ARCHITECTURES LOGICIELLES

Aperçu des principes généraux des architectures logicielles

Par : Vanessa GIACOMONI

I.GENERALITES

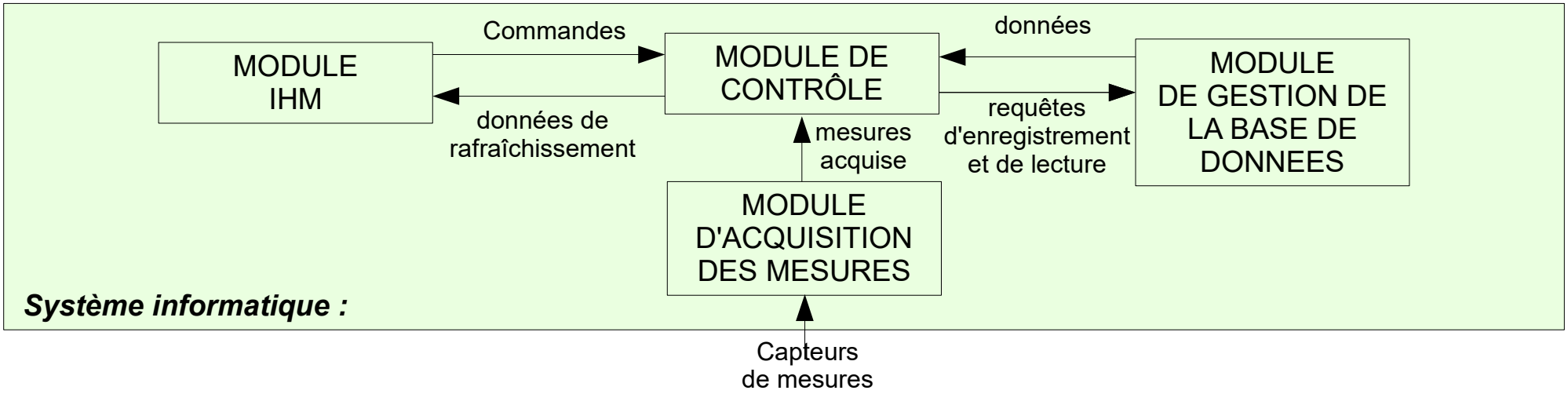
I.1.DEFINITION:

L'ARCHITECTURE d'un logiciel est un concept représentatif de la manière dont un logiciel est conçu et agencé afin de remplir les fonctions qui lui sont assignées.

Alors que les SPÉCIFICATIONS FONCTIONNELLES ET TECHNIQUES, produites pendant la phase d'ANALYSE DU BESOIN ont pour objectif de répondre "QUOI FAIRE" et "POURQUOI", le modèle d'ARCHITECTURE, produit lors de la phase de CONCEPTION, décrit COMMENT il doit être conçu pour répondre aux spécifications.

L'ARCHITECTURE LOGICIELLE décrit de manière SYMBOLIQUE et SCHÉMATIQUE les différents ÉLÉMENTS (modules, composants) d'un ou de plusieurs systèmes informatiques et les INTERACTIONS qui existent entre ces éléments.

EXEMPLE:



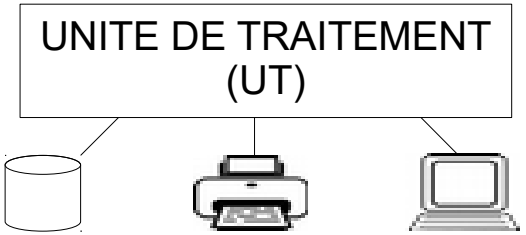
- Ce schéma d'architecture logicielle représente les relations existant entre des entités appelées MODULES. On voit que ces modules ont été délimités en fonction de leur cohérence fonctionnelle (chaque module logiciel réalise une FONCTION du logiciel) : ce n'est qu'un critère parmi d'autres.
- Les INTERACTIONS représentées entre ces modules peuvent être de diverses natures : échange de données, échange de commandes ou de requêtes, etc.

I.GENERALITES

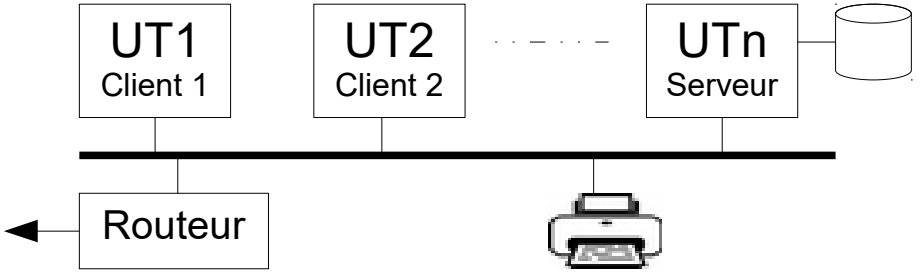
I.2.STRUCTURATION DES LOGICIELS:

Au début de l'évolution de l'informatique (années 1950-1960), les systèmes informatiques étaient dans leur immense majorité, composés d'une seule UNITE DE TRAITEMENT (UT), connectée à divers périphériques. Les applications s'exécutaient donc dans UNE SEULE MACHINE (Architecture MAIN FRAME).

Avec l'apparition des RESEAUX LOCAUX et des RESEAUX ETENDUS (années 1980), l'architecture des systèmes informatique a évolué de plus en plus vers des SYSTEMES REPARTIS composés de plusieurs UT communicant par le réseau. L'architecture répartie est maintenant largement majoritaire.

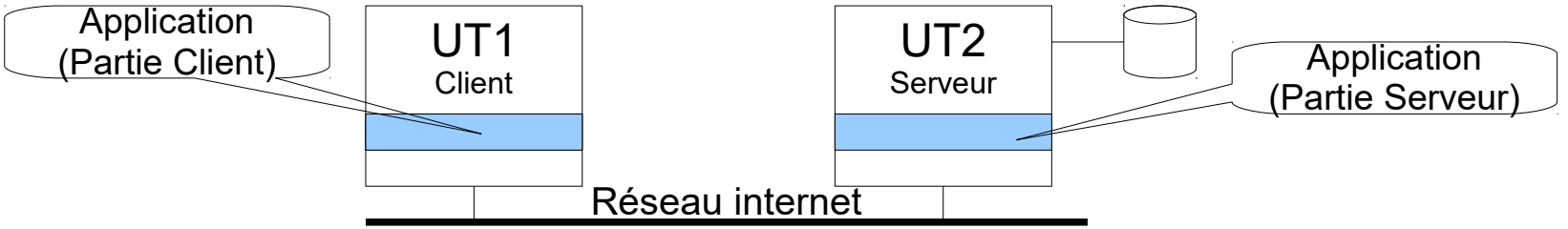


Architecture "main frame"



Architecture "Répartie"

Cette évolution a entraîné l'apparition d'APPLICATIONS REPARTIES sur plusieurs unités de traitement. Les SITES WEB en sont les exemples les plus connus : une application web est répartie au moins sur deux unités de traitement : le Client WEB (navigateur) et le Serveur WEB.



Architecture "Répartie" : site WEB

I.GENERALITES

I.2.STRUCTURATION DES LOGICIELS:

De ce fait, la structuration d'un logiciel doit de nos jours être abordée sous différents angles, selon que l'on considère :

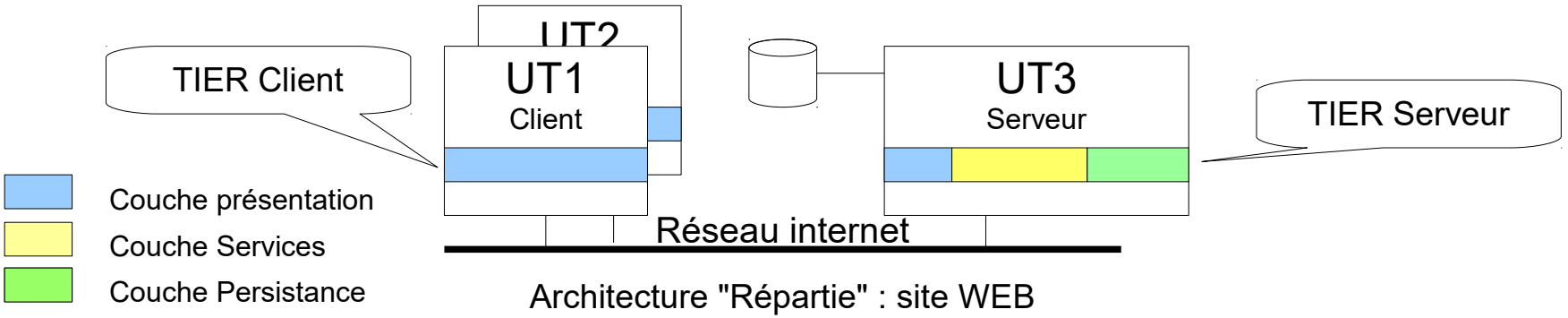
- L'architecture LOGIQUE, faisant abstraction du contexte matériel d'exécution (machines, OS et réseaux) ;
- L'architecture PHYSIQUE qui prend en compte le contexte d'exécution, c'est à dire les composants matériels qui sont sollicités par chaque entité logicielle composant l'application ;

On peut donc distinguer au moins deux types de VUES ARCHITECTURALES :

- La VUE EN COUCHE (LAYER VIEW) qui représente l'architecture LOGIQUE montrant le découpage des fonctions de l'application, indépendante des considérations physiques.
- La VUE EN NIVEAUX (TIER VIEW), qui représente l'architecture PHYSIQUE, tenant compte de la REPARTITION des entités logicielles sur les différentes UNITES DE TRAITEMENT composant le réseau.

REMARQUES :

- Le terme anglais TIER signifie ETAGE ou NIVEAU: il n'a donc aucun rapport avec les proportions.
- Il désigne un concept différent du concept de couche logicielle : on peut dire qu'un TIER désigne un NIVEAU DE SERVICE.
- Une COUCHE peut être répartie sur plusieurs TIER : ainsi, dans un site web classique, la couche PRESENTATION est répartie sur le Client et sur le Serveur.
- Un TIER peut être réparti sur plusieurs UT : ainsi, un TIER Client est présent sur tous les postes des utilisateurs connectés.



Architecture "Répartie" : site WEB

II.LA VUE EN COUCHES LOGICIELLES

II.1.PRINCIPES GENERAUX:

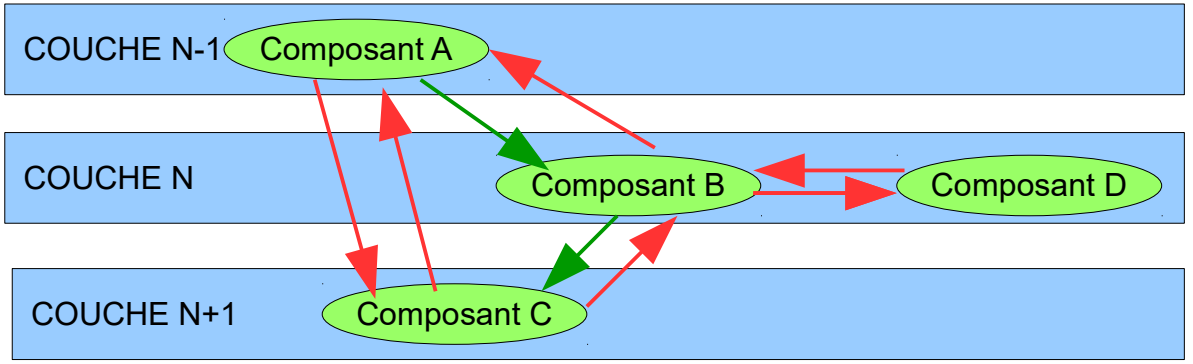
La structuration en couches consiste à répartir les différentes entités composant un logiciel en différentes COUCHES (LAYERS) en respectant les principes suivants :

PRINCIPE N° 1 : les composants sont répartis dans les couches en fonction de la spécificité des traitements qu'ils supportent :

- Les traitements spécifiques de l'application doivent occuper la couche supérieures (En particulier, les IHM doivent occuper la couche supérieure) ;
- MOINS un composant est spécifique à l'application, PLUS il doit être relégué dans une couche inférieure. Ainsi, les composants renfermant des traitements d'intérêt général, susceptibles d'être utilisés dans de nombreux contextes, doivent être placés dans la couche la plus basse.

PRINCIPE N° 2 : Une COUCHE ne peut intéragir qu'avec une couche ADJACENTE : la couche supérieure envoie une REQUETE à la couche inférieure, qui exécute le traitement demandé et renvoie un MESSAGE DE COMPTE-RENDU (un message de compte-rendu peut contenir une donnée résultat, mais celle-ci doit être de faible volume).

PRINCIPE N° 3 : Un composant ne peut pas intéragir avec un composant de sa propre couche.



REMARQUES :

- Les flêches vertes représentent les requêtes autorisées
- Les flêches rouges représentent les requêtes interdites
- Les réponses à ces requètes ne sont pas représentées dans ce schéma. Tout logiciel recevant une requète autorisée peu y répondre.

II.LA VUE EN COUCHES LOGICIELLES :

II.2.COMPOSITION DES COUCHES LOGICIELLES:

II.2.1.INTRODUCTION :

Les COMPOSANTS d'une COUCHE LOGICIELLE peuvent être de 2 types, correspondant à deux STYLES de conception : la CONCEPTION PROCEDURALE et la CONCEPTION OBJET (On dit aussi programmation procédurale et programmation par objets).

II.2.2.CONCEPTION PROCEDURALE :

La démarche de conception procédurale consiste à prendre pour principes directeurs les TRAITEMENTS A EFFECTUER pour résoudre le problème posé. Par exemple, pour concevoir un éditeur de texte, on définit d'abord les traitements qui doivent être mis à disposition des utilisateurs. On obtiendra par exemple :

- Afficher ou réafficher le document à éditer à partir du fichier disque;
- Insérer des caractères dans le texte ;
- Couper une partie du texte ;
- Coller une partie de texte à un point d'insertion donné ;
- Etc.

Ces différents traitements, que l'on choisira le plus indépendants les uns des autres possibles, après programmation, constitueront des composants logiciels (sous-programmes, fonctions) qui seront activées par les commandes de l'IHM. Ils appartiendrons à la couche supérieure du logiciel. A ce niveau, on ne définiera que l'**interface logiciel** de ces composants (paramètres d'appel et paramètres de retour) et les **traitements internes** qu'ils supportent.

II.LA VUE EN COUCHES LOGICIELLES

II.2.COMPOSITION DES COUCHES LOGICIELLES (Suite):

II.2.2.CONCEPTION PROCEDURALE (Suite):

Dans un deuxième temps, on reprendra chacun de ces composants pour en élaborer la **structure interne**.

Supposons que l'on analyse un composant C : on pourra à cette occasion identifier des traitements internes au composant étudié qu'il serait judicieux d'encapsuler dans des composants de niveau inférieur :

- Soit parce que ces traitements sont utilisés à plusieurs reprise par le composant C (FACTORISATION) ;
- Soit parce qu'ils sont susceptibles d'être utilises par d'autres applications (REUTILISATION) ;
- Soit parce qu'ils sont très complexes ou très techniques et que, de ce fait, cela permettrait d'alléger la programmation et la compréhension de C (ABSTRACTION ou ENCAPSULATION) ;
- Il existe d'autres critères que nous n'aborderons pas ici.

EXEMPLE :
Supposons que le composant C encapsule le traitement "Couper une partie du texte". L'analyse du traitement a effectuer va conduire à la décomposition suivante :

- Récupérer le texte que l'utilisateur a sélectionné sur l'écran ;
- Déterminer le début D et la fin F de ce texte dans le texte total du document ;
- Supprimer dans le texte total du document les caractères compris entre D et F inclus ;
- Réafficher le texte total corrigé.

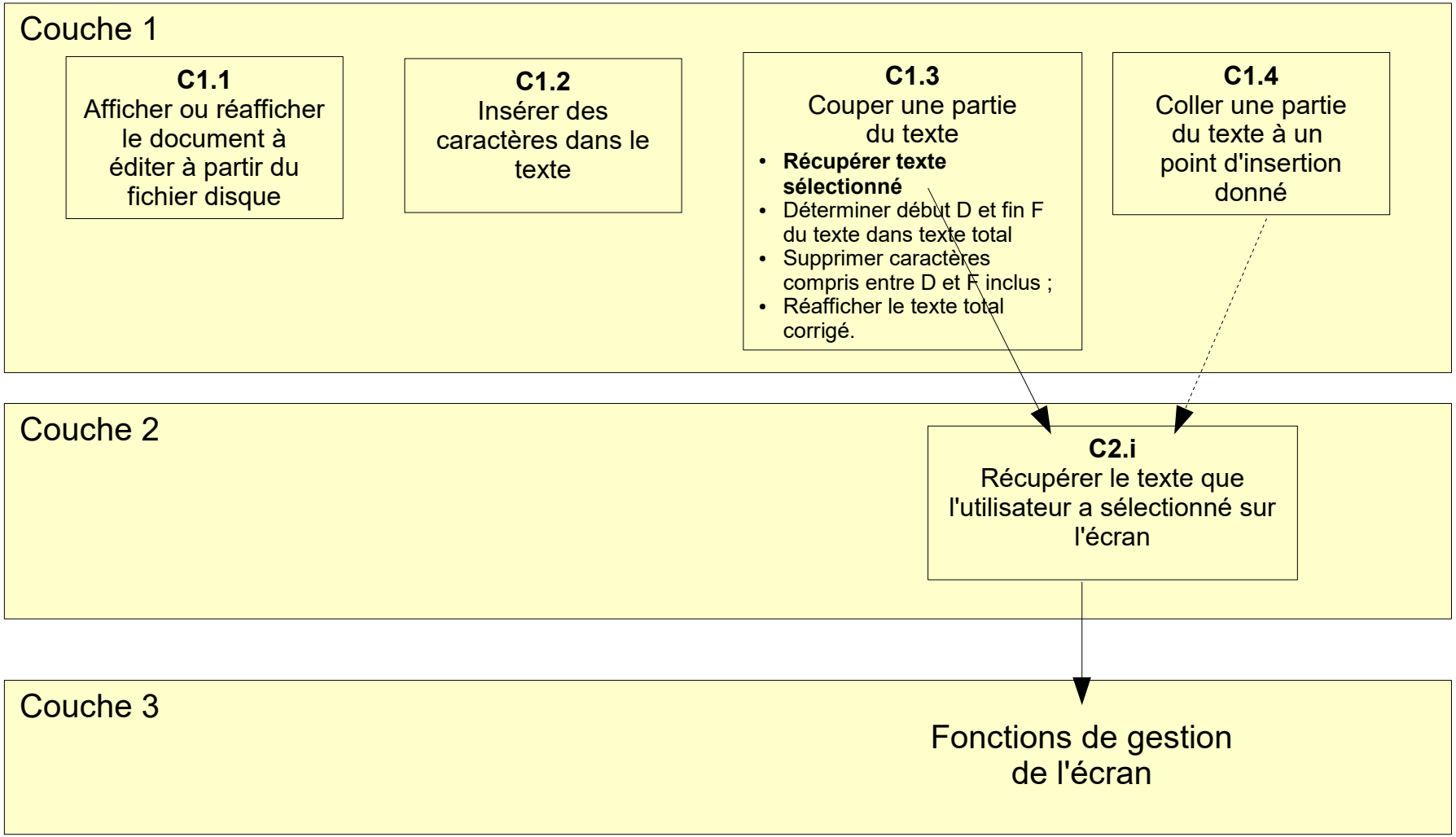
Le premier traitement : "récupérer le texte que l'utilisateur a sélectionné sur l'écran" va faire appel à des traitements de bas niveau, proches du hardware et non spécifiques de l'application. D'autre part, on peut penser que ce traitement peut être nécessaire non seulement à d'autres endroits de l'application (par exemple, dans le cadre du traitement de "copier"), mais aussi dans le cadre d'autres applications. Il est donc judicieux d'encapsuler ce traitement pour un faire un composant de la couche inférieure.

II.LA VUE EN COUCHES LOGICIELLES :

II.2.COMPOSITION DES COUCHES LOGICIELLES (Suite):

II.2.2.CONCEPTION PROCEDURALE (Suite):

A ce stade de la démarche d'analyse, nous pouvons la représenter par le schéma suivant :



II.LA VUE EN COUCHES LOGICIELLES :

II.2.COMPOSITION DES COUCHES LOGICIELLES (Suite):

II.2.2.CONCEPTION OBJET:

CARACTERISTIQUES :

La conception PROCEDURALE a l'avantage d'être très INTUITIVE. Elle a pour inconvénient de ne pas prendre en compte les DONNEES (du moins au début de la conception). Or, les données traitées par une application sont souvent plus STABLES que les traitements : Dans un traitement de texte, les FONCTIONS offertes à l'utilisateur évoluent rapidement dans le temps (Ajouts, perfectionnements, etc.) alors que les FORMATS des fichiers restent beaucoup plus stables).

La CONCEPTION PAR OBJETS, au contraire, a pour caractéristique de tenir compte des DONNEES en même temps que des TRAITEMENTS : en effet, un OBJET encapsule à la fois des TRAITEMENTS et les DONNEES concernées par ces traitements.

De plus, la CONCEPTION OBJET cherche à produire des composants REUTILISABLES : en effet, elle ne crée par des composants UNIQUES mais des MODELES DE COMPOSANTS destinés à être déclinés dans d'autres application. Ces MODELES sont appelés des CLASSES d'objets.

Dans une conception OBJET, les composants des différentes couches sont des CLASSES.

LA NOTION D'OBJET :

Un OBJET est une entité logicielle qui encapsule à la fois :

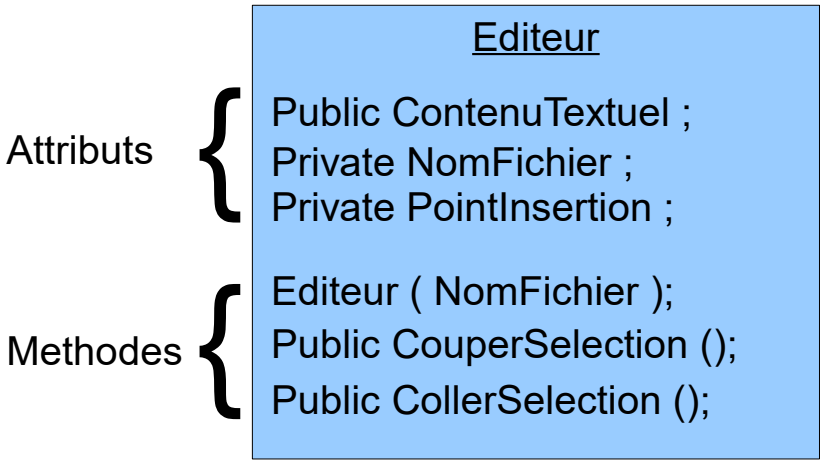
- Des procedures de traitement appelées METHODES : ce sont des sous-programmes pouvant être appelés de diverses façons par l'objet lui-même ou par les autres objets de l'application (si ces methodes sont déclarées PUBLIQUES). Un objet est donc une BIBLIOTHEQUE DE METHODES ;
- Des données associées à ces traitement, appelées ATTRIBUTS. Les méthodes de l'objet, mais aussi par les autres objets de l'application (si ces attributs sont déclarés PUBLICS) peuvent accéder aux données contenues dans ces ATTRIBUTS ;

II.LA VUE EN COUCHES LOGICIELLES :

II.2.COMPOSITION DES COUCHES LOGICIELLES (Suite):

II.2.2.CONCEPTION OBJET (suite):

STRUCTURE D'UNE CLASSE D'OBJET :



Nous avons vu précédemment qu'une CLASSE n'était pas un OBJET, mais un MODELE D'OBJET. Ainsi, la classe ci-contre est le modèle d'un objet implantant un éditeur de textes.

Une CLASSE n'est donc pas exécutable en elle-même. Pour créer à partir de cette classe un OBJET exécutable, il faut l'INSTANCIER. L'instanciation crée un code objet exécutable dans la mémoire de l'ordinateur.

La méthode Editeur, qui a la particularité d'avoir le même nom que l'objet est le CONSTRUCTEUR de la classe. C'est elle qui permet de créer un objet à partir de la classe. Le constructeur est donc appelé lors de l'**instanciation** d'un objet, avec des paramètres qui permettent de déterminer quel objet doit être crée. Ainsi, si l'on veut éditer le fichier texte c://MesDocuments/MonTexte.txt, on écrira une instruction de ce type (ex : en PHP) :

$MonEditeur = new Editeur (c://MesDocuments/MonTexte.txt) ;

L'instruction ci-dessus crée l'objet $MonEditeur à partir de la classe Editeur. L'opérateur new est l'opérateur d'INSTANCIATION dans de nombreux langages.

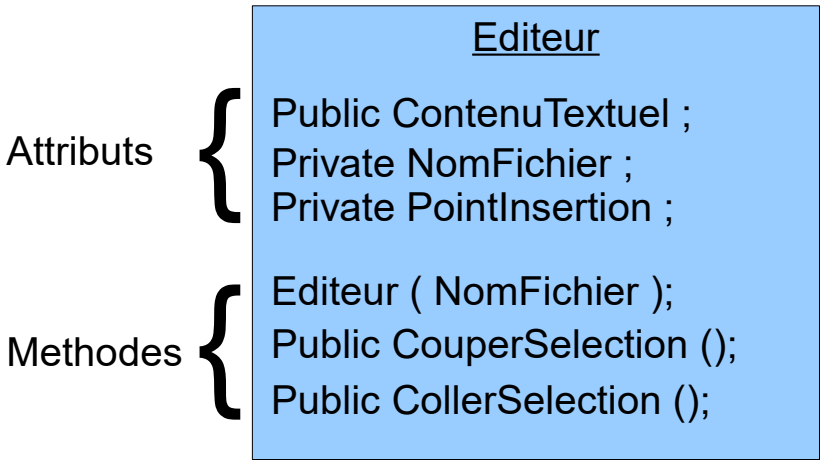
Le mot clef PUBLIC placé devant la déclaration d'une METHODE ou d'un ATTRIBUT indique que cette méthode ou cet attribut sont accessibles depuis d'autres objets que l'objet instancié par cette classe. Le mot clef PRIVATE interdit l'accès aux autres objets. Le CONSTRUCTEUR est évidemment toujours PUBLIC.

II.LA VUE EN COUCHES LOGICIELLES :

II.2.COMPOSITION DES COUCHES LOGICIELLES (Suite):

II.2.2.CONCEPTION OBJET (Suite):

CODAGE D'UNE CLASSE D'OBJET :



EN PHP :

Le mot clef **class** indique que ce qui suit est la déclaration de la classe (Ici, classe Editeur) ;

Dans le code objet qui utilisera un objet de cette classe, il faudra d'abord instancier l'objet :

```
$MonObjet = new Editeur ( "./MonDocument.txt" );
```

Une fois l'objet instancié, je pourrai appeler les méthodes grâce à l'opérateur → . Exemple :

Appel CouperSelection : `$MonObjet->CouperSelection();`

Accès à la valeur de ContenuTextuel : `$MonObjet->ContenuTextuel;`

En PHP :

```
<?php
class Editeur
{
     public $ContenuTextuel ;
     private $NomFichier ;
     private $PointInsertion ;

     // constructeur
     function Editeur ( $nomfichier )
     {
          // corps de la methode
     }

     // methode couperselection
     function CouperSelection ()
     {
          // corps de la methode
     }

     // methode collerselection
     function CollerSelection ()
     {
          // corps de la methode
     }
     etc....
}?>
```

II.LA VUE EN COUCHES LOGICIELLES :

II.2.COMPOSITION DES COUCHES LOGICIELLES (Suite):

II.2.2.CONCEPTION OBJET (Suite):

CLASSES DERIVEES, NOTION D'HERITAGE :

La programmation OBJET permet de créer une CLASSE D'OBJETS à partir d'une classe déjà existante, en RAJOUTANT des methodes ou des attributs ou en REDÉFINISSANT des methodes ou des attributs existants.

La CLASSE ainsi dérivée HERITE de tous les attributs et toutes les methodes de la classe d'origine (éventuellement modifiés), plus les attributs et methodes rajoutés.

Le procédé consistant à redéfinir une méthode ou un attribut s'appelle la SURCHARGE.

EXEMPLE :

Imaginons qu'on veuille dériver de la classe Editeur une autre classe (Editeur1) héritant de Editeur, mais possédant en plus la methode InsererCaractere ($Caractere) Il suffira d'écrire en PHP :

En PHP :

```
<?php
class Editeur1 extends Editeur
{
     // methode InsererCaractere
     function InsererCaractere ( $Caractere)
     {
          // corps de la methode
     }
     etc....
}?>
```

Le mot-clef **Extends** permet à la classe Editeur1 d'HERITER de la classe Editeur.

Editeur1 récupère donc tous les attributs et toutes les methodes de Editeur. De plus, il est doté d'une nouvelle méthode InsererCaractere.

II.LA VUE EN COUCHES LOGICIELLES :

II.2.COMPOSITION DES COUCHES LOGICIELLES (Suite):

II.2.2.CONCEPTION OBJET (Suite):

CONCLUSION :

La démarche de conception par OBJETS est donc assez différente de la démarche PROCEDURALE :

Au lieu de faire un RAFFINEMENT des TRAITEMENTS à effectuer, on cherche à représenter les OBJETS de l'environnement (Ici, un EDITEUR) par des CLASSES D'OBJETS logicielles, auxquelles on affecte :

- D'abord des ATTRIBUTS, qui sont les DONNEES PRINCIPALES à prendre en compte (Ici, le texte du document, l'adresse du fichier de texte, la position du point d'insertion du curseur) ;
- Puis des METHODES agissant sur ces ATTRIBUTS ;

Les classes représentant les objets de l'environnement sont placées dans la couche supérieure du logiciel. Les classes encapsulant des traitements internes à l'application sont placées dans des couches plus profondes.

II.LA VUE EN COUCHES LOGICIELLES :

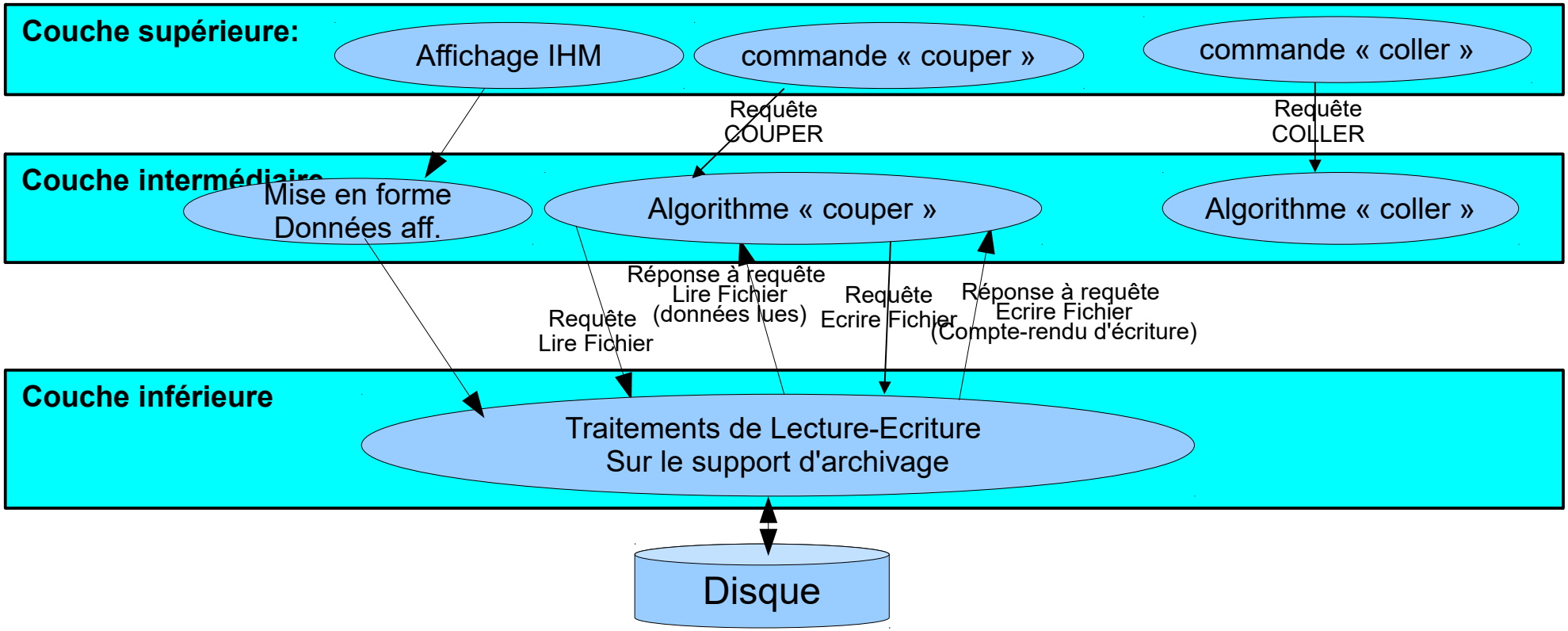
II.2.COMPOSITION DES COUCHES LOGICIELLES (Suite):

II.2.3.AVANTAGES DE LA STRUCTURATION EN COUCHES:

La structuration des applications en couches permet :

- De MAÎTRISER LA COMPLEXITÉ des applications (en évitant les intéractions "dans tous les sens")
- D'OPTIMISER LES TEMPS DE DÉVELOPPEMENT, en favorisant la définition de COMPOSANTS RÉUTILISABLES.
- De FAVORISER LA COMMUNICATION : à l'intérieur d'une application, en structurant les échanges entre les différentes couches.
- De RATIONNALISER et D'UNIFIER L'ARCHITECTURE : Chaque couche a ses propres responsabilités et utilise la couche située en dessous.

EXEMPLE : *Logiciel de traitement de texte*



II.LA VUE EN COUCHES LOGICIELLES :

II.3.MODELES DE CONCEPTION EN COUCHES:

Il existe de nombreux modèles de décomposition en couches logicielles, chacun d'eux étant adapté à un type d'application et au degrès de complexité de cette application.
De plus, le concepteur d'une application peut supprimer certaines couches d'un modèle ou au contraire diviser certaines couches en plusieurs sous-couches en fonction du cas qu'il a a résoudre.

MODELE A TROIS COUCHES :

C'est le modèle MINIMUM. Il peut très bien convenir à une application simple, faiblement ou pas du tout répartie. Il comprend :

- Une couche PRESENTATION renfermant les composants d'affichage et de commandes (IHM) ;
- Une couche TRAITEMENTS contenant les composants METIER de l'application (algorithmes spécifiques à l'application) ;
- Une couche GESTION DES DONNEES ET SERVICES DE BASE comprenant les composants de gestion des BASES DE DONNEES et les composants de service basiques (bibliothèques de gestion de fichiers, de communication réseau, etc.).

MODELE A CINQ COUCHES :

Ce modèle peut convenir à des applications complexes, fortement réparties :

COUCHE	DESCRIPTION
PRESENTATION	Interface homme-machine (affichage, interactions utilisateur), Clients légers (utilisation d'un navigateurs), Clients lourds (IHM gérant aussi les écrans)
CONTROLE	Gestion des sessions, habilitations, droits d'accès, gestion des erreurs. C'est dans cette couche que se place le modèle MVC.
SERVICES	Elle implémente la logique "métier" de l'application (Services "métier" enchaînements des rêgles métier). Ex : virement de compte à compte.
DOMAINE	Elle fournit des services qui gèrent les rêgles métier. Ex : Gestion d'un compte bancaire.
PERSISTANCE	Elle gère les données persistantes de l'application (En particulier, la Base de Données)

III.LA VUE EN ETAGES DE SERVICES (TIER) :

III.1.GENERALITES:

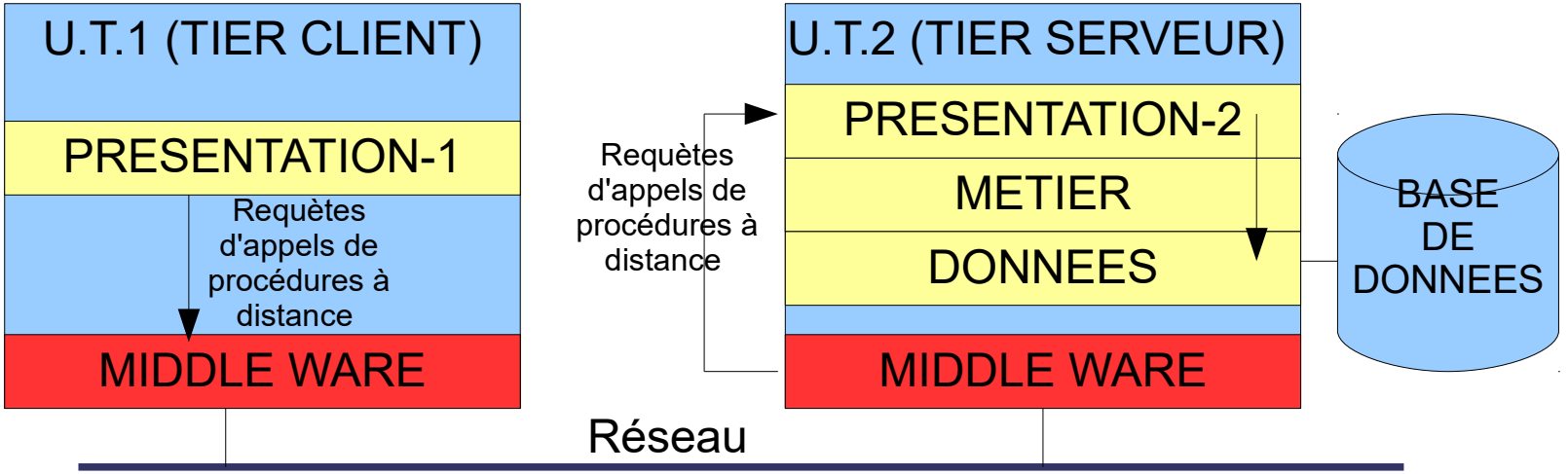
Lorsque l'application est destinée à s'exécuter sur une seule unité de traitement (architecture MAINFRAME), le point de vue LOGIQUE et la décomposition en couches qui en résulte suffisent pour définir l'architecture générale.

Dans le cas contraire (architecture REPARTIE), il faut prendre en compte la structure de cette architecture dans la conception. Cette opération consiste à REPARTIR LES TRAITEMENTS sur différentes UNITES DE TRAITEMENT (U.T.). Dans cette répartition, certaines couches logicielles peuvent se retrouver supportées par plusieurs U.T. Les composants d'un logiciel qui concourrent à réaliser un NIVEAU DE SERVICE donné constituent un TIER ou ETAGE.

la liaison entre les TIER est effectuée par des MIDDLE WARES. Ceux-ci peuvent utiliser :

- Soit des ECHANGES DE MESSAGES (Ex : échange de requètes HTTP (URL) ;
- Soit des APPELS DE PROCEDURES A DISTANCE (Ex : Un composant déclenche l'exécution d'une procédure dans la machine distante). C'est le cas de DCOM (Distributed Component Object Model) de Microsoft;
- Soit l'INVOCATION DE METHODES A DISTANCE (Ex : Un composant déclenche l'exécution d'une méthode d'un objet situé dans la machine distante). C'est le cas de CORBA(Common Object Request Broker Architecture), RMI (Remote Method Invocation de Java).

EXEMPLE : architecture en 3 couches répartie sur 2 U.T ou architecture 2 TIER (Client-Serveur) :



III.LA VUE EN ETAGES DE SERVICES (TIER) :

III.2.ARCHITECTURE 1-TIER:

L'architecture 1-TIER correspond au cas où l'application est supportée par une seule unité de traitement (U.T).

III.3.ARCHITECTURE 2-TIER:

L'architecture 2-TIER classique a été créée à l'origine pour soulager les MAIN FRAMES de la gestion des DONNEES. La répartition était :

- TIER n°1 : couches PRESENTATION et couches METIER
- TIER n° 2 : couche DONNEES.

C'est cette architecture qui est appelée CLIENT-SERVEUR.

Remarquons que le TIER CLIENT est en général représenté par plusieurs postes CLIENTS (Un serveur est fait pour gérer plusieurs clients)

L'architecture CLIENT-SERVEUR présente les défauts suivants :

- Le POSTE CLIENT supporte l'essentiel des traitements. Il a tendance à devenir très complexe ;
- La cohabitation de la couche PRESENTATION et des COUCHES METIER sur le CLIENT pose des problèmes et complique les évolutions ;
- Le dialogueCLIENT-SERVEUR consomme beaucoup de bande passante.

Cependant, l'architecture CLIENT-SERVEUR permet l'implantation d'IHM dits "riches" car plus complets que ne le permet un navigateur seul. On parle de CLIENT LOURDS ou CLIENTS RICHES.

REMARQUE :

L'exemple du chapitre précédent représente une architecture 2-TIER différente qui pourrait être utilisée dans le cas des SERVICES WEB :

- Le TIER CLIENT (Poste client web) ne supporte que le NAVIGATEUR (CLIENT LÉGER) ;
- Le TIER SERVEUR supporte le reste de la couche PRESENTATION et les deux autres couches.

Il s'agit donc également d'une architecture 2-TIER.

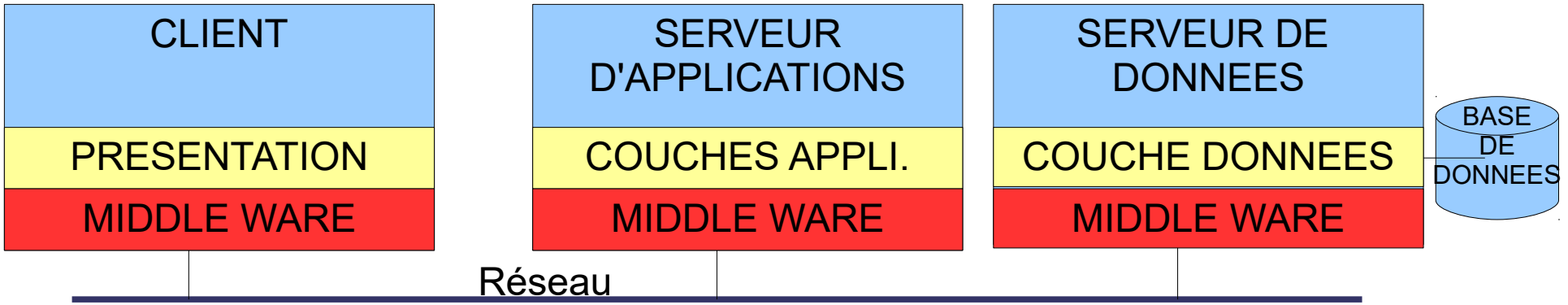
III.LA VUE EN ETAGES DE SERVICES (TIER) :

III.4.ARCHITECTURE 3-TIER:

Dans l'architecture 3-TIER, les trois types de traitements :

- PRESENTATION
- TRAITEMENTS APPLICATIFS (METIER)
- TRAITEMENT DES DONNEES

Sont répartis sur trois U.T. différentes. La couche PRESENTATION est donc séparée physiquement des couches APPLICATIVES :



L'architecture 3-TIER présente l'avantage d'être très EVOLUTIVE. En effet, il est possible de faire évoluer les logiciels de chacun des 3 TIER sans perturber les autres couche de l'application. Ainsi, une application bancaire peut présenter un IHM différent suivant le type de poste client qui est utilisé (PC, tablette, smartphone, distributeur bancaire).

III.5.ARCHITECTURE N-TIER (ou MULTI-TIER):

C'est une architecture distribuée sur de multiples serveurs. Cependant, les 3 NIVEAUX DE SERVICES de l'architecture 3-TIER restents valides. Simplement, un TIER peut être supporté par plusieurs serveurs : on peut ainsi avoir plusieurs serveurs de donées, plusieurs serveurs d'application, etc.

IV.LE PATTERN MVC

Un PATTERN DE CONCEPTION (ou MODELE DE CONCEPTION) désigne un schéma architectural GÉNÉRIQUE applicable à une problématique donnée. Actuellement, la littérature technique définit une trentaine de ces patterns.

Le PATTERN M.V.C (Modèle-Vue-Contrôle) est un modèle d'architecture qui est utilisé surtout pour développer des SITES WEB. Il est à la base de nombreux frameworks comme ZEND.

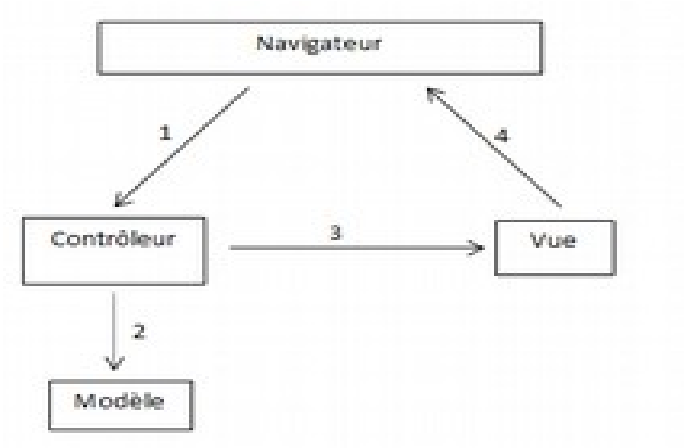


Schéma de fonctionnement MVC:

1- Action utilisateur via une requête HTTP (validation de formulaires ou activation de liens ;
2-Consultation et/ou mise à jour du MODÈLE
3-Le CONTRÔLEUR décide de la vue à afficher
4-La VUE renvoi le HTML au navigateur

- Le MODÈLE gère les DONNÉES et les ETATS de l'application web (par exemple le panier de courses sur un site d'e-commerce, ou la liste des produits à acheter,...). En général, ces données sont géréespar un ensemble de classes qui permettent d'accéder à une base de données. Le MODÈLE ne connaît ni la VUE, ni le CONTRÔLEUR. Sa seule finalité est d'être consulté ou modifié par ces derniers.
- La VUE détermine ce qu'on voit sur l'écran du navigateur web. Elle est constituée composants codes en HTML/CSS. Son butest de présenter les données issues du modèle.
- Le CONTRÔLEUR gére les interactions entre la VUE et le MODÈLE. Il traite les interactions avec l'utilisateur et détermine les traitements qui doivent être activé pour réaliser ces intéractions. Pour cela, il utilise les données du MODÈLE.

ATTENTION: *le PATTERN MVC n'a rien à voir avec une architecture 3-tiers: en effet, dans une architecture N-Tiers, il n'est possible de dialoguer qu'avec les tiers* ***adjacents****.*