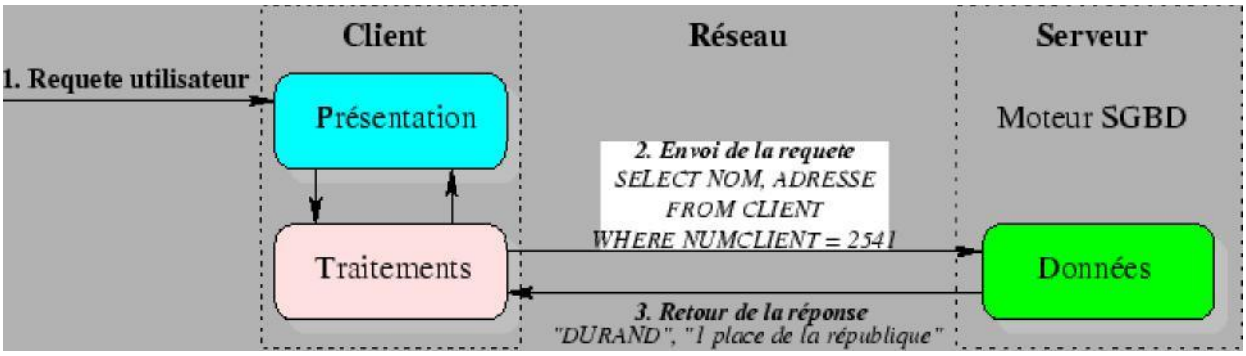
LES DIFFÉRENTES ARCHITECTURES CLIENT/SERVEUR

L'ARCHITECTURE 2 TIERS

Dans une architecture deux tiers, encore appelée client-serveur de première génération ou client-serveur de données, le poste client se contente de déléguer la gestion des données à un service spécialisé. Le cas typique de cette architecture est une application de gestion fonctionnant sous Windows ou Linux et exploitant un SGBD centralisé.

Ce type d'application permet de tirer partie de la puissance des ordinateurs déployés en réseau pour fournir à l'utilisateur une interface riche, tout en garantissant la cohérence des données, qui restent gérées de façon centralisée.

La gestion des données est prise en charge par un SGBD centralisé, s'exécutant le plus souvent sur un serveur dédié. Ce dernier est interrogé en utilisant un langage de requête qui, le plus souvent, est SQL. Le dialogue entre client et serveur se résume donc à l'envoi de requêtes et au retour des données correspondant aux requêtes.



Cet échange de messages transite à travers le réseau reliant les deux machines. Il met en œuvre des mécanismes relativement complexes qui sont, en général, pris en charge par un middleware.

Remarque : *Cette architecture est celle que nous avons mise en place au cours du TP 10.*

L'expérience a démontré qu'il était coûteux et contraignant de vouloir faire porter l'ensemble des traitements applicatifs par le poste client. On en arrive aujourd'hui à dire que ce que l'on appelle le client lourd, a un certain nombre d'inconvénients :

- on ne peut pas soulager la charge du poste client, qui supporte la grande majorité des traitements applicatifs,
- le poste client est fortement sollicité, il devient de plus en plus complexe et doit être mis à jour régulièrement pour répondre aux besoins des utilisateurs,
- les applications se prêtent assez mal aux fortes montées en charge car il est difficile de modifier l'architecture initiale,
- la relation étroite qui existe entre le programme client et l'organisation de la partie serveur complique les évolutions de cette dernière,
- ce type d'architecture est grandement rigidifié par les coûts et la complexité de sa maintenance.

Malgré tout, l'architecture deux tiers présente de nombreux avantages qui lui permettent de présenter un bilan globalement positif :

- elle permet l'utilisation d'une interface utilisateur riche,
- elle a permis l'appropriation des applications par l'utilisateur,
- elle a introduit la notion d'interopérabilité.

Pour résoudre les limitations du client-serveur deux tiers tout en conservant ses avantages, on a cherché une architecture plus évoluée, facilitant les forts déploiements à moindre coût. La réponse est apportée par les architectures distribuées.

L'ARCHITECTURE 3 TIERS

Les limites de l'architecture deux tiers proviennent en grande partie de la nature du client utilisé :

- le frontal est complexe et non standard (même s'il s'agit presque toujours d'un PC sous Windows),
- le middleware entre client et serveur n'est pas standard (dépend de la plate-forme, du SGBD ...).

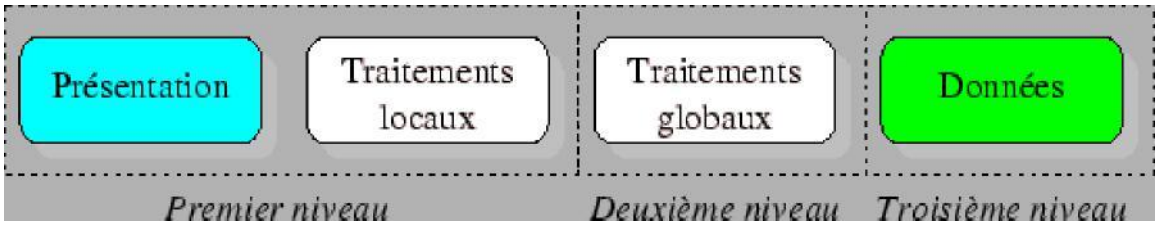
La solution résiderait donc dans l'utilisation d'un poste client simple communicant avec le serveur par le biais d'un protocole standard.

Dans ce but, l'architecture trois tiers applique les principes suivants :

- les données sont toujours gérées de façon centralisée,
- la présentation est toujours prise en charge par le poste client,
- la logique applicative est prise en charge par un serveur intermédiaire.

Cette architecture trois tiers, également appelée client-serveur de deuxième génération ou client-serveur distribué sépare l'application en 3 niveaux de services distincts, conformes au principe précédent :

- premier niveau : l'affichage et les traitements locaux (contrôles de saisie, mise en forme de données...) sont pris en charge par le poste client,
- deuxième niveau : les traitements applicatifs globaux sont pris en charge par le service applicatif,
- troisième niveau : les services de base de données sont pris en charge par un SGBD.



Tous ces niveaux étant indépendants, ils peuvent être implantés sur des machines différentes, de ce fait:

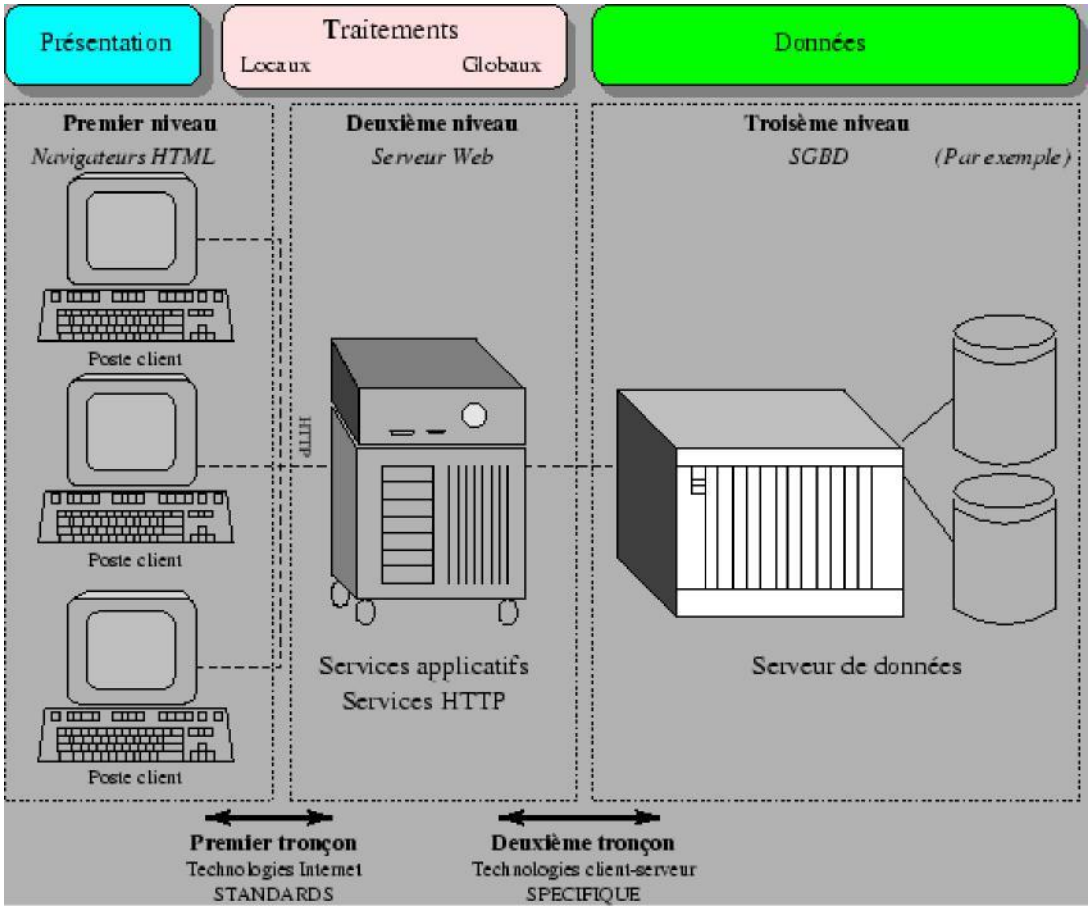
- le poste client ne supporte plus l'ensemble des traitements, il est moins sollicité et peut être moins évolué, donc moins coûteux,
- les ressources présentes sur le réseau sont mieux exploitées, puisque les traitements applicatifs peuvent être partagés ou regroupés (le serveur d'application peut s'exécuter sur la même machine que le SGBD),
- la fiabilité et les performances de certains traitements se trouvent améliorées par leur centralisation,
- il est relativement simple de faire face à une forte montée en charge, en renforçant le service applicatif.

Dans l'architecture trois tiers, le poste client est communément appelé client léger ou Thin Client, par opposition au client lourd des architectures deux tiers. Il ne prend en charge que la présentation de l'application avec, éventuellement, une partie de logique applicative permettant une vérification immédiate de la saisie et la mise en forme des données.

Le serveur de traitement constitue la pierre angulaire de l'architecture et se trouve souvent fortement sollicité. Dans ce type d'architecture, il est difficile de répartir la charge entre client et serveur. On se retrouve confronté aux épineux problèmes de dimensionnement serveur et de gestion de la montée en charge rappelant l'époque des mainframes.

Les contraintes semblent inversées par rapport à celles rencontrées avec les architectures deux tiers : le client est soulagé, mais le serveur est fortement sollicité.

Remarque : Nous avons mis en place cette architecture au cours du TP 11



L'ARCHITECTURE N-TIERS

L'architecture n-tiers a été pensée pour pallier aux limites des architectures trois tiers et concevoir des applications puissantes et simples à maintenir. Ce type d'architecture permet de distribuer plus librement la logique applicative, ce qui facilite la répartition de la charge entre tous les niveaux.

Cette évolution des architectures trois tiers met en œuvre une approche objet pour offrir une plus grande souplesse d'implémentation et faciliter la réutilisation des développements.

Cette architecture est basée sur l'utilisation de composants "métier'', spécialisés et indépendants, introduits par les concepts orientés objets (langages de programmation et middleware). Elle permet de tirer pleinement partie de la notion de composants métiers réutilisables.

Ces composants rendent un service si possible générique et clairement identifié. Ils sont capables de communiquer entre eux et peuvent donc coopérer en étant implantés sur des machines distinctes.