CHAPITRE 1 - INTRODUCTION A UML

Table des matières

1.1 - DÉFINITION

UML : Unified Modeling Language (Langage de Modélisation Unifié).

UML est un langage de modélisation.
Ce n'est pas une méthode, une démarche.

Les outils d'UML permettent :

- ✔ de décrire un problème, un état (Analyse),
- ✔ de décrire une solution (Conception),
- ✔ de proposer une solution (Réalisation),
- ✔ (de modéliser la maintenance et l'évolution du système).

Il y a donc un avant ou un présent et un après, un futur donc un axe temporel.
Il y a aussi un niveau système et un niveau informatique donc un axe d'abstraction.

Abstraction/Temps	Présent	Futur
Système	Analyse	Conception
Logiciel	(code)	Réalisation

UML s'appuie sur des modèles représentés - principalement - par des diagrammes utilisant des formalismes graphiques mais aussi par des modèles textuels.

UML 2.x propose 13 diagrammes (UML 1.X en comportait 9).

1.2 - LA DÉMARCHE : 2TUP

A côté d'UML il faut bien une méthode – une démarche – pour mettre en place les modèles UML pour arriver à la production du logiciel en partant des besoins.
UP, **Unified Process** (Processus Unifié) est une méthode de développement pour les logiciels orientés objets créée par Ivar Jacobson (un des fondateurs d'UML) en 1995.

Le processus unifié est un processus de développement logiciel : il regroupe les activités à mener pour transformer les besoins d'un utilisateur en système logiciel.

Caractéristiques essentielles du processus unifié :

- Le processus unifié est à base de composants,
- Le processus unifié utilise le langage UML (ensemble d'outils et de diagrammes),
- Le processus unifié est piloté par les cas d'utilisation,
- Le processus unifié est centré sur l'architecture,
- Le processus unifié est itératif et incrémental.

2TUP (Two Tracks Unified Process de Valtech créé par Pascal Roques) est parente de RUP (Rational Unified Process, 1998).

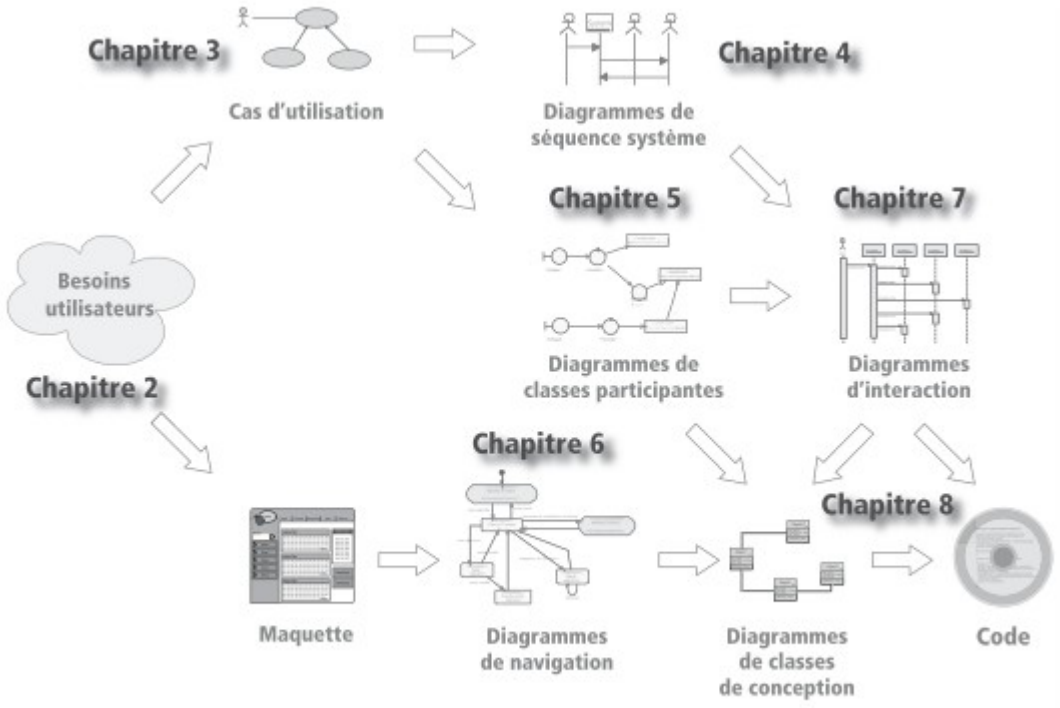


Figure 1–21 Les chapitres du livre replacés dans la démarche globale de modélisation

Schéma emprunté à « UML 2, Modéliser une application web » de Pascal Roques

1.3 - PRINCIPES DE LA MODÉLISATION

Un modèle est une représentation partielle de la réalité :

- ✔ C'est une abstraction de ce qui est intéressant dans un contexte donné,
- ✔ C'est une vue subjective et simplifiée d'un système,
- ✔ UML s'intéresse principalement aux modèles d'applications informatiques (un modèle est un ensemble de diagrammes UML).

Utilité des modèles :

- ✔ Faciliter la compréhension d'un système,
- ✔ Permettre la communication avec le client et les concepteurs-développeurs (communication, documentation),
- ✔ Définir voire simuler le fonctionnement d'un système (la précision est requise pour permettre la production de code).

Critères de qualité d'un système logiciel :

- ✔ Validité (répond aux besoins des utilisateurs),
- ✔ Facilité d'utilisation,
- ✔ Performance (temps de réponse, débit),
- ✔ Fiabilité (tolérant aux pannes),
- ✔ Sécurité (intégrité des données, protection des accès),
- ✔ Maintenabilité (facile à corriger ou à transformer),
- ✔ Portabilité (adaptable au changement d'environnement matériel ou logiciel).

Les principes de l'ingénierie logicielle :

- ✔ Rigueur : principale source d'erreurs humaines, s'assurer par tous les moyens que ce qu'on écrit est bien ce qu'on veut dire et que ça correspond à ce qu'on a promis (outils, revue de code),
- ✔ Abstraction : extraire des concepts, puis instancier les solutions sur les cas particuliers,
- ✔ Décomposition en sous-problèmes : traiter chaque aspect séparément pour simplifier,
- ✔ Modularité : partition du logiciel en modules interagissant, remplissant une fonction et ayant une interface cachant l'implantation aux autres modules,
- ✔ Construction incrémentale : construction pas à pas, intégration progressive,
- ✔ Généricité : proposer des solutions générales pour pouvoir les réutiliser et les adapter,
- ✔ Anticipation des évolutions : liée à la généricité et à la modularité, prévoir les ajouts/ modifications possibles de fonctionnalités,
- ✔ Documentation : pour le suivi de projet et la communication,
- ✔ Standardisation/Normalisation : aide à la communication pour le développement, la maintenance et la réutilisation.

Premier aperçu de la démarche :

Ensemble d'activités successives, organisées en vue de la production d'un logiciel :

- ✔ Analyse des besoins,
- ✔ Spécification,
- ✔ Conception,
- ✔ Programmation,
- ✔ Validation et vérification,
- ✔ Livraison.

Pour plus de détails cf les documents UP (Unified Process), RUP (Rational Unified Process d'IBM), 2TUP (Two Tracks Unified Process de Valtech), Agile, SCRUM, XP (eXtreme Programming) et MDA (Model Drive Architecture).

Notes :
l'ordre Programmation/Validation est inversé dans le cadre du Test Driven Development (TDD) – Développement Piloté par les Tests - (cf les méthodes/démarches X-Unit).

(*) Two Tracks Unified Process (track = rail), Instanciation de UP proposée par Valtech prenant en compte les aléas et contraintes liées aux changements perpétuels et rapides des SI des entreprises.

1.4 - DÉMARCHES

1.4.1 - Pascal Roques : UML et méthode AGILE

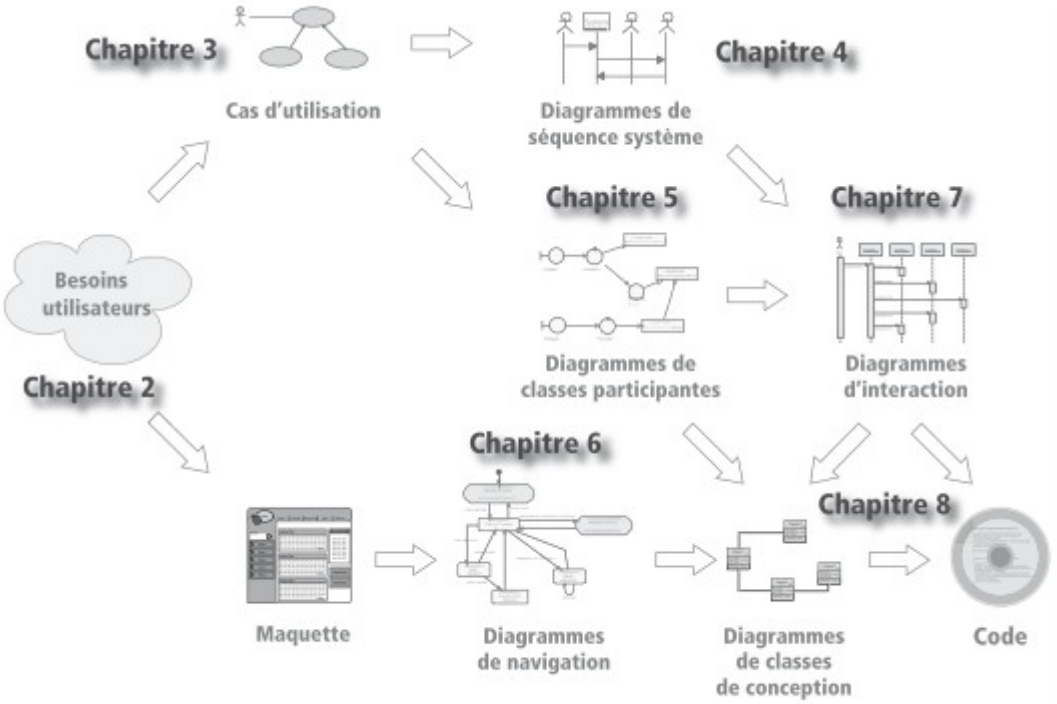
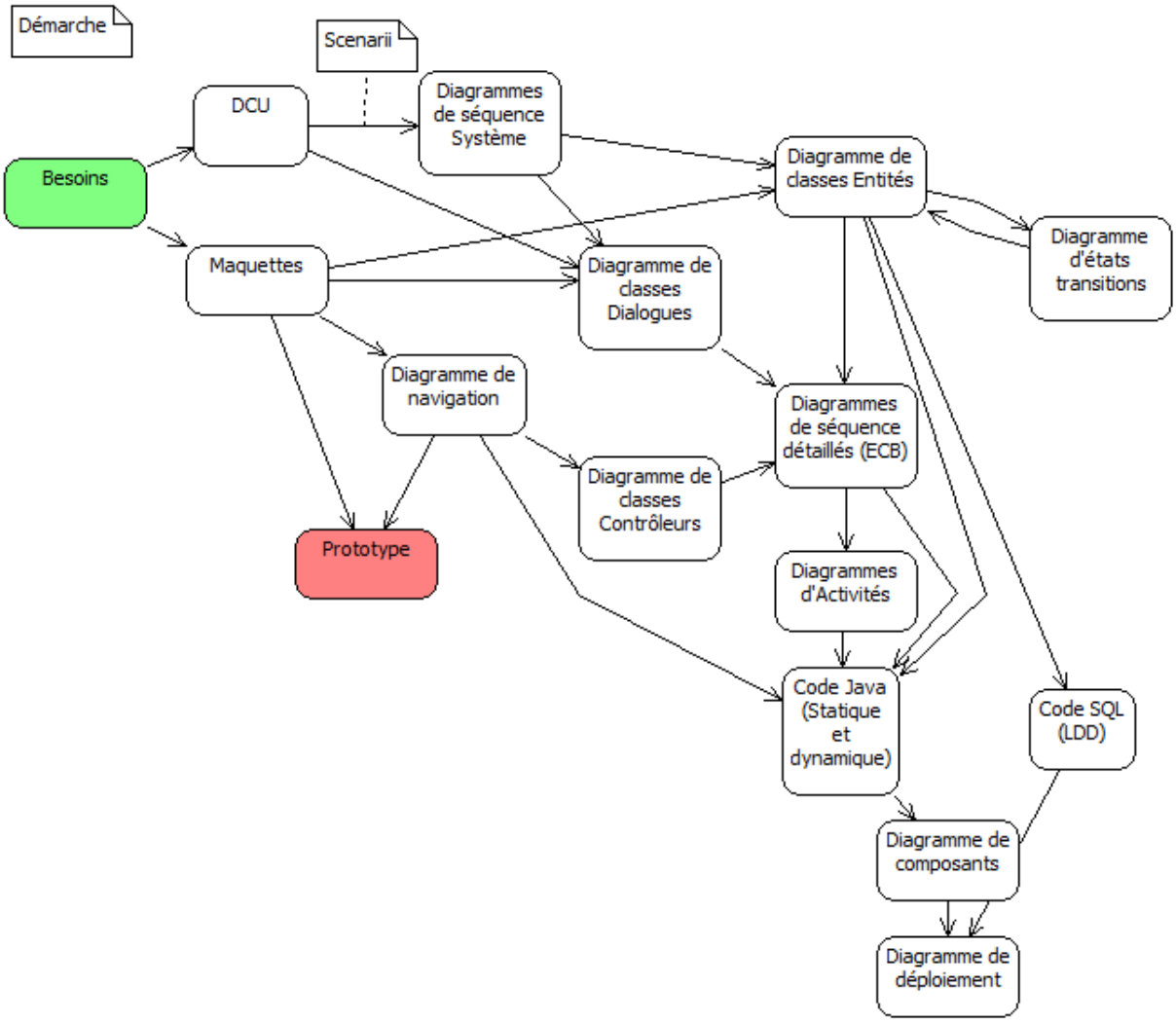


Figure 1–21 Les chapitres du livre replacés dans la démarche globale de modélisation

Schéma emprunté à Pascal Roques, « UML2 - Modéliser une application web » édité chez Eyrolles.

1.4.2 - Autre démarche

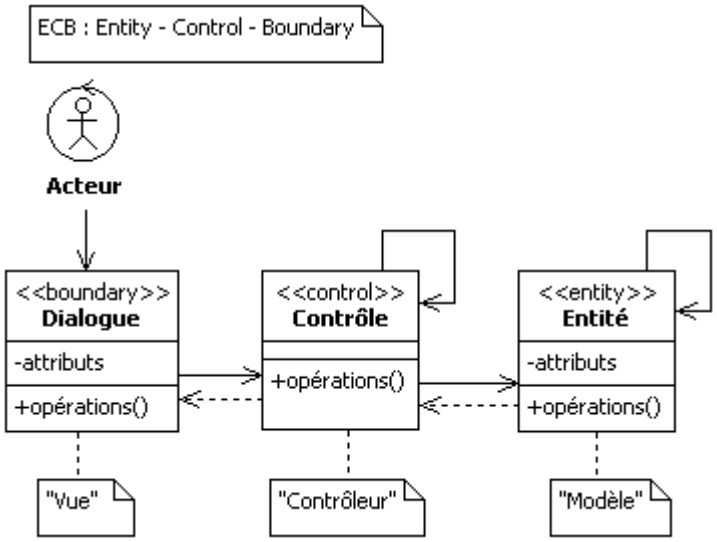
Cf generalites.uml.



1.4.3 - UML et ECB

Ce paragraphe illustre les inter-actions entre les différents classes et la présence d'attributs et de méthodes dans les différentes catégories de classes.

L'utilisateur du système interagit au travers d'interfaces de dialogue (des écrans).



Règles :

les classes Dialogue possèdent des attributs (champs de saisie, champs résultats donc éventuellement ce sont des attributs dérivés) et des opérations (interactions de l'utilisateur avec l'IHM),
les classes Dialogue sont associées à des classes Contrôle ou à d'autres classes Dialogue.

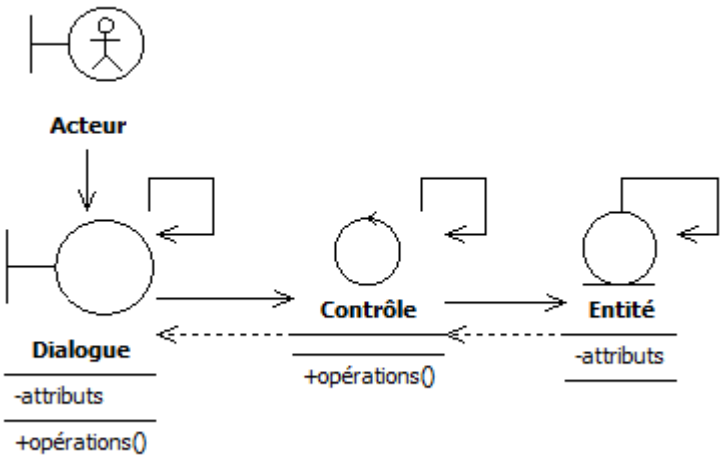
Les classes Contrôle possèdent des opérations (elles représentent la logique de l'application, la navigation entre écrans, la récupération d'informations des classes de persistance, etc),
une classe Contrôle correspond le plus souvent à un cas d'utilisation,
les classes Contrôle sont associées à toutes les autres classes.

Les classes Entité possèdent des attributs,
les classes Entité sont associées à des classes Contrôle ou à d'autres classes Entités.

Discussion :

Pascal Roques, "UML2 – Modéliser une application WEB", page 111.
La liaison entre classes Dialogue est déconseillée selon certains auteurs.
La liaison entre classes Entité est déconseillée selon certains auteurs.
La liaison entre classes Entité n'est possible que si les entités possèdent des opérations.

Variation avec des représentations iconiques (cf le diagramme de classes participantes).



1.5 - LES ARCHITECTURES APPLICATIVES

1.5.1 - Les différentes architectures

L'architecture est l'ensemble des décisions d'organisation du système logiciel qui défend les intérêts de son propriétaire final. Les intérêts s'expriment en termes d'exigences fonctionnelles, techniques et économiques. L'architecture y répond par l'intégration de plusieurs styles de développement informatique qu'elle adapte aux éléments logiciels d'un contexte existant.

Les principales architectures sont les suivantes :

- ✔ architecture client/serveur en tiers,
- ✔ architecture en couches,
- ✔ architecture en niveaux (local, départemental, central),
- ✔ architecture à base de composants.

1.5.2 - Les architectures Client/Serveur n-tiers

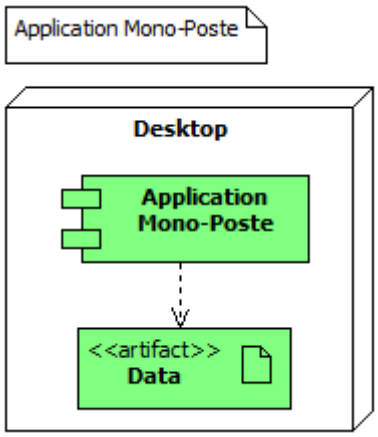
1.5.2.1 - Définition

Les architectures client/serveur en tiers - prononcez … **tier** [tɪə] - (2-tiers, 3-tiers ou n-tiers) concernent la capacité de montée en charge du système. Les architectures 2-tiers visent des applications départementales à nombre limité d'utilisateurs. Elles mettent généralement en jeu des clients et un serveur de base de données. Les architectures 3-tiers ou n-tiers permettent l'évolution du nombre des utilisateurs par l'introduction d'un middleware, qui distribue les services entre les clients et les serveurs.

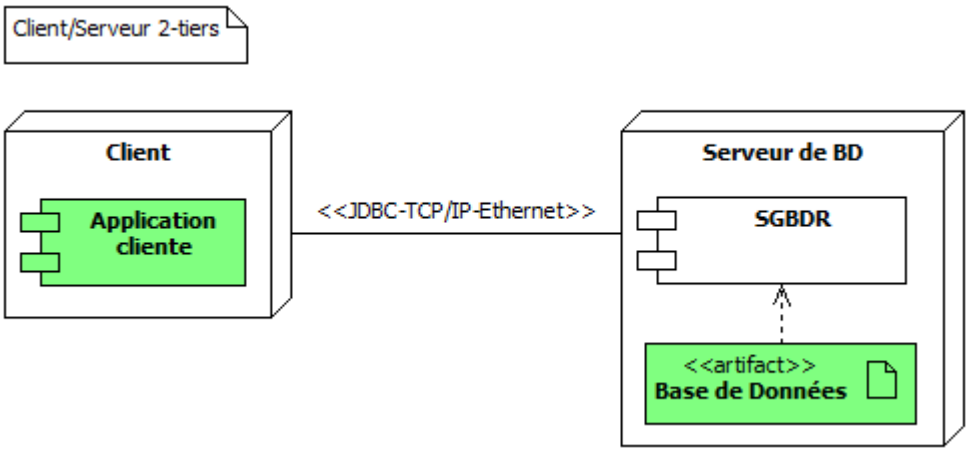
Cf Architectures_Applicatives.uml

1.5.2.2 - Application Mono-Poste

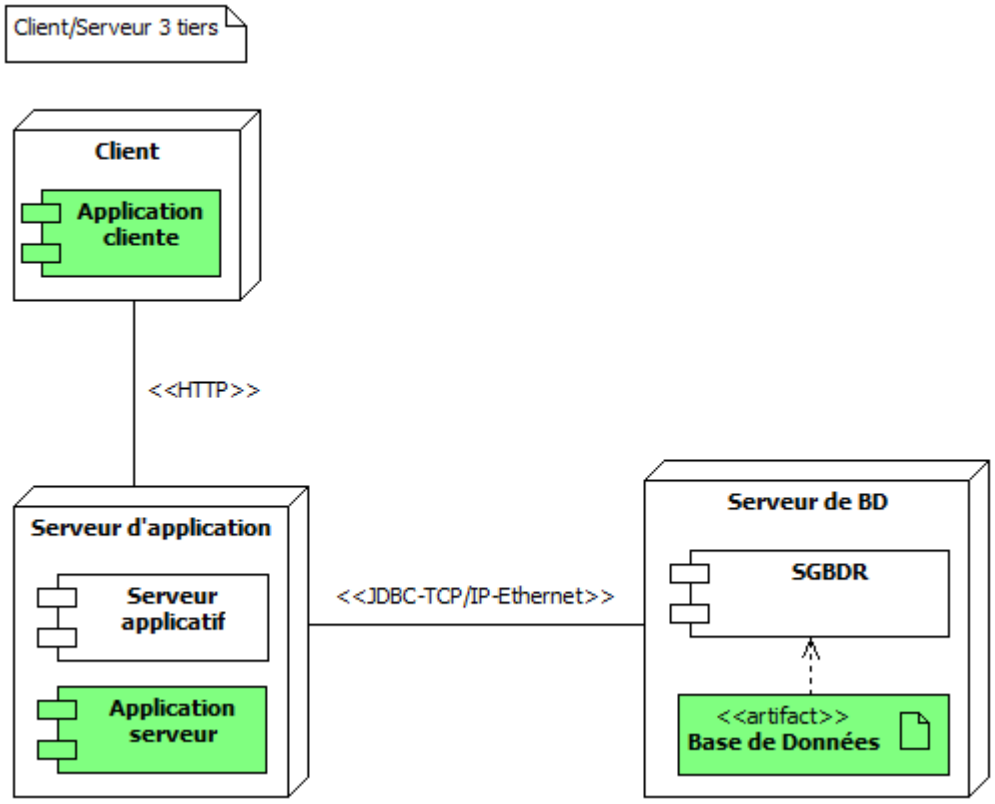
Pour rappel !



1.5.2.3 - Application Client/Serveur 2-tiers



1.5.2.4 - Application Client/Serveur 3-tiers



1.5.3 - Les architectures multi-niveaux

Les architectures en niveaux correspondent au déploiement des fonctions sur les postes de travail des utilisateurs. Les 3 niveaux considérés pour l'entreprise sont le niveau central, le niveau départemental et le niveau local.

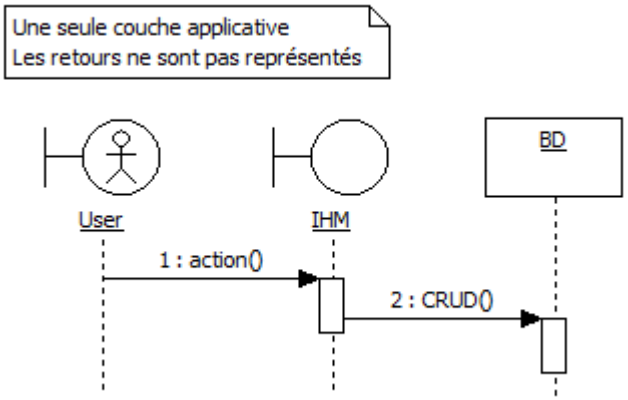
1.5.4 - Architectures multi-couches

1.5.4.1 - Définition

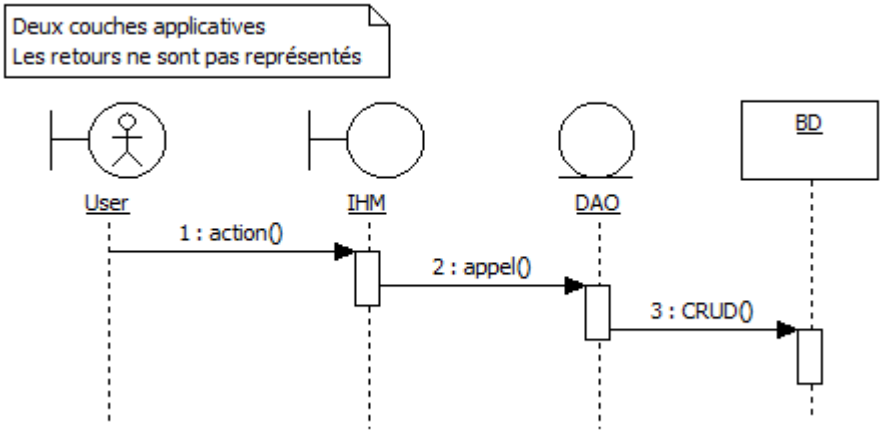
Les architectures en couches visent la distribution des responsabilités techniques sur les parties développées pour le système logiciel. L'architecture la plus fréquente est la répartition en cinq couches (si l'on considère la Base de Données comme une couche) : présentation, application, métier, accès aux données et stockages des données.

Pas à pas vers une architecture multi-couches à 5 niveaux.

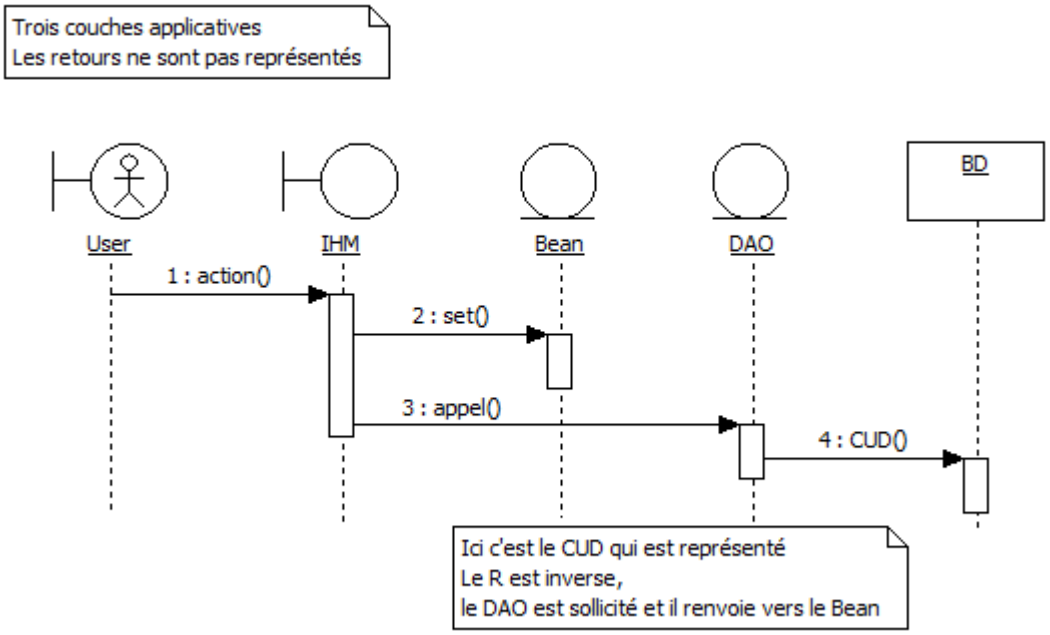
1.5.4.2 - Une couche

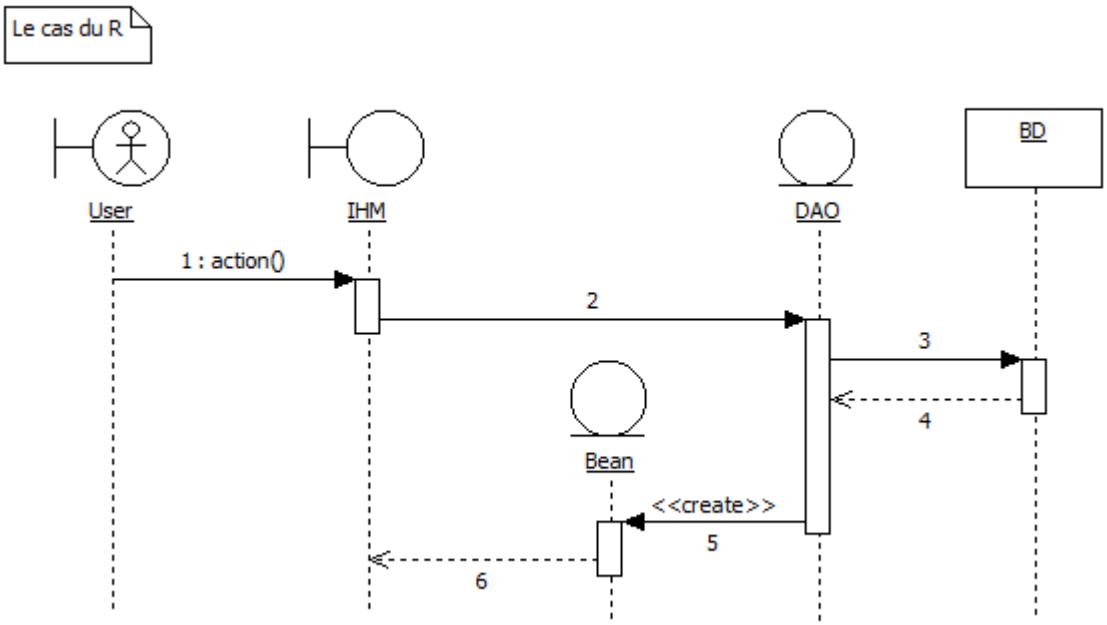


1.5.4.3 - Deux couches

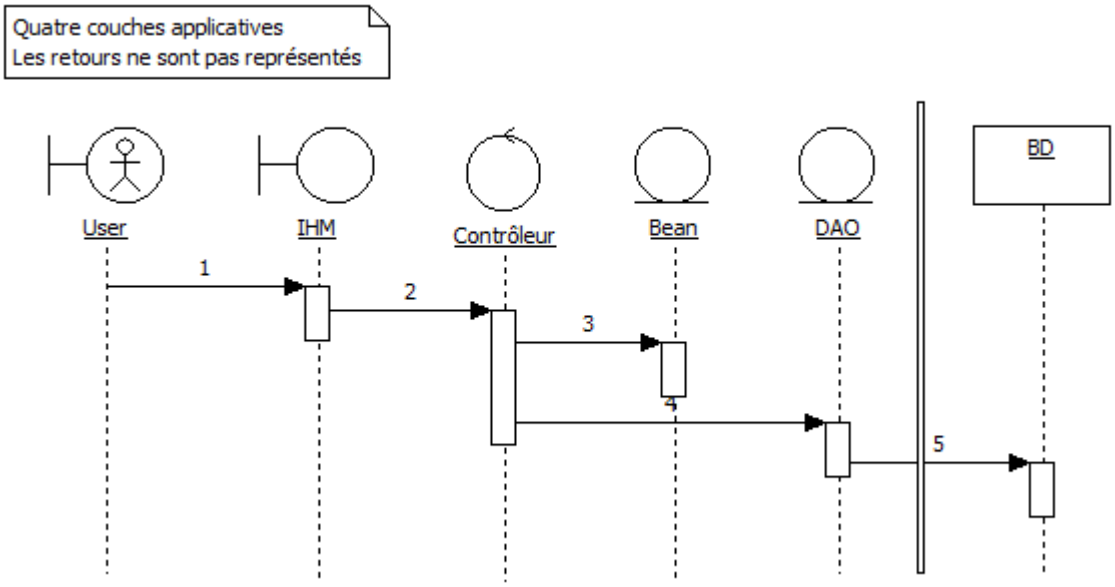


1.5.4.4 - Trois couches

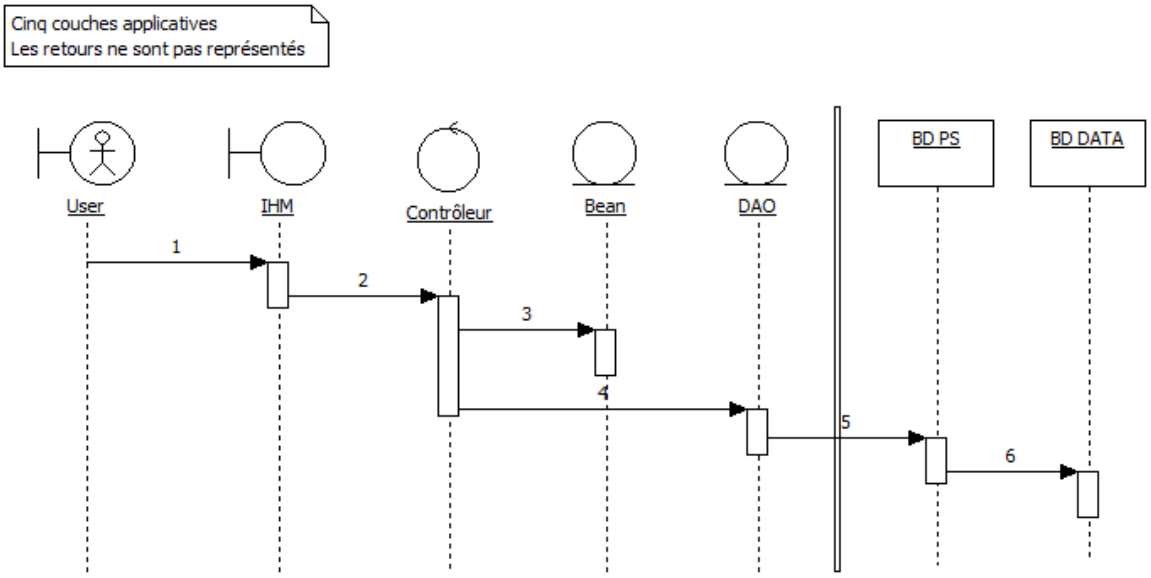




1.5.4.5 - Quatre couches



1.5.4.6 - Cinq couches



1.5.4.7 - Encore plus !

Pensez que la couche IHM est décomposable au minimum en 2 couches (la couche statique et la couche dynamique - interactive par exemple en HTML avec JavaScript) et souvent en 3 couches si l'on applique le pattern MVC (Swing de Java – JTable, DefaultTableModel, event - , AngularJS, etc).