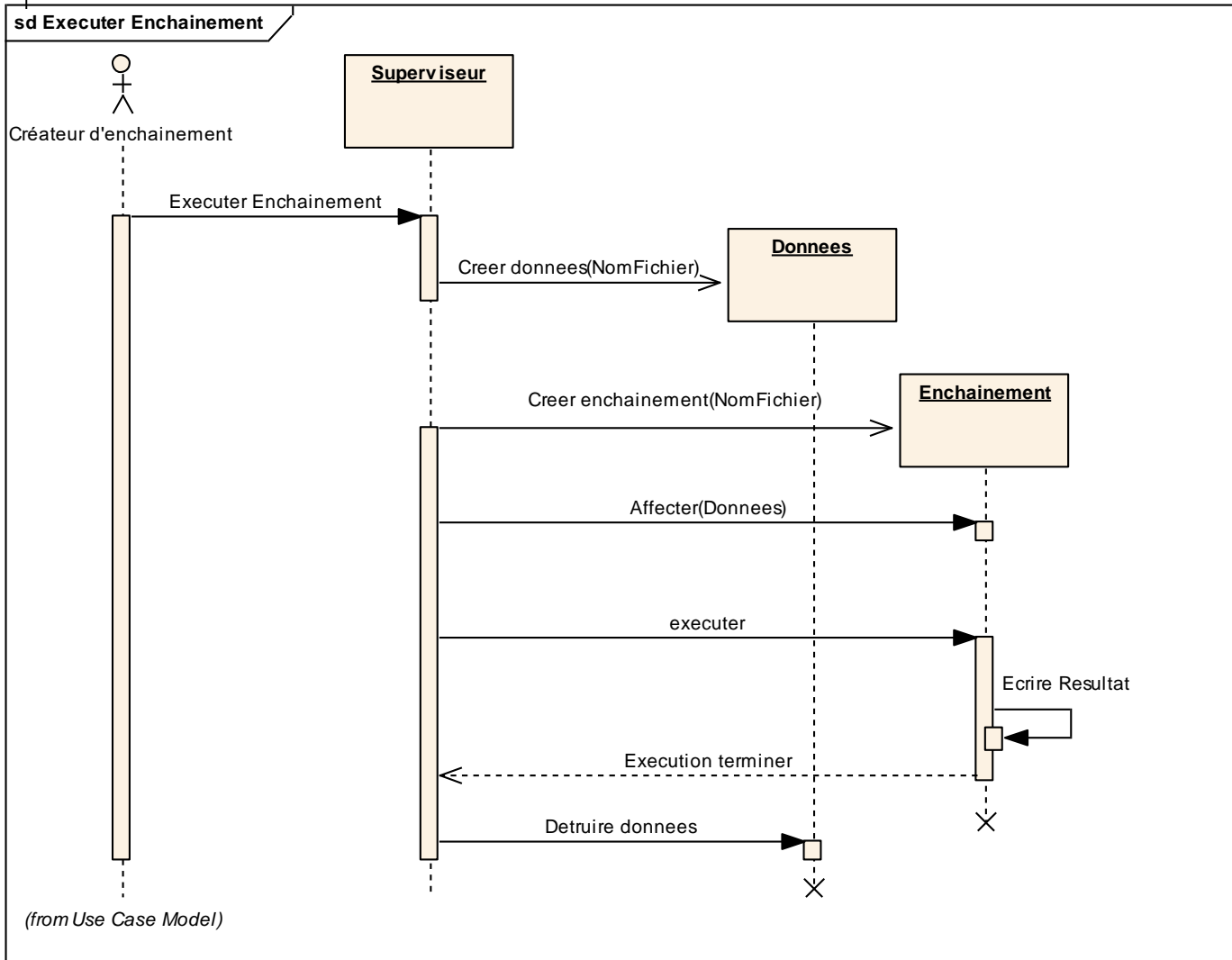


- Exemple : Séquence chaîne de calcul
- Exemple : Exécuter enchaînement
- Exemple : Créer enchaînement

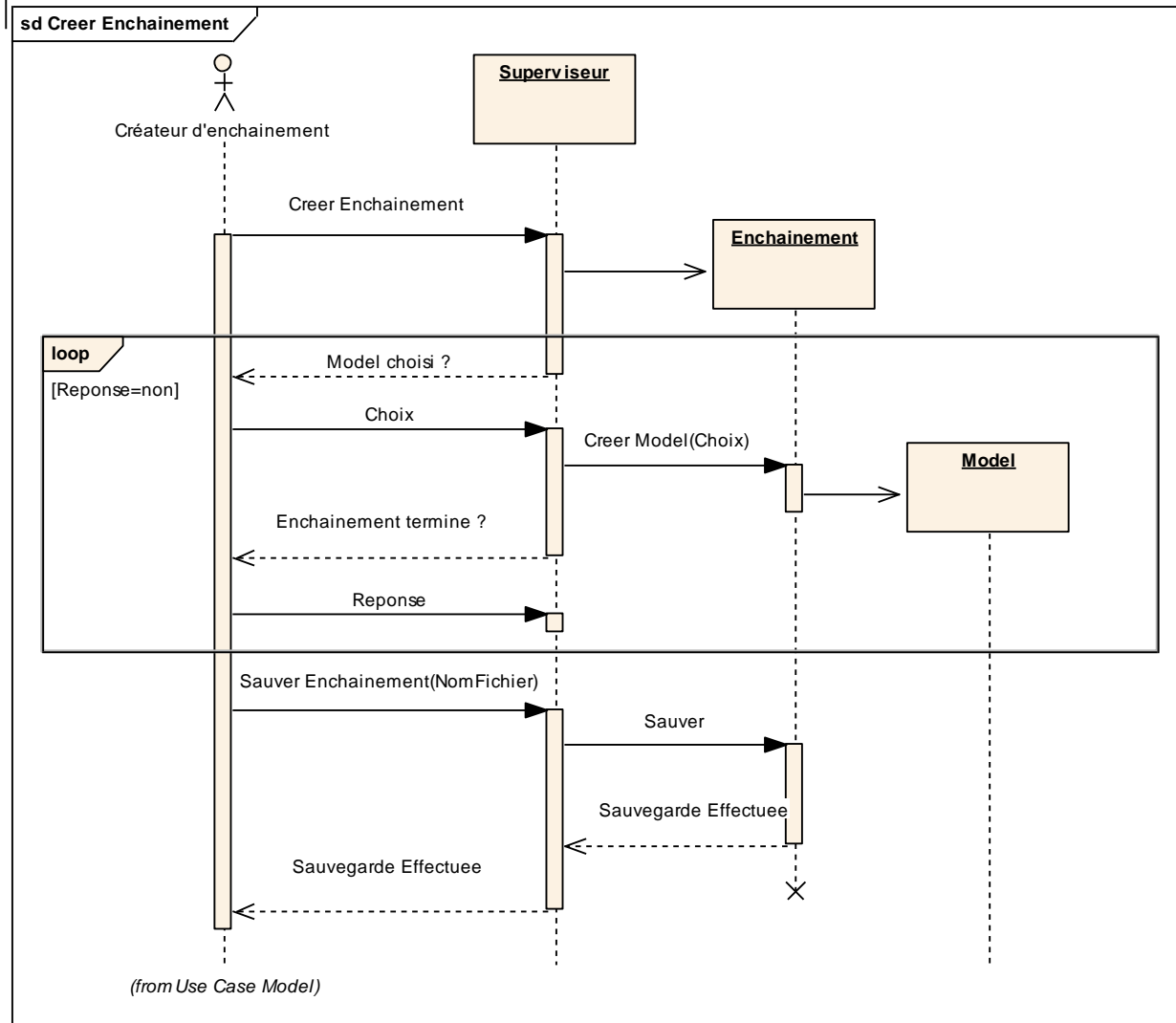
2.5.4.1 – Exemple : Séquence chaîne de calcul



2.5.4.2 – Exemple : Exécuter enchaînement



2.5.4.3 – Exemple : Créer enchaînement



2.6 - le diagramme d'états-transitions



- Fonctionnalité
- Mise en œuvre



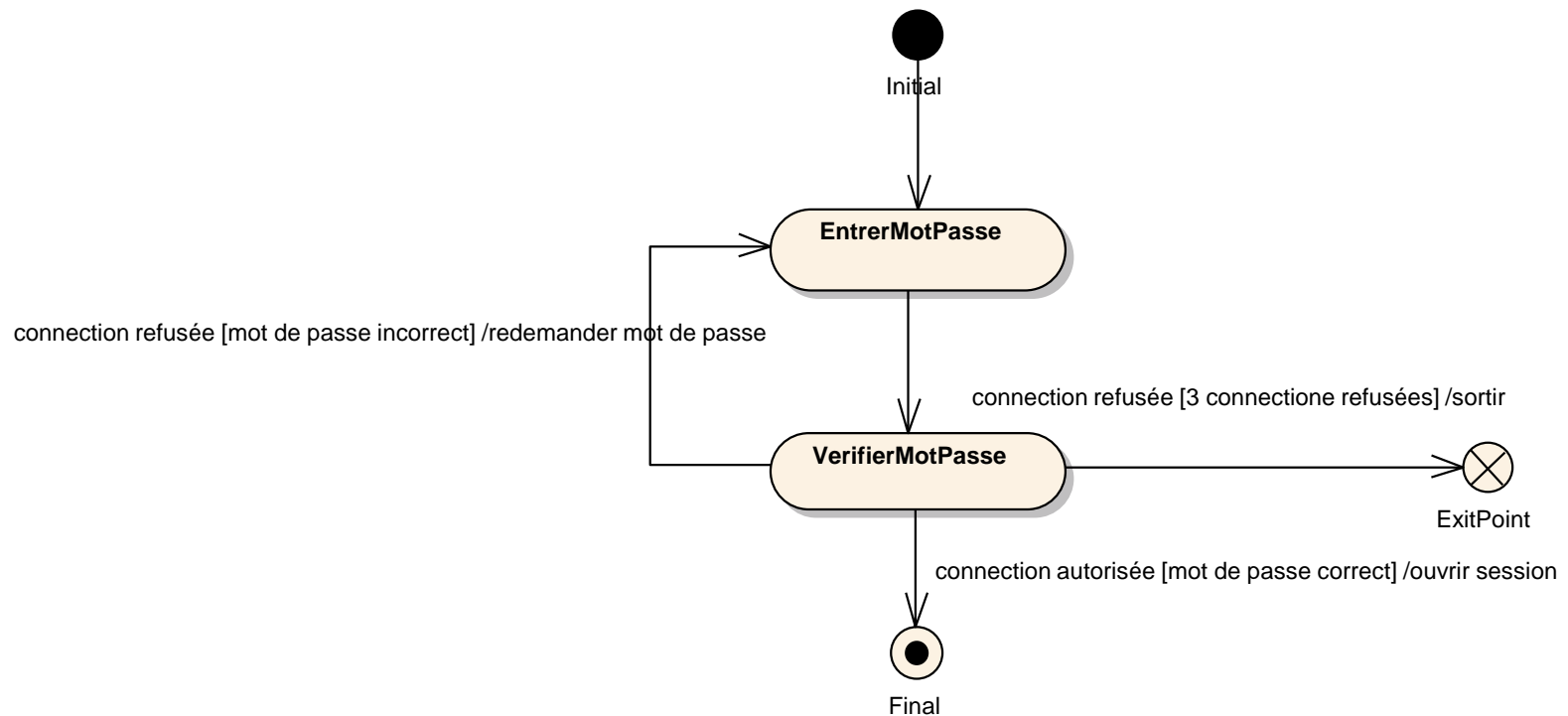
Les diagrammes d'états-transitions représente sous forme de machine à états finis le comportement du système ou de ses composants et permet de :

- 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.
- Représenter le cycle de vie des objets dans le processus
- Mettre en forme la dynamique du système.
- Représenter des automates d'états finis, sous forme de graphes d'états, reliés par des arcs orientés qui décrivent les transitions.

2.6.2.2a - Les transitions/Les évènements



sm Data Model

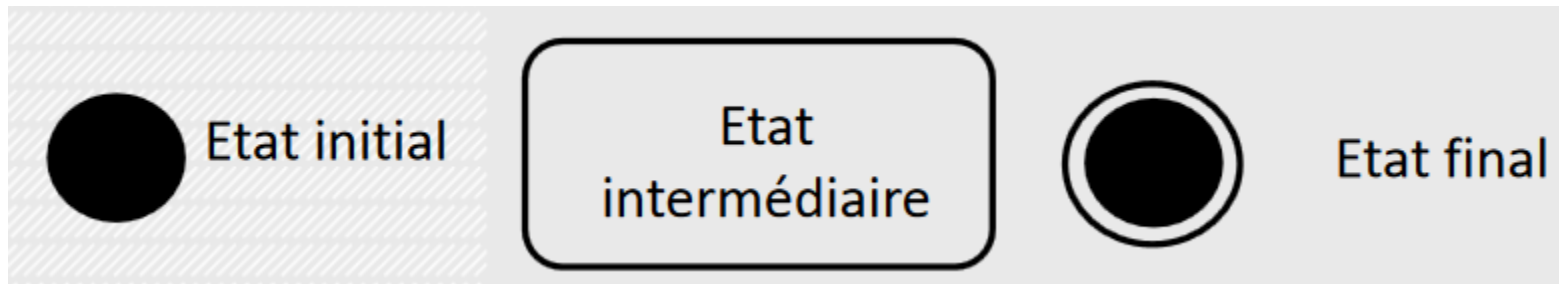


2.6.2.2b - Etat



Cette exécution est enrichie lorsque les états définissent une action d'entrée et une action de sortie : l'action de sortie de l'état de départ est exécutée d'abord, puis l'action de la transition, puis l'action de l'état d'arrivée.

- Un état = étape dans le cycle de vie d'un objet
- Chaque objet possède à un instant donné un état particulier
- Chaque état est identifié par un nom.
- Un état est stable et durable
- Chaque diagramme d'états-transitions comprend un état
- Il est possible de n'avoir aucun état final



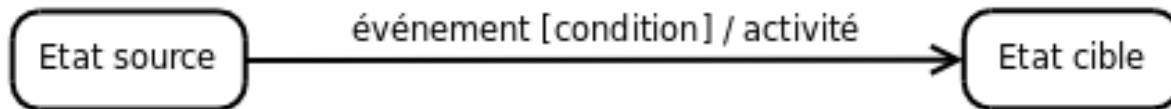
2.6.2.2c - Transition



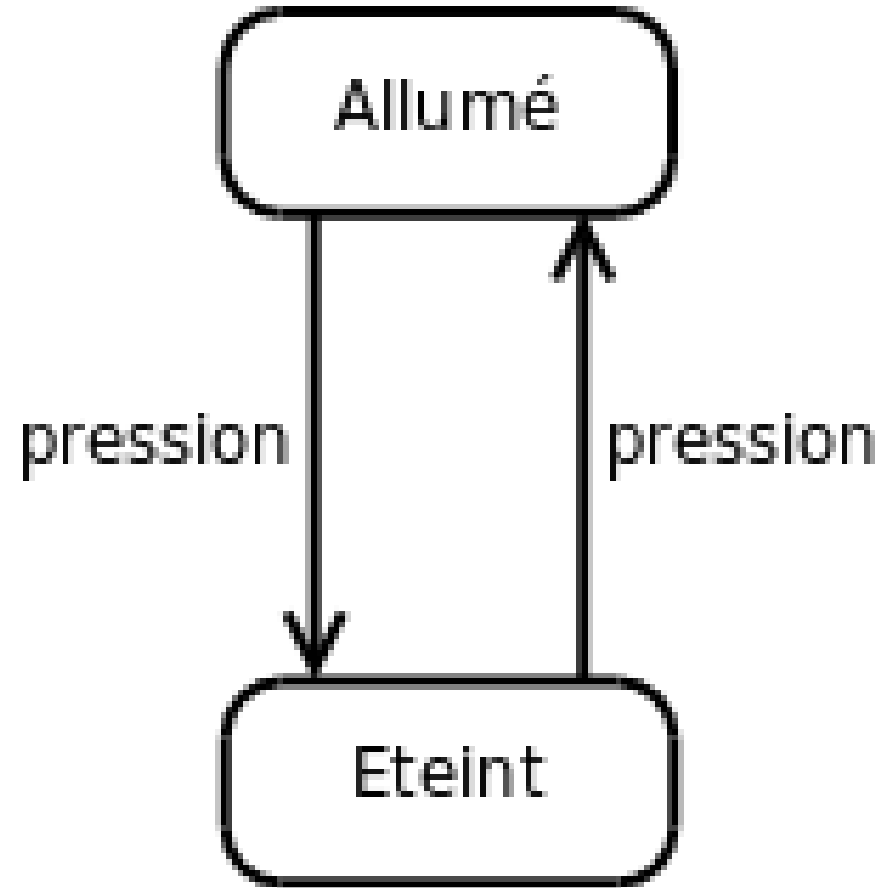
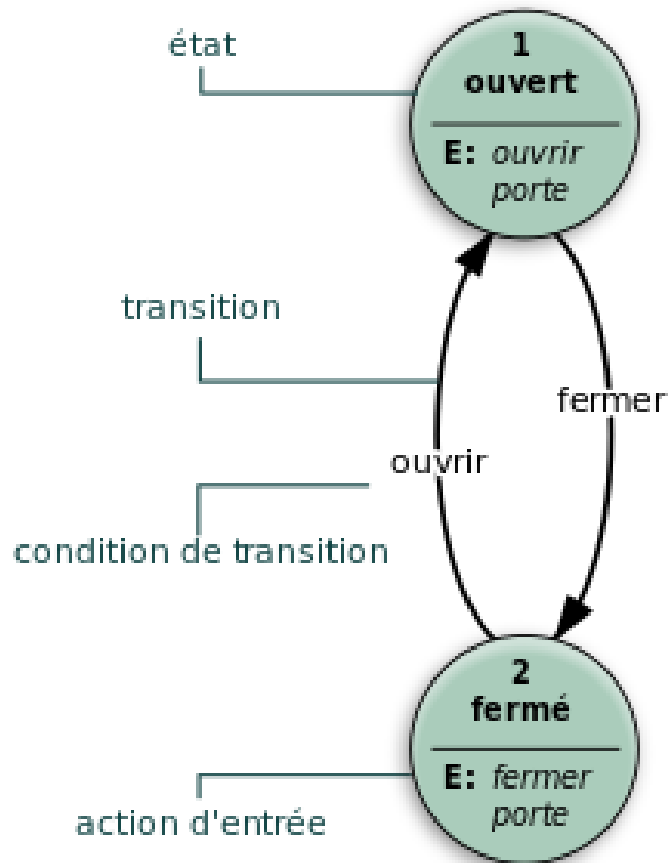
En plus des états de départ (au moins un) et d'arrivée (nombre quelconque), une transition peut comporter les éléments facultatifs suivants :

- Un évènement
- Une condition de garde
- Une liste d'actions

Quand l'évènement se produit alors que les états de départ sont actifs et que la condition de garde est vraie alors les actions seront déclenchées.



2.6.2.2 - Exemple



https://fr.wikipedia.org/wiki/Diagramme_%C3%A9tats-transitions

2.6.3 – Transitions Internes



Les transitions internes possèdent des noms d'événement prédéfinis correspondant à des **déclencheurs** particuliers : **entry**, **exit**, **do** et **include**.

Ces mots-clefs réservés viennent prendre la place du nom de l'événement dans la syntaxe d'une transition interne.

entry : spécifie une activité qui s'accomplit quand on entre dans l'état.

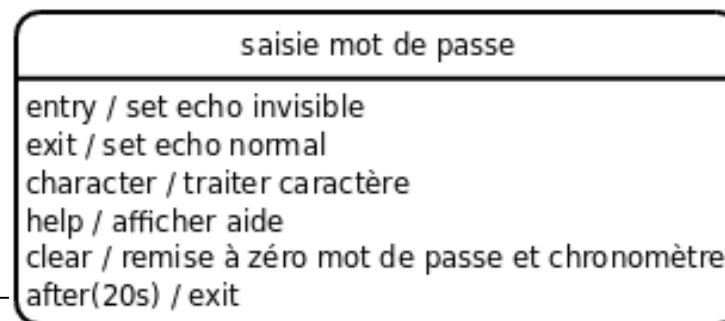
exit : spécifie une activité qui s'accomplit quand on sort de l'état.

do : une activité **do** commence dès que l'activité **entry** est terminée.

Lorsque cette activité est **terminée**, une transition **d'achèvement** peut être déclenchée, après l'exécution de l'activité **exit** bien entendu.

Si une transition se déclenche pendant que l'activité **do** est en cours, cette dernière est interrompue et l'activité **exit** de l'état s'exécute.

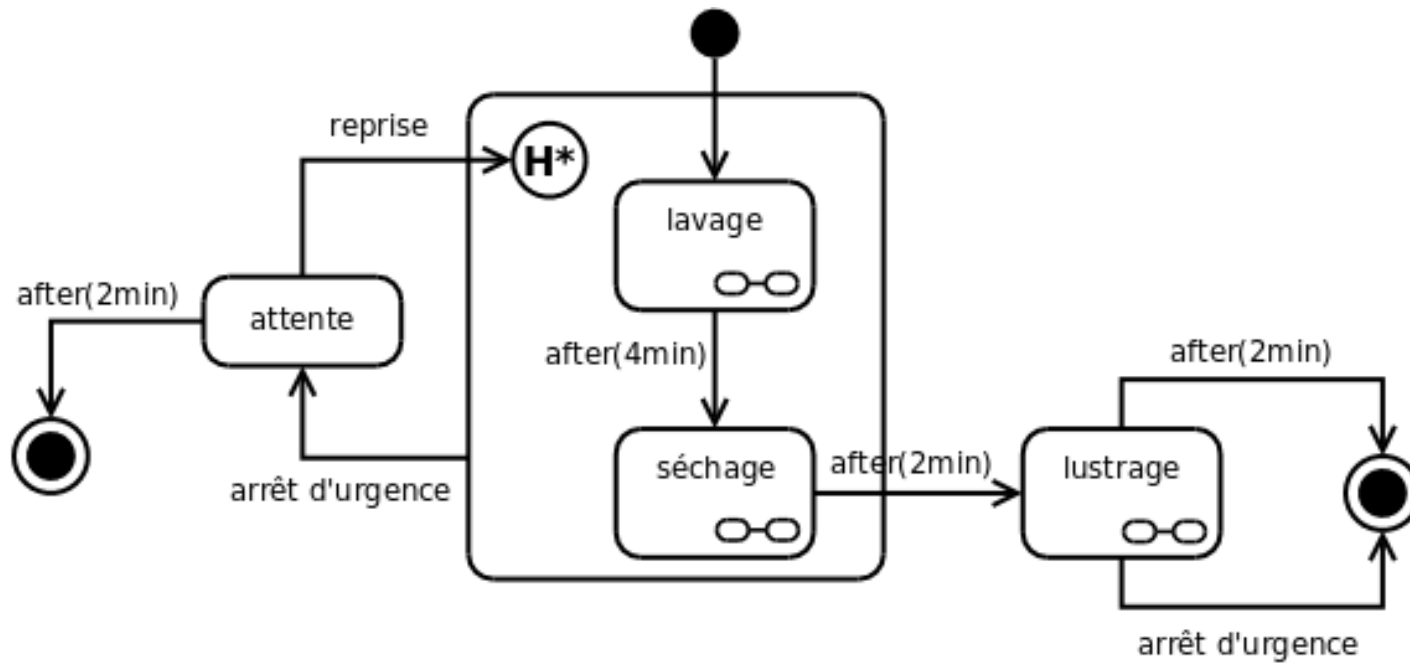
include : permet d'invoquer un sous-diagramme d'états-transitions.





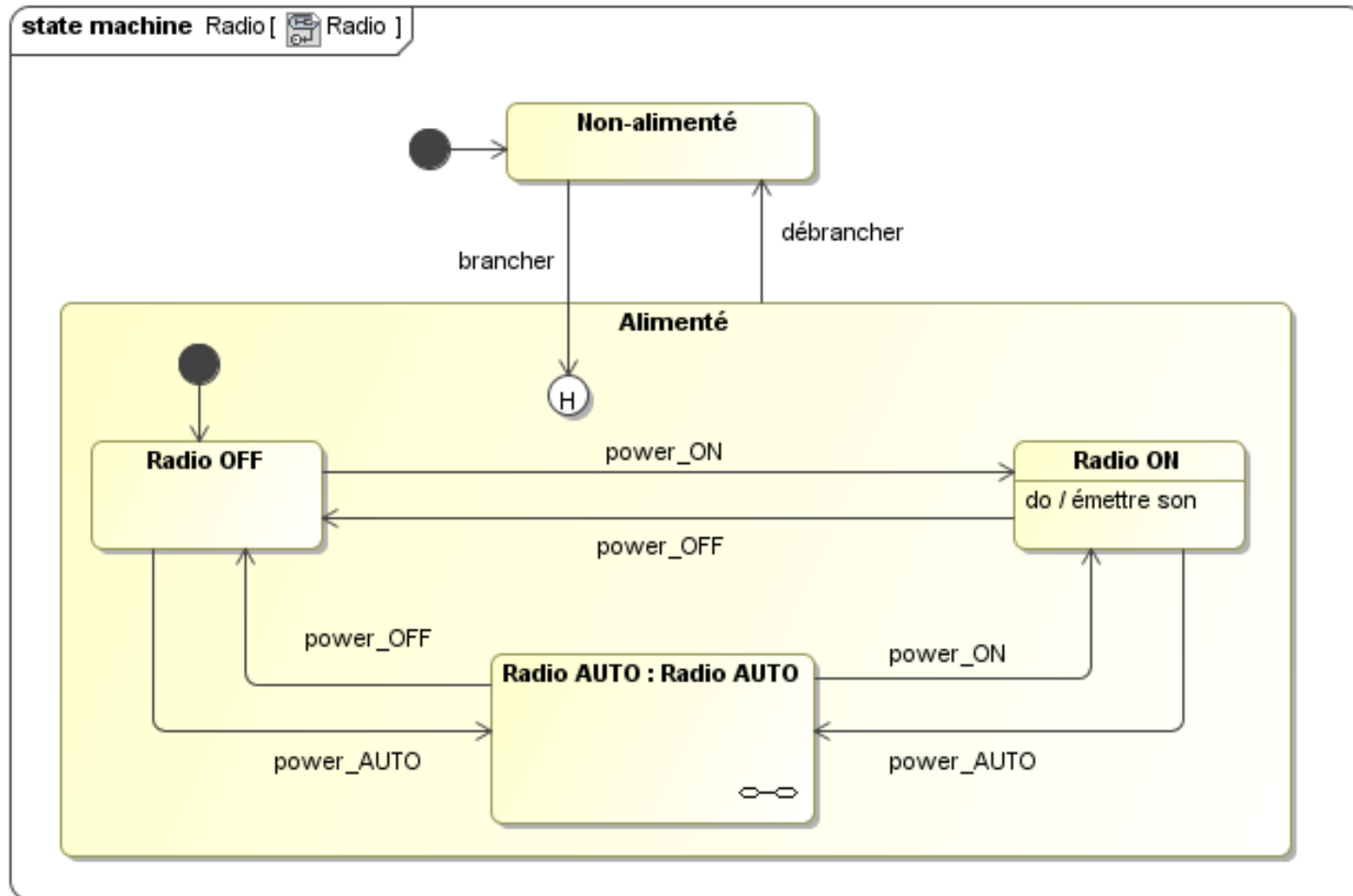
- Mettre en place le diagramme d'état pour décrire le comportement d'une classe intéressante, (mal adapté au comportement de plusieurs objets) .
- Ne pas créer un diagramme d'état par classe et par opération

2.6.5 - Exemple



<http://laurent-audibert.developpez.com/Cours-UML/?page=diagramme-etats-transitions>

2.6.5 - Exemple



http://www.siloged.fr/cours/STI2D_sysml/index.html?Diagrammedetat.html

2.7 - Le diagramme d'activité



- Fonctionnalité
- Mise en œuvre



Les diagrammes d'activités permettent de :

- Représenter graphiquement le comportement d'une méthode ou le déroulement d'un cas d'utilisation, à l'aide de diagrammes d'activités (une variante des diagrammes d'états-transitions)
- Visualiser l'exécution d'un mécanisme, un déroulement d'étapes séquentielles
- Décrire tous les mécanismes dynamiques (en théorie); seuls les mécanismes complexes ou intéressants méritent d'être représentés

2.7.2 – Différence diagramme d'activités / diagramme d'états



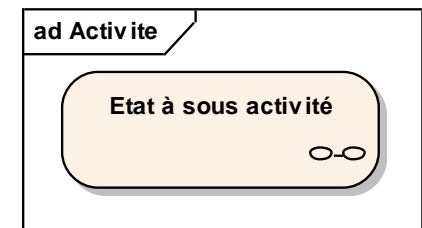
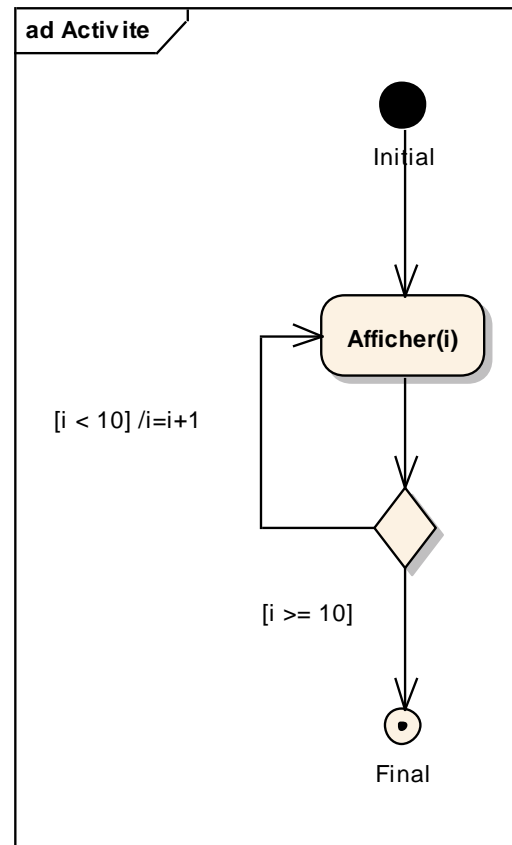
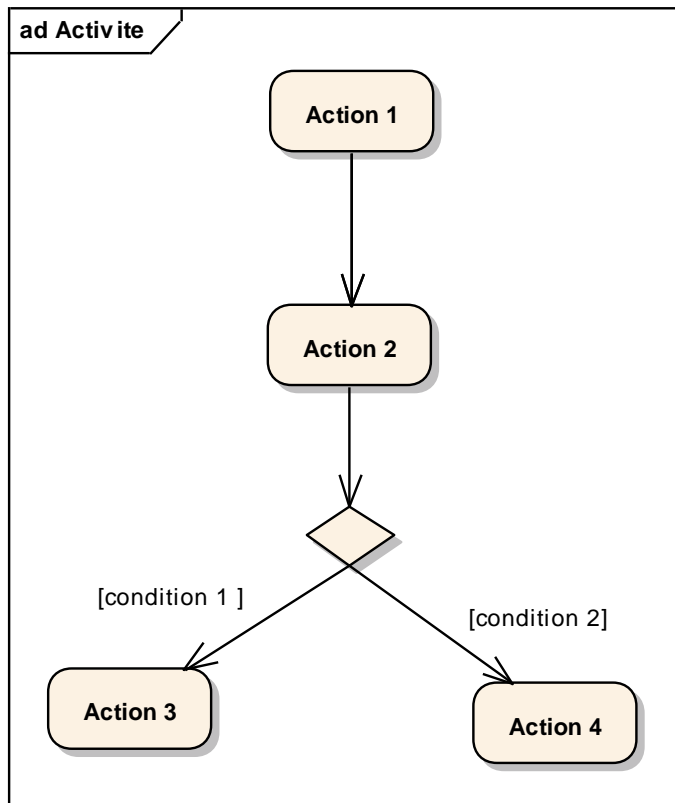
Les **diagrammes d'activités** et les diagrammes **d'états** sont **associés**.

Tandis qu'un **diagramme d'états** concentre l'attention sur le **processus** en cours d'un **objet** (ou sur un processus pris comme objet), un diagramme **d'activités** se focalise sur **le flot des activités** impliquées dans un processus unique.

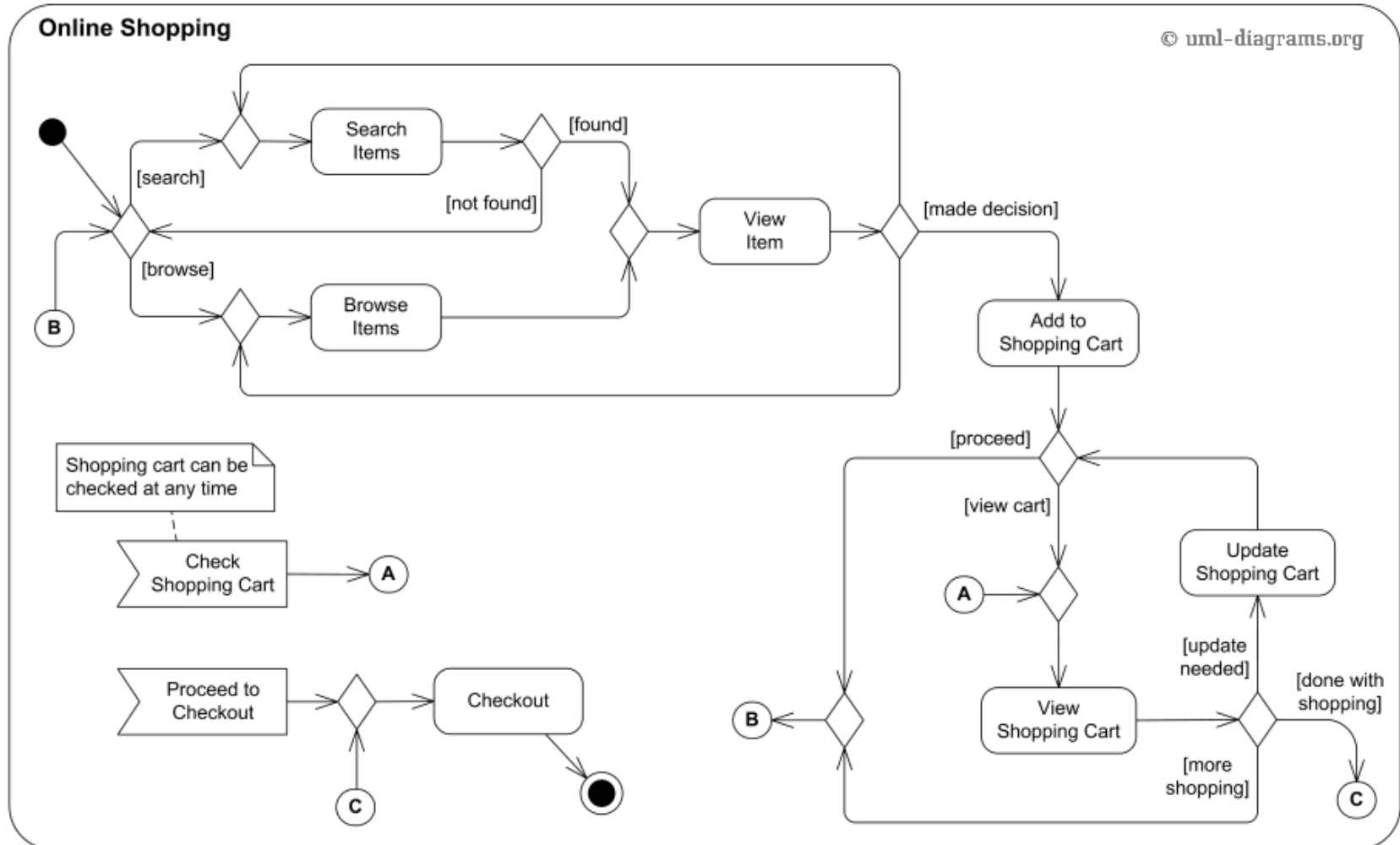
Le diagramme d'activités :

- semblable à un organigramme.
- montre la façon dont ces activités à processus unique dépendent les unes des autres.
- peuvent être divisés en travées d'objets qui déterminent l'objet responsable d'une activité.

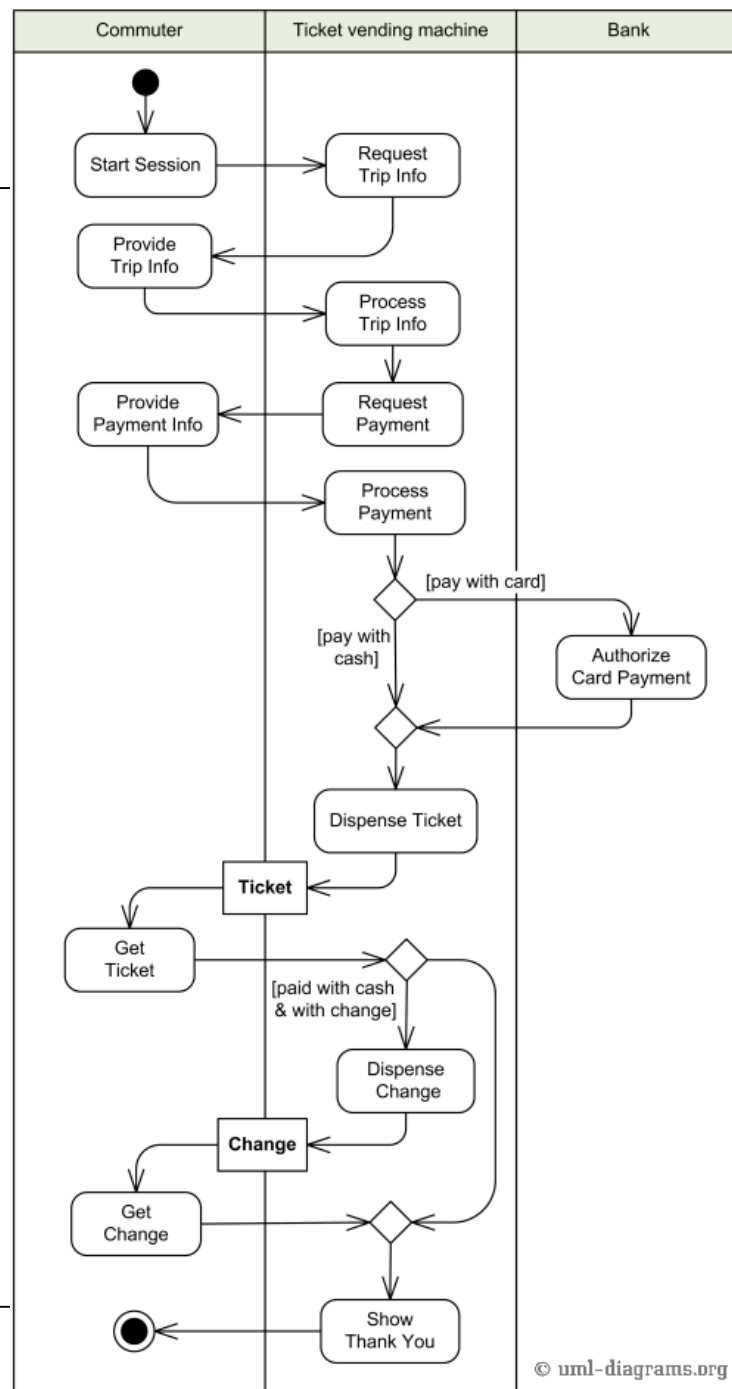
2.7.3.1 – Transitions/États-action



2.7.4 - Exemple : Achat en ligne



2.7.4 - Exemple : Distributeur automatique de billets





Le diagramme d'activité peut être utilisé pour :

- Représenter du parallélisme
- Représenter un organigramme

Ce diagramme est peu utilisé

2.8 - Le diagramme de composants



- Fonctionnalité
- Mise en œuvre



Les diagrammes de composants permettent de :

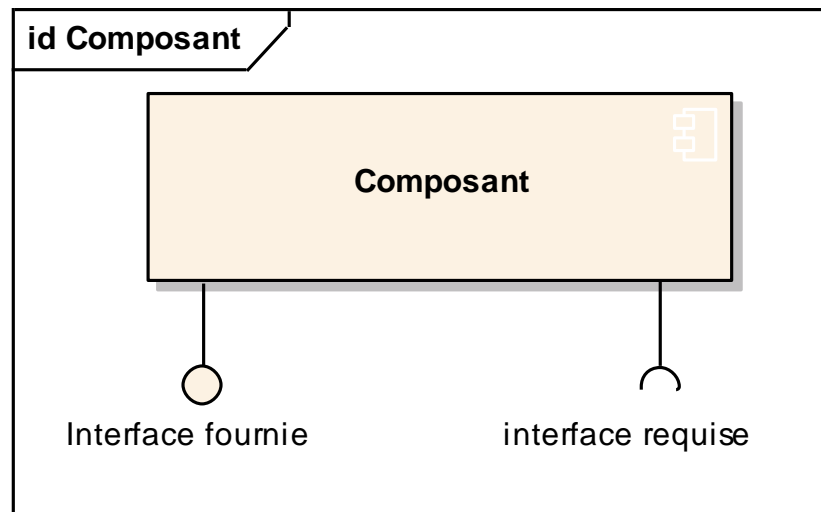
- Décrire les éléments physiques et leurs relations dans l'environnement de réalisation
- Montrer les choix de réalisation
- Décrire l'architecture physique et statique d'une application en terme de modules : fichiers sources, librairies, exécutables, etc.
- Identifier les contraintes de compilation et mettre en évidence la réutilisation de composants
- Gérer la complexité, par encapsulation des détails d'implémentation.

2.8.2 - Notation

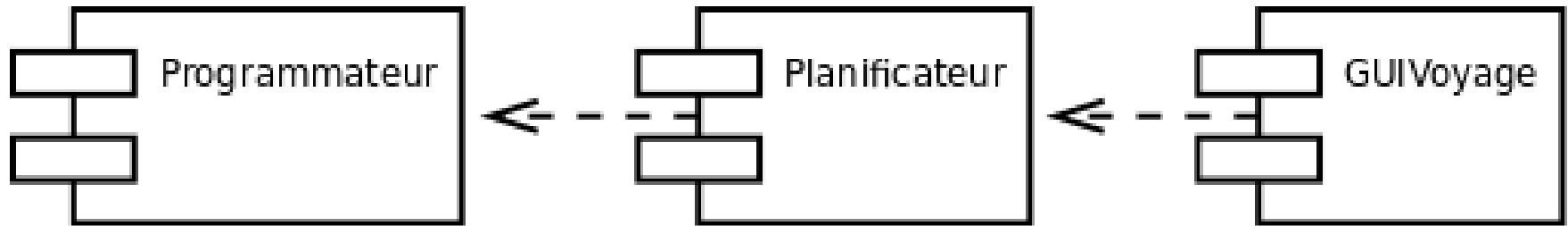


- Composants
- Dépendances entre composants

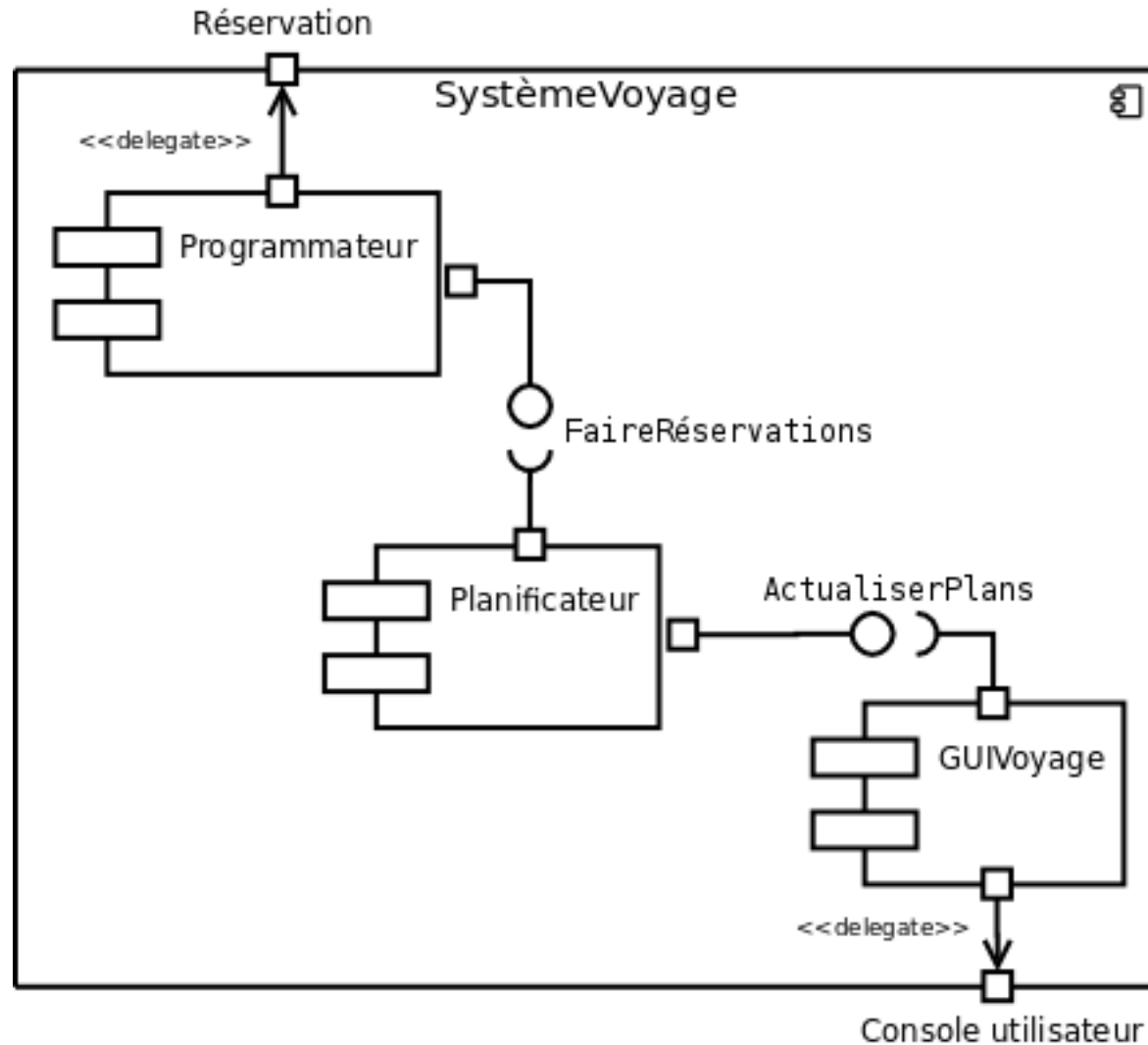
2.8.2 - Composants



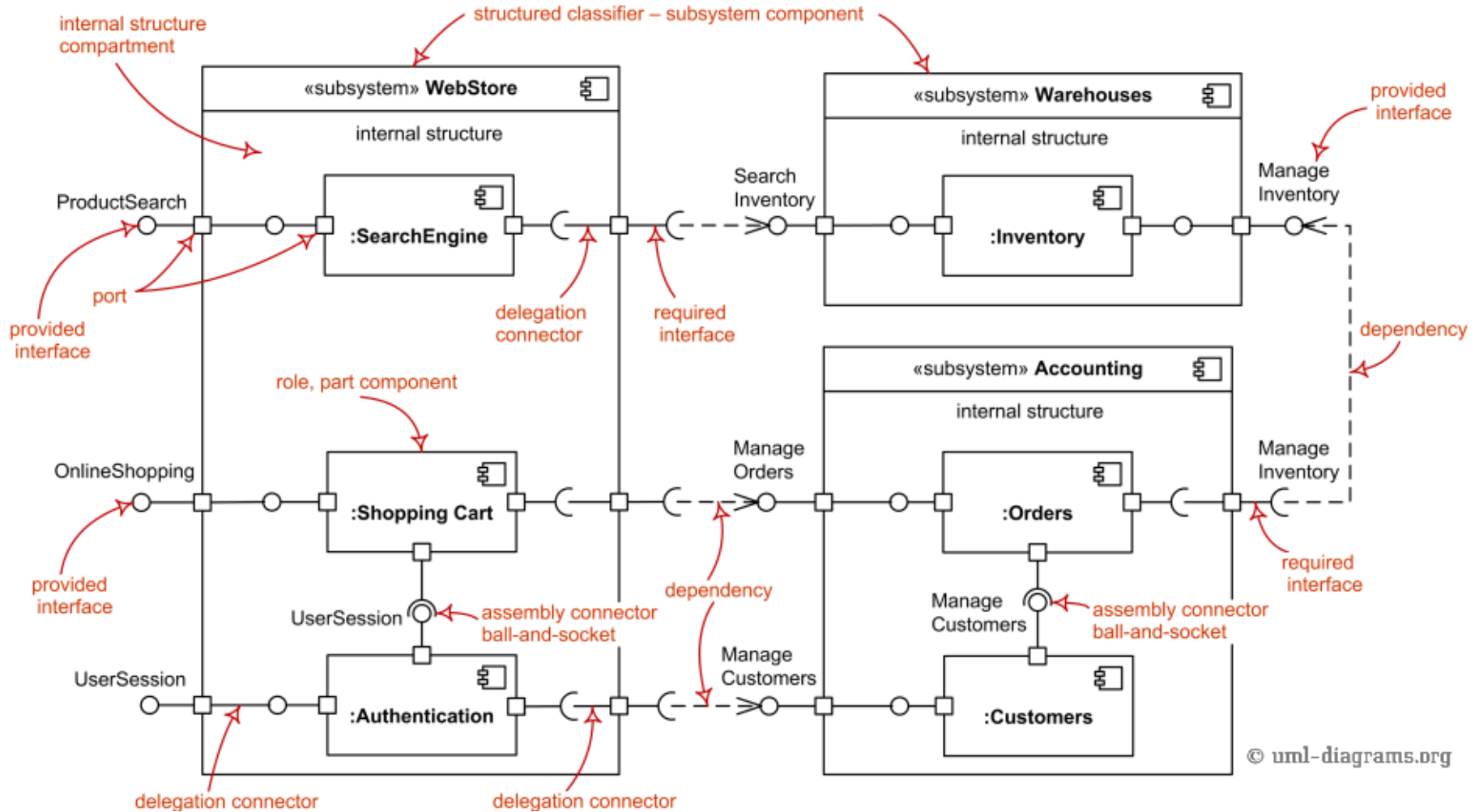
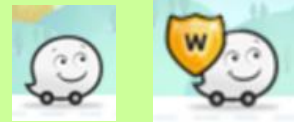
2.8.3 - Dépendance entre composant



2.8.4 – Exemple



2.8.4 – Exemple





Le diagramme de composants s'utilise si l'on veut représenter les relations entre composants

<http://laurent-audibert.developpez.com/Cours-UML/?page=diagrammes-composants-deploiement#L8>

2.9 - Le diagramme de déploiement



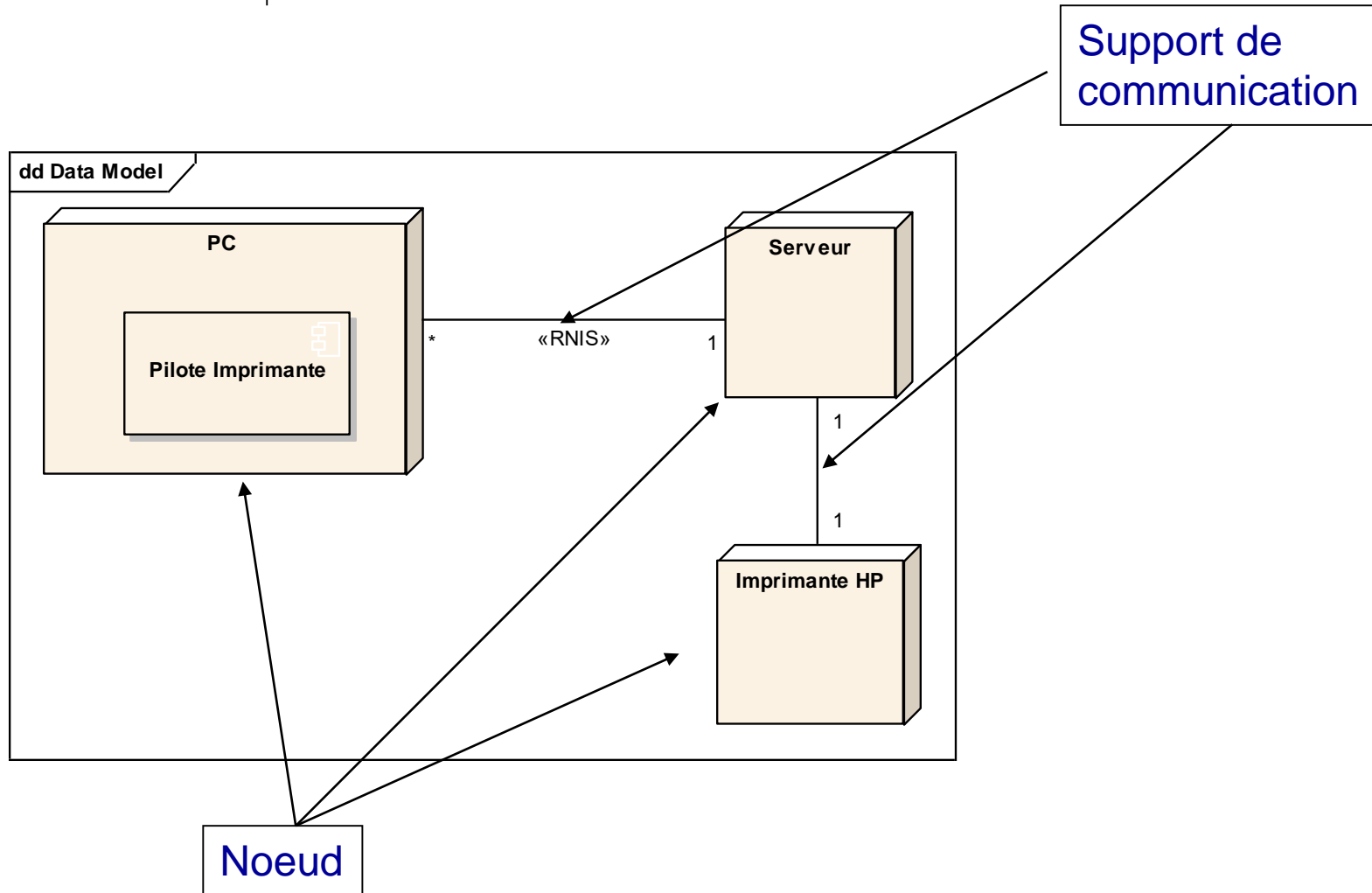
- Fonctionnalité
- Mise en œuvre



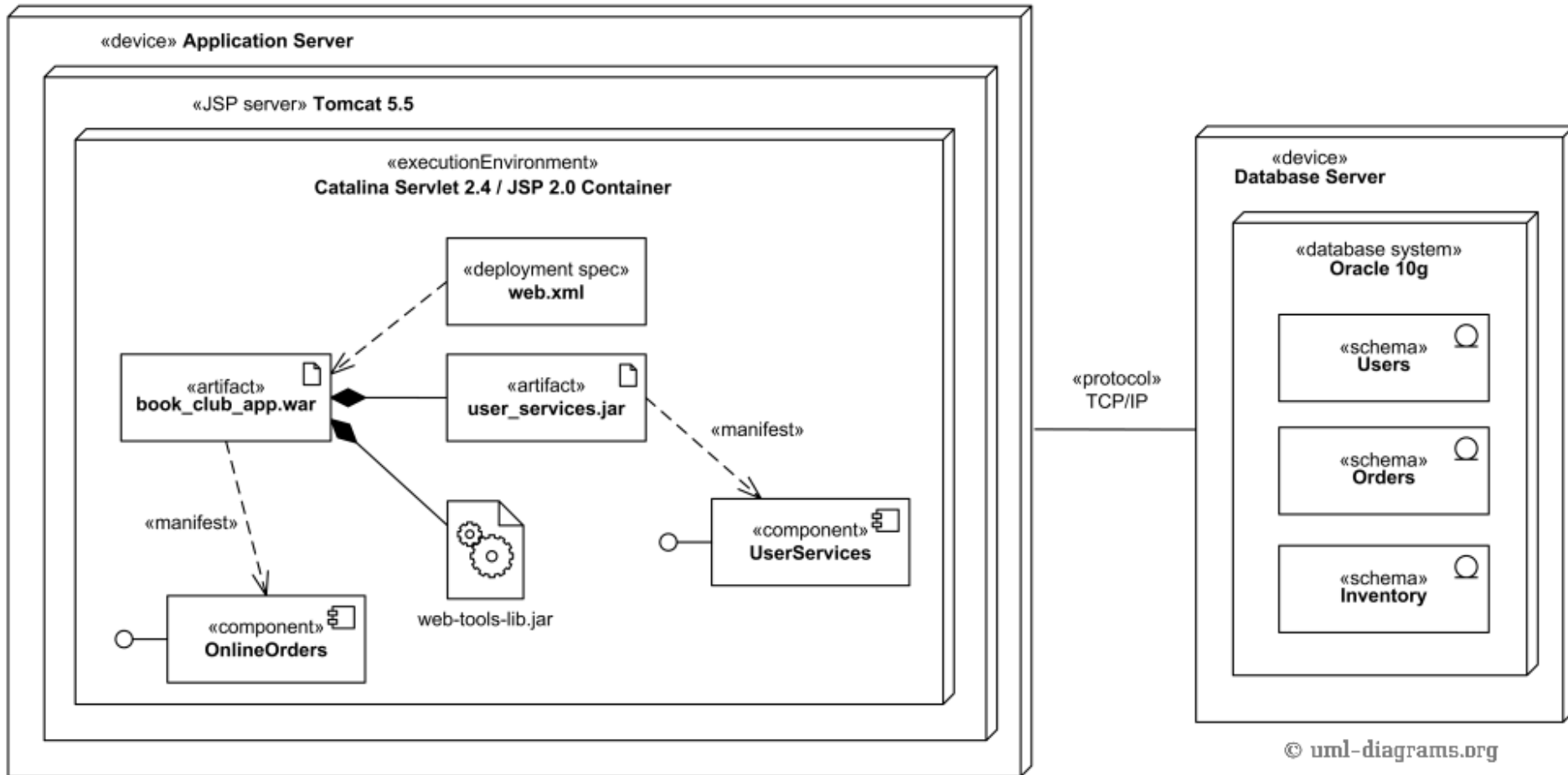
Le diagramme de déploiement présente l'architecture physique hôte du système, de manière très simplifiée et schématique.

Il permet de représenter la **répartition** des programmes exécutables sur les différents noeuds

2.9.2 - Notation



2.9.3 - Exemple Web Application





A utiliser pour toute opération de
déploiement non trivial

2.10 - Le diagramme de Communication/Collaboration



- Fonctionnalité
- Mise en œuvre



Les diagrammes de collaboration permettent de :

- Montrer des **interactions** entre objets (structure statique)
- Représenter le **contexte** d'une interaction (message)

2.10.2.1 - Les rôles



cd Component Model

Objet "instanceClasse" instance de la classe "Classe1", jouant le rôle "Role"

instanceClasse / Role :Classe 1

cd Component Model

David / Formateur :Personne

/ Employer :Personne

cd Component Model

David / Formateur :Personne

1

10

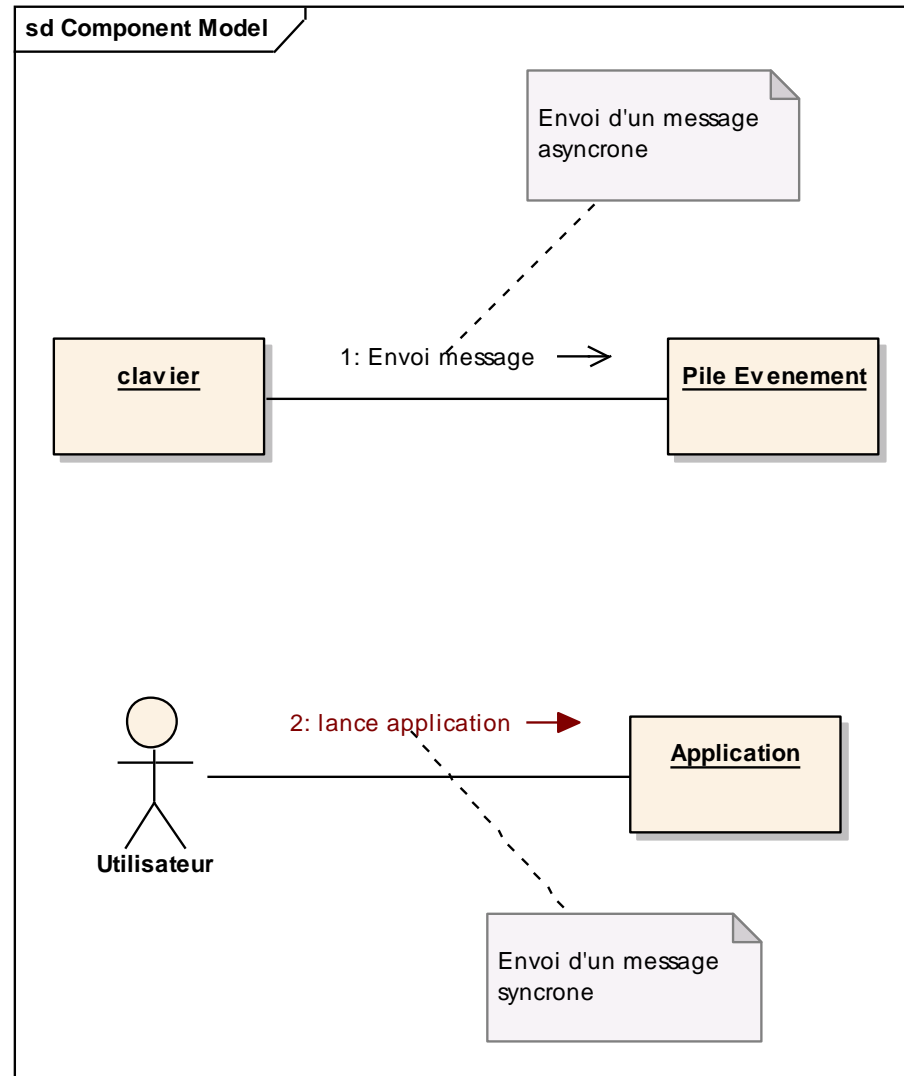
/ Employer :Personne

1

1

:SupportCours

2.10.2.3 - Envois de messages





Les diagrammes de Communication/Collaboration peuvent servir à illustrer :

- la vue des cas d'utilisation :
 - rôle identique au diagramme de séquence
 - réalisation un cas d'utilisation au moyen de collaborations entre objets qui y participent
- la vue logique :
 - transition vers l'objet => sert à découvrir les objets du domaine
 - illustration/démonstration d'un modèle achevé

2.11 - Le diagramme de vue d'ensemble des interactions



- Fonctionnalité
- Notation
- Mise en œuvre

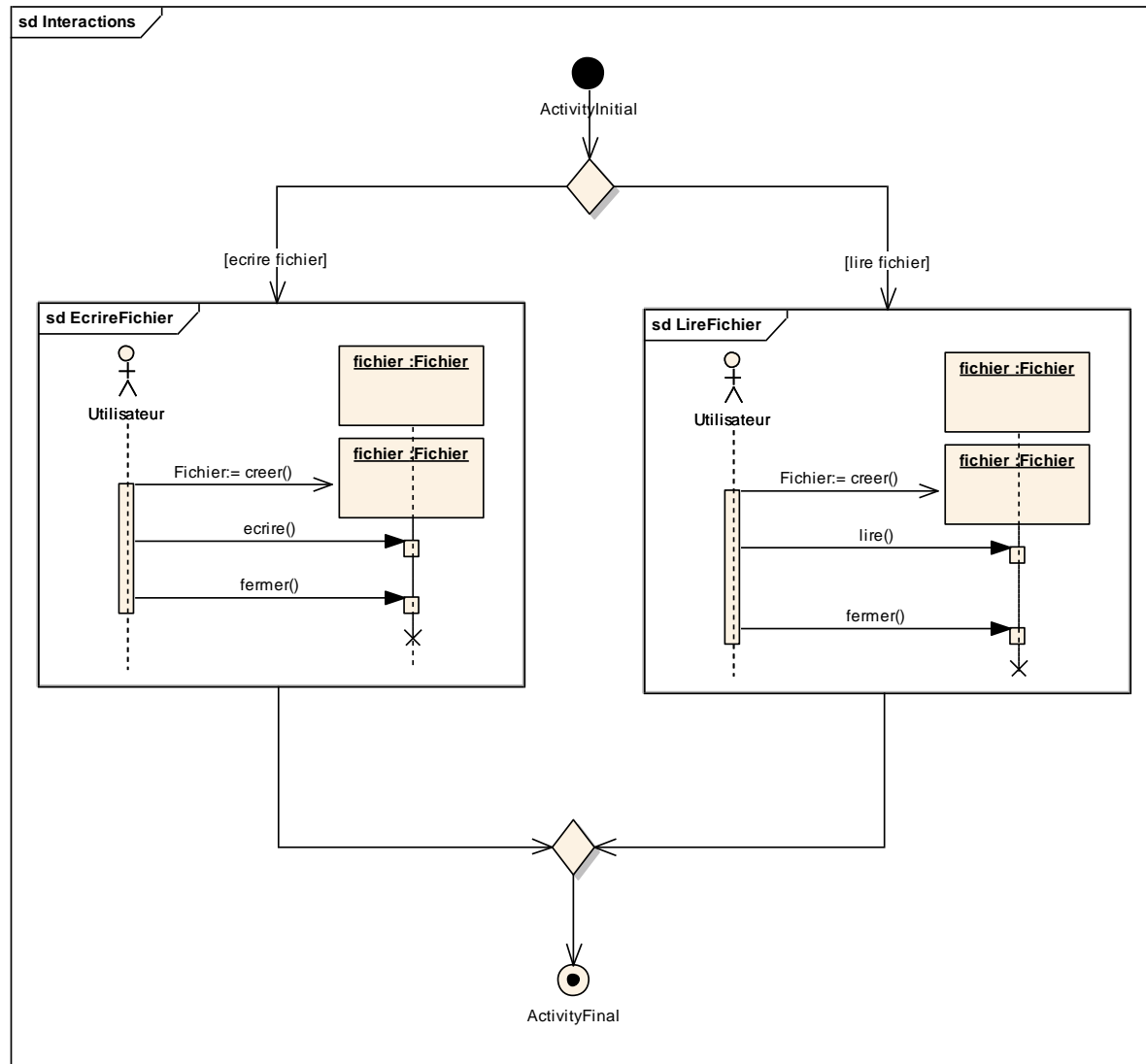


C'est le **diagramme global d'interaction** (interaction overview)

Il associe les notations du diagramme de séquence à celle du diagramme d'activité, ce qui permet de décrire une méthode complexe.

C'est une variante du diagramme d'activité.

2.11.2 - Notation





Le diagramme de **vue d'ensemble** des interactions est un nouveau diagramme UML 2.0 qui permet d'avoir une vue d'ensemble.
A utiliser avec parcimonie.

2.12 - Le diagramme de structure composite

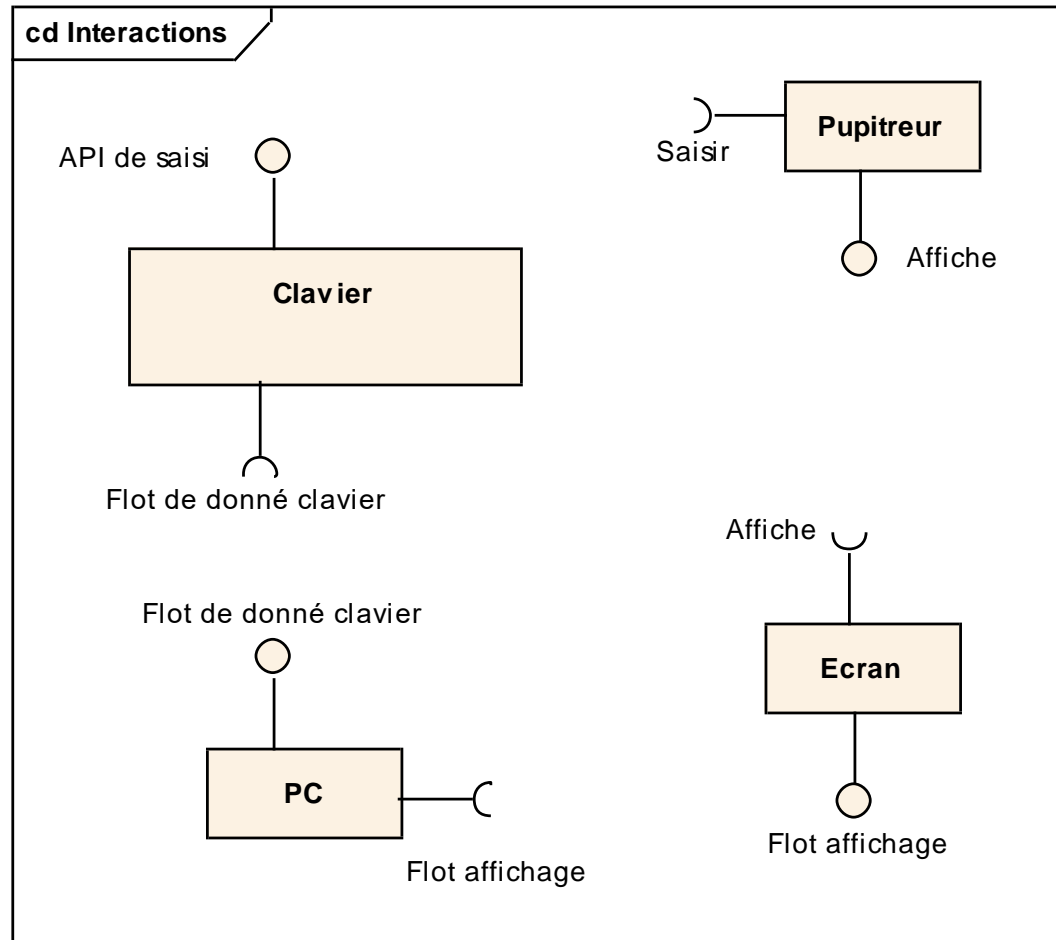


- Fonctionnalité
- Notation
- Mise en œuvre



Les diagrammes de structure composite permettent de **décrire la structure interne d'un objet complexe** lors de son exécution (au run-time - décrire l'exécution du programme), et donc ses **points d'interaction** avec le reste du système

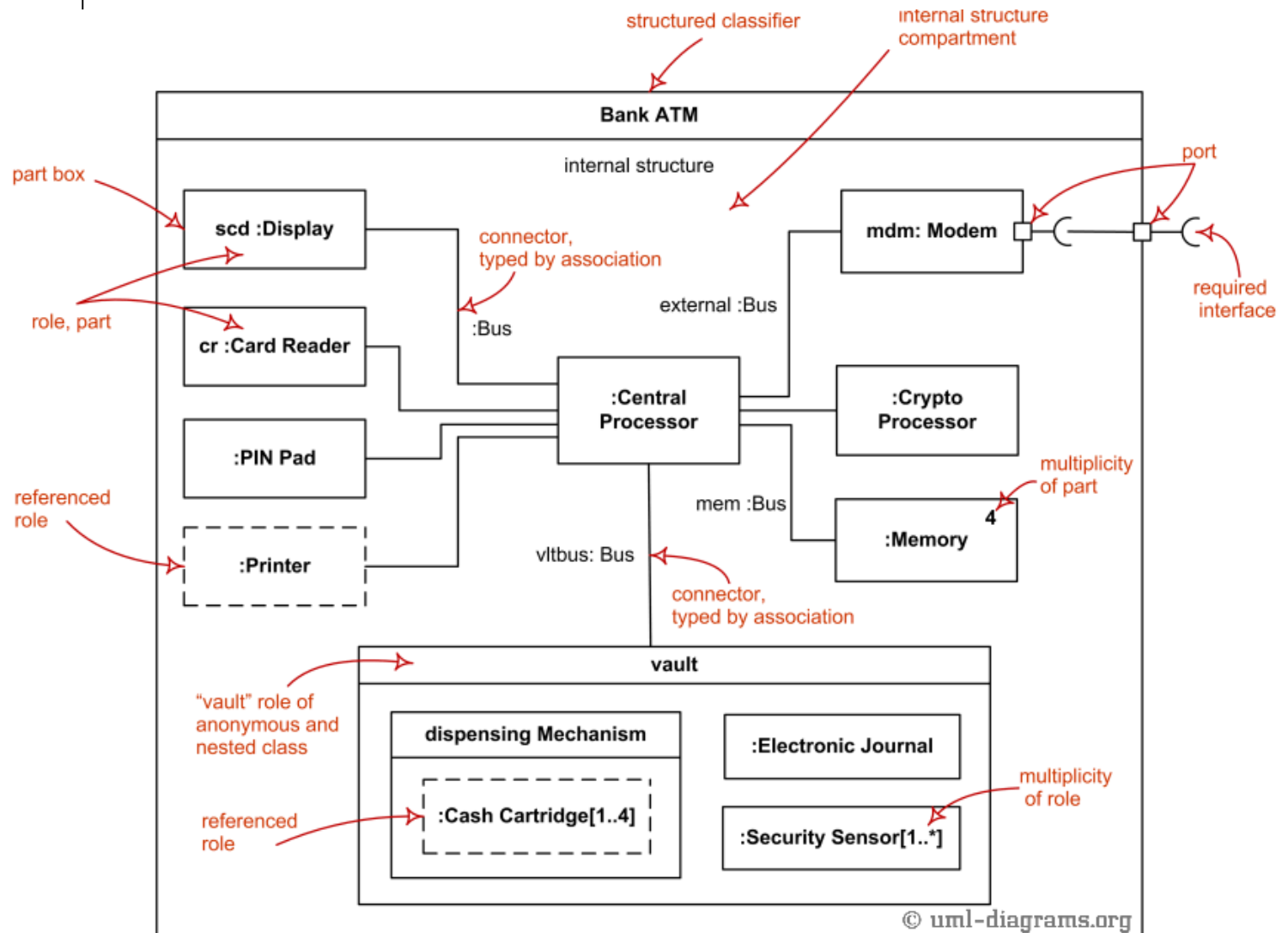
2.12.2 - Notation





Le diagramme de structure composite est un nouveau diagramme UML 2.0 adapté à la représentation des composants et à la façon de les décomposer.

2.12.3 - Exemple



2.13 - Le diagramme de timing

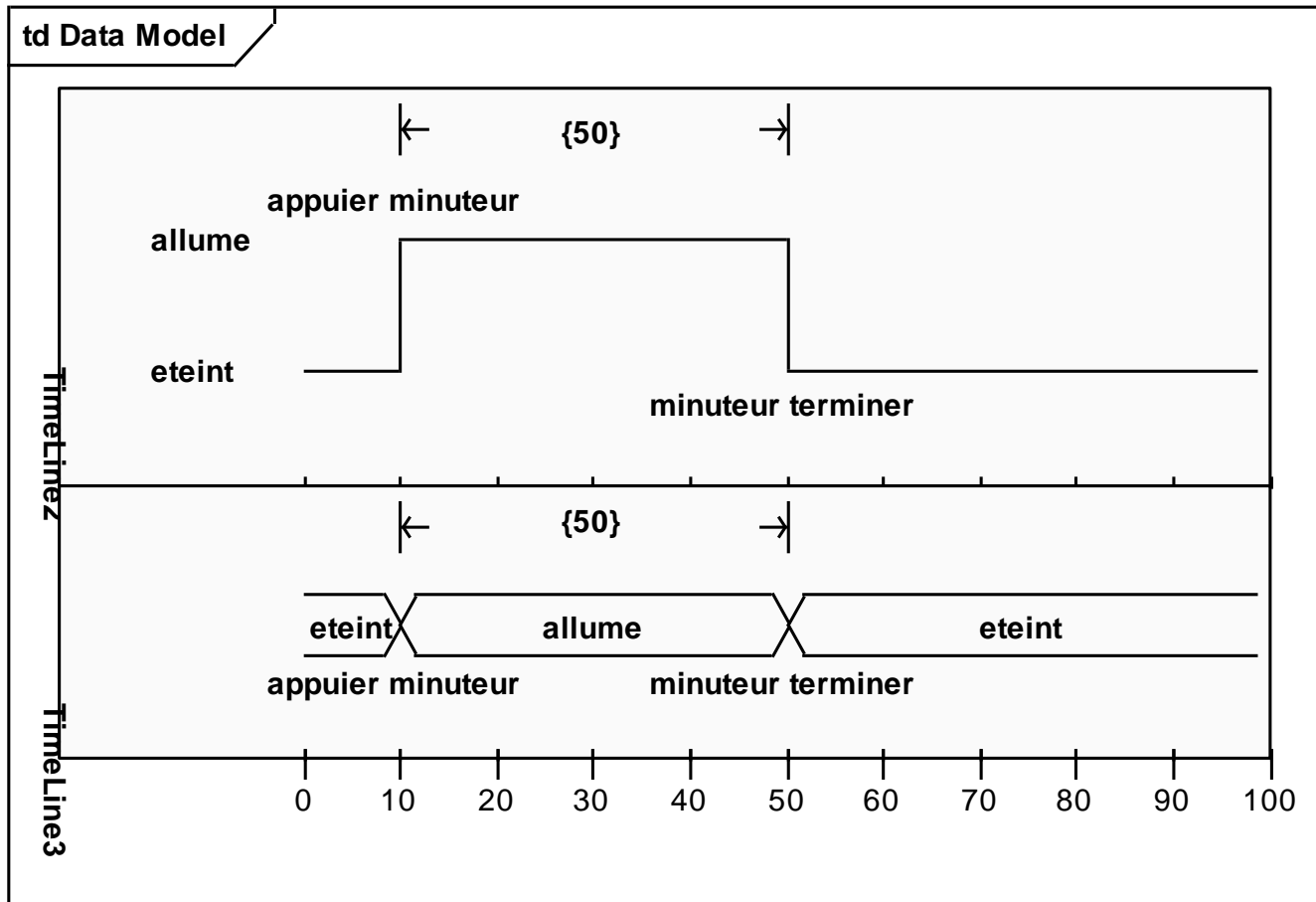


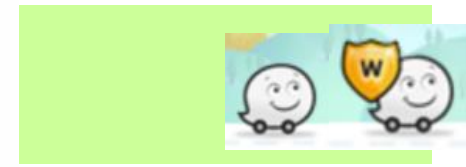
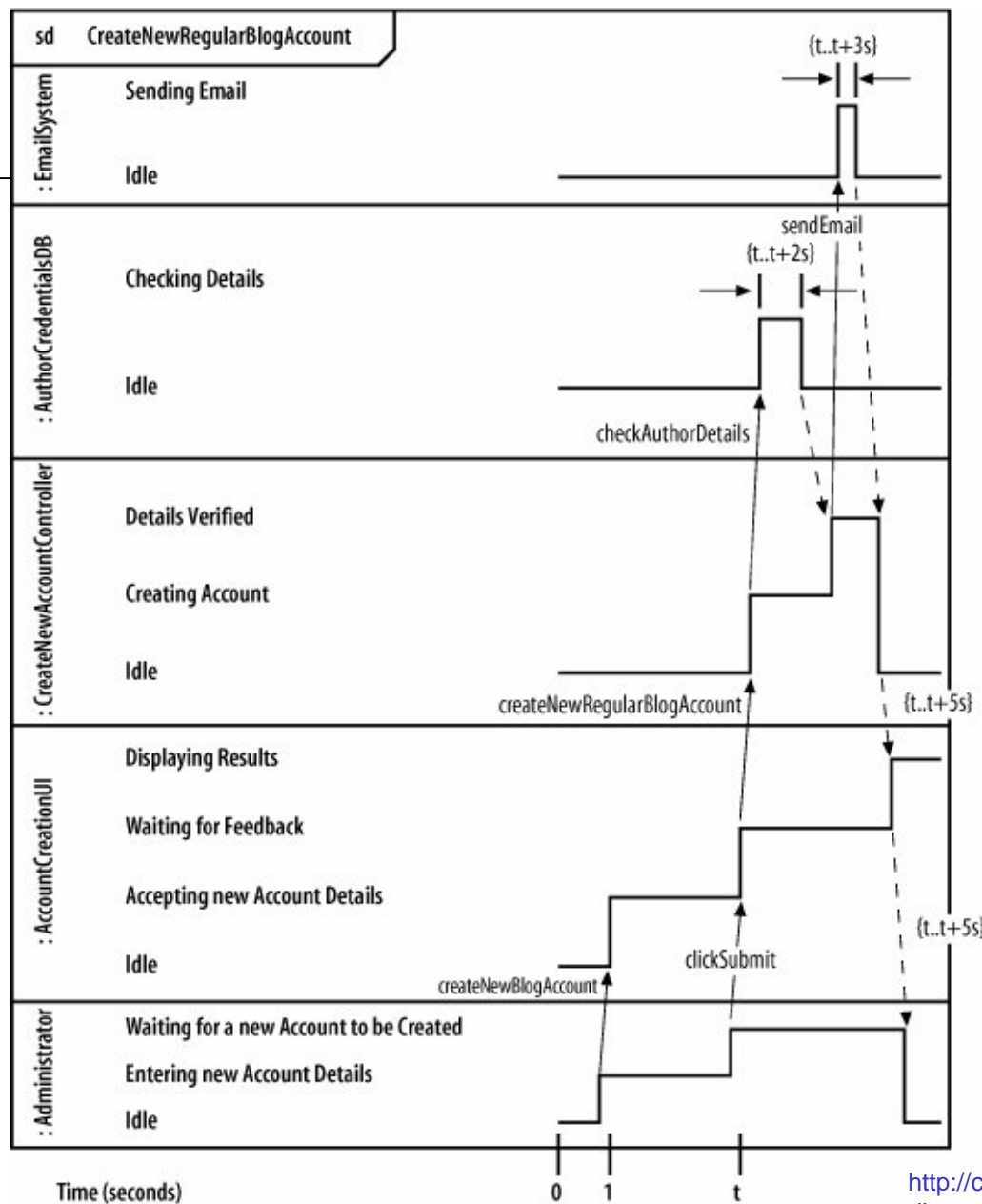
- Fonctionnalité
- Notation
- Mise en œuvre



Le diagramme de timing (timing diagram) :
permet de **modéliser les contraintes d'interaction** entre plusieurs objets,
comme le changement d'état en réponse
à un évènement extérieur.

2.13.2 - Notation







Le diagramme de timing est un nouveau diagramme UML 2.0 utile pour représenter les contraintes temporelles (électronique)

2.14 - Langage d'expression de contraintes sur objets : OCL



- Fonctionnalité
- Exemple



UML formalise l'expression des contraintes avec OCL (Object Constraint Language).

OCL est une contribution d'IBM à UML 1.1.

Langage formel simple d'accès qui possède une grammaire élémentaire

OCL permet de décrire des invariants dans un modèle objet, sous forme de pseudo-code :

- pré et post-conditions pour une opération,
- expressions de navigation,
- expressions booléennes, etc...
- langage déclaratif
- invariant
- contraintes

2.14.2 - Complement



<http://laurent-audibert.developpez.com/Cours-UML/?page=object-constraint-langage-ocl>



Club des développeurs et IT pro

[Forums](#) [Tutoriels](#) [Magazine](#) [FAQs](#) [Blogs](#) [Chat](#) [Newsletter](#) [Études](#) [Emploi](#) [Club](#) [Contacts](#)

Recherche person...

[Accueil](#) [ALM](#) [Java](#) [.NET](#) [Dév. Web](#) [EDI](#) [Programmation](#) [SCBD](#) [Office](#) [Solutions d'entreprise](#) [Applications](#) [Mobiles](#) [Systèmes](#)

[ALM](#) [Merise](#) [UML](#)

[ACCUEIL UML](#) [FORUM UML](#) [F.A.Q UML](#) [TUTORIELS UML](#) [LIVRES UML](#) [OUTILS UML](#)

UML 2

De l'apprentissage à la pratique

[Précédent](#) [Sommaire](#) [Suivant](#)

Table des matières
4. Chapitre 4 Langage de contraintes objet (Object Constraint Language : OCL)
4-1. Expression des contraintes en UML
4-1-1. Introduction
4-1-2. Écriture des contraintes
4-1-3. Représentation des contraintes et contraintes prédéfinies
4-2. Intérêt d'OCL
4-2-1. OCL - Introduction
4-2-1-a. Qu'est-ce OCL ?
4-2-1-b. Pourquoi OCL ?
4-2-2. Illustration par exemple
4-2-2-a. Mise en situation
4-2-2-b. Diagramme de classes
4-3. Typologie des contraintes OCL
4-3-1. Diagramme support des exemples illustratifs
4-3-2. Contexte (context)
4-3-2-a. Syntaxe
4-3-2-b. Exemple
4-3-3. Invariants (inv)
4-3-3-a. Syntaxe
4-3-3-b. Exemple
4-3-4. Préconditions et postconditions (pre, post)
4-3-4-a. Syntaxe
4-3-4-b. Exemple
4-3-5. Résultat d'une méthode (body)
4-3-5-a. Syntaxe
4-3-5-b. Exemple
4-3-6. Définition d'attributs et de méthodes (def et let...in)
4-3-6-a. Syntaxe de let...in
4-3-6-b. Syntaxe de def
4-3-6-c. Exemple

4. Chapitre 4 Langage de contraintes objet (Object Constraint Language : OCL)

4-1. Expression des contraintes en UML

4-1-1. Introduction

Nous avons déjà vu comment exprimer certaines formes de contraintes avec UML :

Contraintes structurelles :

les attributs dans les classes, les différents types de relations entre classes (généralisation, association, agrégation, composition, dépendance), la cardinalité et la navigabilité des propriétés structurelles, etc. ;

Contraintes de type :

typage des propriétés, etc. ;

Contraintes diverses :

les contraintes de visibilité, les méthodes et classes abstraites (contrainte *abstract*), etc.

Dans la pratique, toutes ces contraintes sont très utiles, mais se révèlent insuffisantes. Toutefois, UML permet de spécifier explicitement des contraintes particulières sur des éléments de modèle.

4-1-2. Écriture des contraintes

Une contrainte constitue une condition ou une restriction sémantique exprimée sous forme d'instruction dans un langage textuel qui peut être naturel ou formel. En général, une contrainte peut être attachée à n'importe quel élément de modèle ou liste d'éléments de modèle. Une contrainte désigne une restriction qui doit être appliquée par une implémentation correcte du système.

On représente une contrainte sous la forme d'une chaîne de texte placée entre accolades {}. La chaîne constitue le corps écrit dans un langage de contrainte qui peut être :

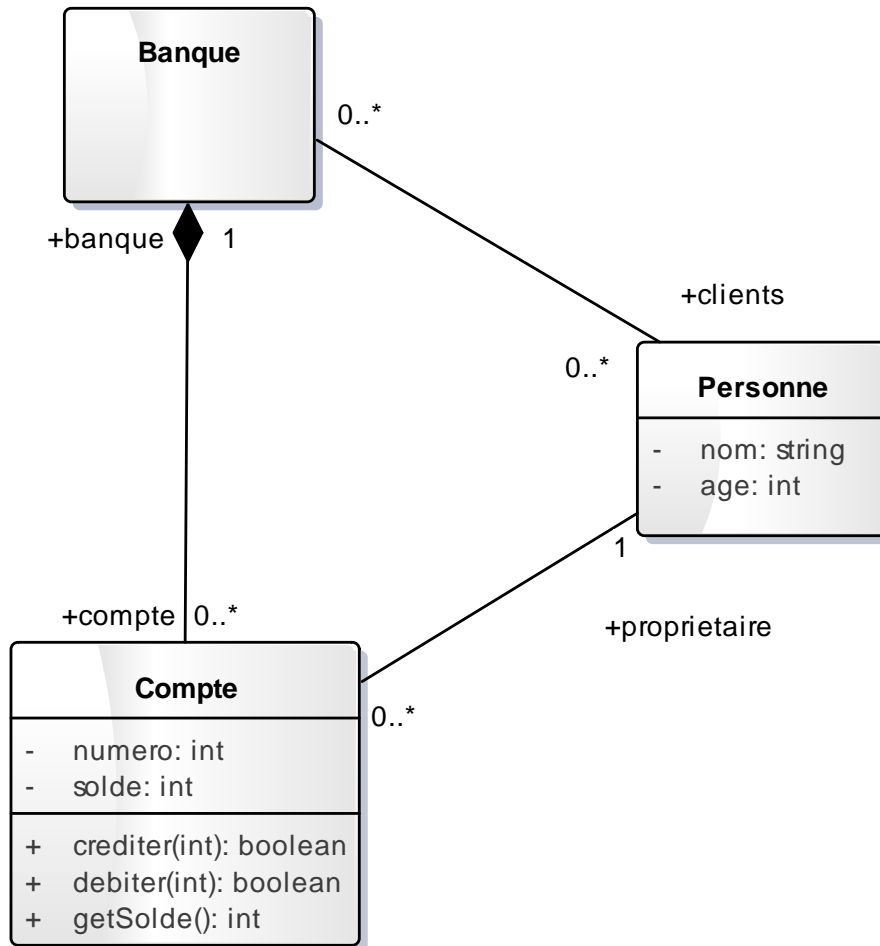
- naturel ;
- dédié, comme OCL ;
- ou encore directement issu d'un langage de programmation.

Si une contrainte possède un nom, on présente celui-ci sous forme d'une chaîne suivie d'un double point (:), le tout précédant le texte de la contrainte.

2.14.3- Exemple



class Domain Model



```

context Compte
inv: banque.clients -> includes (proprietaire)

context Compte : debiter(somme : int)
pre: somme > 0
post: solde = solde@pre - somme

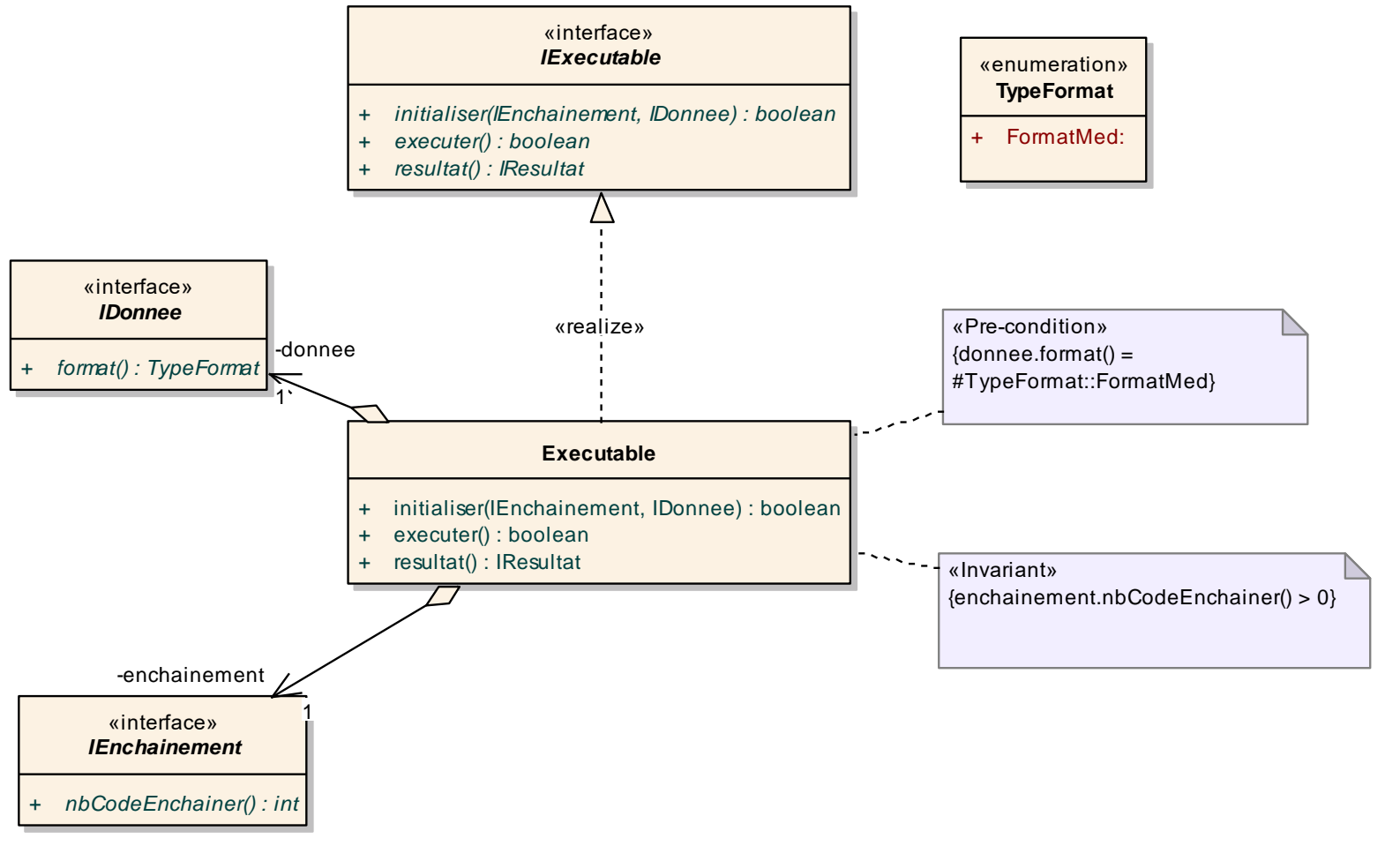
context Compte
inv: proprietaire.age >= 18
inv: solde > 0
proprietaire -> notEmpty()
proprietaire -> size() = 1

context Banque
inv: not( clients -> exists (age < 18) )
  
```

2.14.3 - Exemple



cd Executable



4 – Conclusion



4.1 - Avantages et Inconvénients de l'approche objet

4.2 - Comment utiliser UML



Avantages d'UML

UML est un langage formel et normalisé

- gain de précision
- gage de stabilité
- accélère et structure les développements logiciels
- encourage l'utilisation d'outils

UML est un support de communication performant

- Il cadre l'analyse.
- Il facilite la compréhension de représentations abstraites complexes.
- Son caractère polyvalent et sa souplesse en font un langage universel.

Inconvénients d'UML

La mise en pratique d'UML nécessite un apprentissage et passe par une période d'adaptation.

Le processus (non couvert par UML) est une autre clé de la réussite d'un projet.

.



Multipliez les vues sur vos modèles !

Un diagramme n'offre qu'une vue très partielle et précise d'un modèle.
Croisez les vues complémentaires (dynamique / statique)

Restez simple !

Utilisez les niveaux d'abstraction pour synthétiser vos modèles
(ne vous limitez pas aux vues d'implémentation).

Ne surchargez pas vos diagrammes.

Commentez vos diagrammes (notes, texte...).

Utilisez des outils appropriés pour réaliser vos modèles !

5 – Bibliographie



- Modélisation objet avec UML, P. Muller, N. Gaertner, Eyrolles
- UML 2.0, M. Fowler, Campus Press
- UML 2.0, Grady Booch, Ivar Jacobson, James Rumbaugh, Campus Press
- L'eXtreme Programming, J.Bénard, L.Bossavit, R.Médina, D.Williams , Eyrolles
- Guide pratique du RUP, P. Kruchten, P. Kroll , Campus Press
- RUP, XP, architectures et outils, P.Cloux , Dunod
- Design Patterns, E.Gamma R.Helm, Vuibert