Sommaire

1 Introduction

Ces vingt dernières années ont vues une évolution majeure des systèmes d'information, à savoir le passage d'une architecture centralisée à travers de grosses machines (des Mainframe) vers une architecture distribuée basée sur l'utilisation de serveurs et de postes clients grâce à l'utilisation des PC et des réseaux.

Cette évolution a été possible essentiellement grâce à 2 facteurs qui sont :

- la baisse des prix de l'informatique personnelle
- le développement des réseaux.

2 L'architecture client-serveur

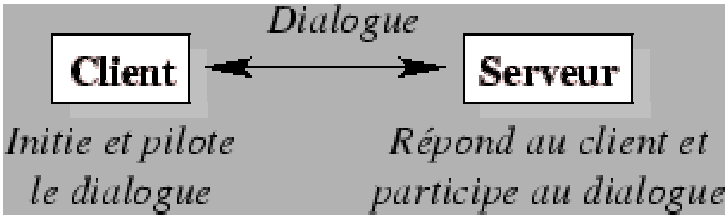
2.1 Définition

L'architecture client-serveur est un modèle de fonctionnement logiciel qui peut se réaliser sur tout type d'architecture matérielle (petites ou grosses machines), à partir du moment ou ces architectures peuvent être interconnectées.

On parle de fonctionnement logiciel dans la mesure où cette architecture est basée sur l'utilisation de deux types de logiciels, à savoir un logiciel serveur et un logiciel client s'exécutant normalement sur 2 machines différentes. L'élément important dans cette architecture est l'utilisation de mécanismes de communication entre les 2 applications.

Le dialogue entre les applications peut se résumer par :

- Le client demande un service au serveur
- Le serveur réalise ce service et renvoie le résultat au client



Un des principes fondamental est que le serveur réalise un traitement pour le client.

2.2 Les principes généraux

Il n'y a pas véritablement de définition exhaustive de la notion de client-serveur, néanmoins des principes régissent ce que l'on entend par client-serveur :

- Service.
  Le serveur est fournisseur de services. Le client est consommateur de services.
- Protocole.
  C'est toujours le client qui déclenche la demande de service. Le serveur attend passivement les requêtes des clients.
- Partage des ressources.
  Un serveur traite plusieurs clients en même temps et contrôle leurs accès aux ressources.
- Localisation.
  Le logiciel client-serveur masque aux clients la localisation du serveur.
- Hétérogénéité.
  Le logiciel client-serveur est indépendant des plate-formes matérielles et logicielles.
- Redimensionnement.
  Il est possible d'ajouter et de retirer des stations clientes. Il est possible de faire évoluer les serveurs.
- Intégrité.
  Les données du serveur sont gérées sur le serveur de façon centralisée. Les clients restent individuels et indépendants.
- Souplesse et adaptabilité.
  On peut modifier le module serveur sans toucher au module client. La réciproque est vraie. Si une station est remplacée par un modèle plus récent, on modifie le module client (en améliorant l'interface, par exemple) sans modifier le module serveur.

2.3 La répartition des tâches

Dans l'architecture client-serveur, une application est constituée de trois parties :

- l'interface utilisateur
- la logique des traitements
- la gestion des données.

Le client n'exécute que l'interface utilisateur (souvent un interfaces graphique) ainsi que la logique des traitements (formuler la requête), laissant au serveur de bases de données la gestion complète des manipulations de données.

La liaison entre le client et le serveur correspond à tout un ensemble complexe de logiciels appelé middleware qui se charge de toutes les communications entre les processus.

2.4 Les différents modèles de client-serveur

En fait, les différences sont essentiellement liées aux services qui sont assurés par le serveur. On distingue couramment :

2.4.1 le client-serveur de donnée.

Dans ce cas, le serveur assure des tâches de gestion, stockage et de traitement de données. C'est le cas le plus connu de client-serveur est qui est utilisé par tous les grands SGBD :

La base de donnée avec tous ses outils (maintenance, sauvegarde …) est installée sur un poste serveur.

Sur les clients, un logiciel d'accés est installé permettant d'accéder à la base de donnée du serveur.

Tous les traitements sur les données sont effectués sur le serveur qui renvoie les informations demandées (souvent à travers une requête SQL) par le client

2.4.2 Client-serveur de présentation

Dans ce cas la présentation des pages affichées par le client est intégralement prise en charge par le serveur. Cette organisation présente l'inconvénient de générer un fort trafic réseau.

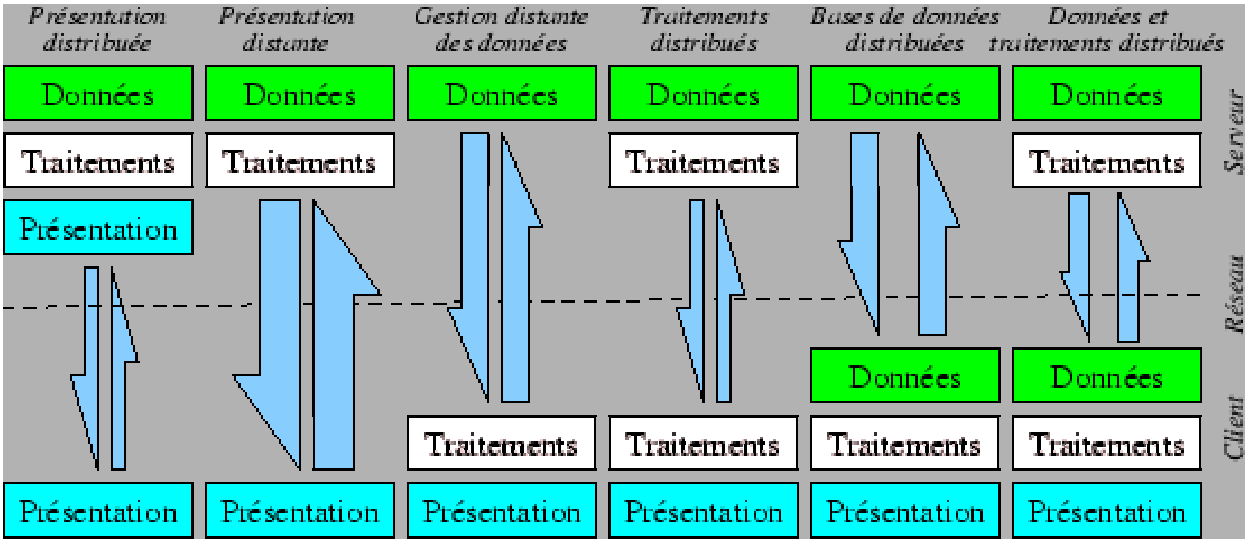
2.4.3 le client-serveur de traitement

Dans ce cas, le serveur effectue des traitements à la demande du client. Il peut s'agir de traitement particulier sur des données, de vérification de formulaires de saisie, de traitements d'alarmes …

Ces traitements peuvent être réalisés par des programmes installé sur des serveurs mais également intégrés dans des bases de données (triggers, procédures stockées), dans ce cas, la partie donnée et traitement sont intégrées.

2.4.4 Une synthèse des différents cas

Cette synthèse s'illustre par un schéma du Gartner Group qui représente les différents modèles ainsi que la répartition des tâches entre serveur et client.



Sur ce schéma, le trait horizontal représente le réseau et les flèches entre client et serveur, le trafic réseau généré par la conversation entre client et serveur.
Nous verrons par la suite que la vision du Gartner Group, en ne prenant en compte qu'un découpage en deux niveaux, est quelque peu limitative.
Le Gartner Group distingue les types de client-serveur suivants, en fonction du type de service déporté du cœur de l'application :

1. **Présentation distribuée** : Correspond à l'habillage ``graphique'' de l'affichage en mode caractères d'applications fonctionnant sur site central. Cette solution est aussi appelée *revamping*. La classification ``client-serveur'' du revamping est souvent jugée abusive, du fait que l'intégralité des traitements originaux est conservée et que le poste client conserve une position d'esclave par rapport au serveur.
2. **Présentation distante** : Encore appelée client-serveur de présentation. L'ensemble des traitements est exécuté par le serveur, le client ne prend en charge que l'affichage. Ce type d'application présentait jusqu'à présent l'inconvénient de générer un fort trafic réseau et de ne permettre aucune répartition de la charge entre client et serveur.

   S'il n'était que rarement retenu dans sa forme primitive, il connaît aujourd'hui un très fort regain d'intérêt avec l'exploitation des standards Internet.

3. **Gestion distante des données** : Correspond au client-serveur de données, sans doute le type de client-serveur le plus répandu. L'application fonctionne dans sa totalité sur le client, la gestion des données et le contrôle de leur intégrité sont assurés par un SGBD centralisé.

   Cette architecture, de part sa souplesse, s'adapte très bien aux applications de type info centre, interrogeant la base de façon ponctuelle. Il génère toutefois un trafic réseau assez important et ne soulage pas énormément le poste client, qui réalise encore la grande majorité des traitements.

4. **Traitement distribué** : Correspond au client-serveur de traitements. Le découpage de l'application se fait ici au plus près de son noyau et les traitements sont distribués entre le client et le(s) serveur(s).

Le client-serveur de traitements s'appuie, soit un mécanisme d'appel de procédure distante, soit sur la notion de procédure stockée proposée par les principaux SGBD du marché.
Cette architecture permet d'optimiser la répartition de la charge de traitement entre machines et limite le trafic réseau. Par contre il n'offre pas la même souplesse que le client-serveur de données puisque les traitements doivent être connus du serveur à l'avance.

5. **Bases de données distribuées** : Il s'agit d'une variante du client-serveur de données dans laquelle une partie de données est prise en charge par le client. Ce modèle est intéressant si l'application doit gérer de gros volumes de données, si l'on souhaite disposer de temps d'accès très rapides sur certaines données ou pour répondre à de fortes contraintes de confidentialité.

   Ce modèle est aussi puissant que complexe à mettre en oeuvre.

6. **Données et traitements distribués**. Ce modèle est très puissant et tire partie de la notion de composants réutilisables et distribuables pour répartir au mieux la charge entre client et serveur.

   C'est, bien entendu, l'architecture la plus complexe à mettre en oeuvre.

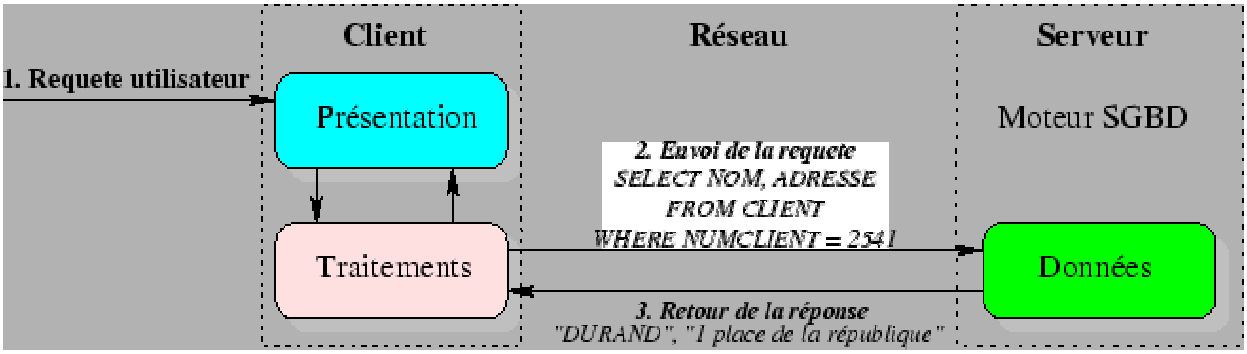
2.5 Les différentes architectures

2.5.1 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 oeuvre des mécanismes relativement complexes qui sont, en général, pris en charge par un middleware.

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 ce que l'on appelle le client lourd, avec 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.

2.5.2 L'architecture 3 tiers

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

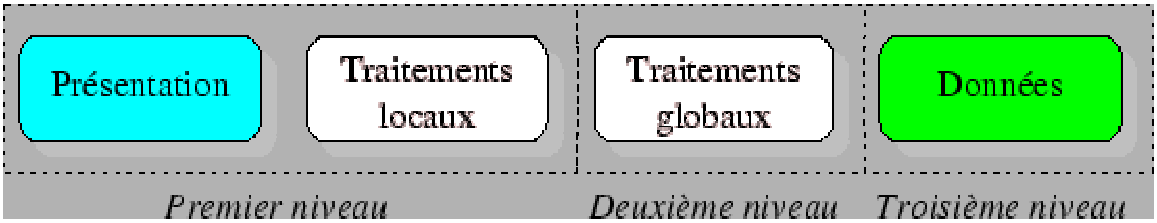
- le frontal est complexe et non standard (même s'il s'agit presque toujours d'un PC sous Windows),
- le middleware entre client et serveur n'est pas standard (dépend de la plate-forme, du SGBD …).

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.
De plus, les solutions mises en oeuvre sont relativement complexes à maintenir et la gestion des sessions est compliquée.

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

2.5.3 L'architecture n-tiers

L'architecture n-tiers a été pensée pour pallier aux limitations 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 oeuvre une approche objet pour offrir une plus grande souplesse d'implémentation et faciliter la réutilisation des développements.
Théoriquement, ce type d'architecture supprime tous les inconvénients des architectures précédentes :

- elle permet l'utilisation d'interfaces utilisateurs riches,
- elle sépare nettement tous les niveaux de l'application,
- elle offre de grandes capacités d'extension,
- elle facilite la gestion des sessions.

L'appellation ``n-tiers'' pourrait faire penser que cette architecture met en oeuvre un nombre indéterminé de niveaux de service, alors que ces derniers sont au maximum trois (les trois niveaux d'une application informatique). En fait, l'architecture n-tiers qualifie la distribution d'application entre de multiples services et non la multiplication des niveaux de service.

Cette distribution est facilitée par 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.

La distribution des services applicatifs facilite aussi l'intégration de traitements existants dans les nouvelles applications. On peut ainsi envisager de connecter un programme de prise de commande existant sur le site central de l'entreprise à une application distribuée en utilisant un middleware adapté.

Ces nouveaux concepts sont basés sur la programmation objet ainsi que sur des communications standards entre application. Ainsi est né le concept de Middleware objet.

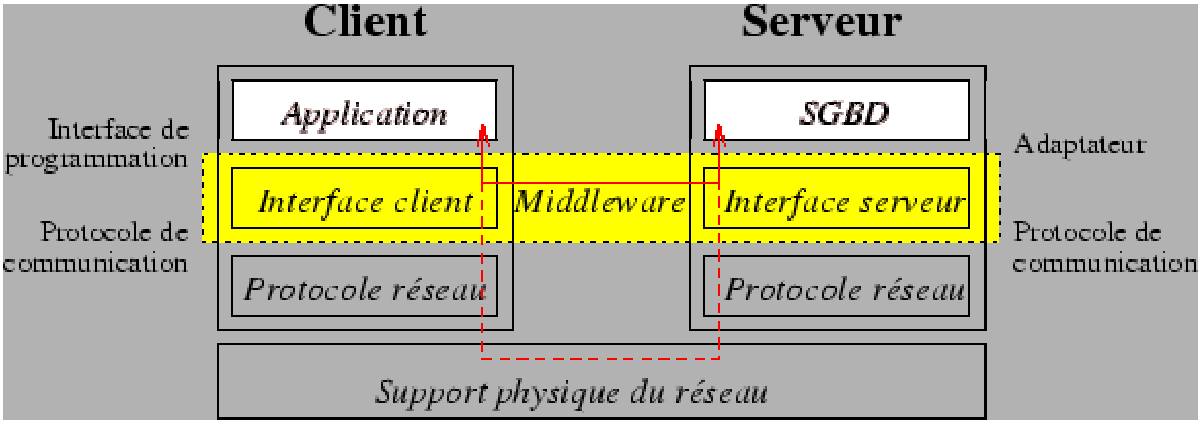
2.6 Les middleware

2.6.1 Présentation

On appelle middleware (ou logiciel médiateur en français), littéralement ``élément du milieu'', l'ensemble des couches réseau et services logiciel qui permettent le dialogue entre les différents composants d'une application répartie. Ce dialogue se base sur un protocole applicatif commun, défini par l'API du middleware.

Le Gartner Group définit le middleware comme une interface de communication universelle entre processus. Il représente véritablement la clef de voûte de toute application client-serveur.

L'objectif principal du middleware est d'unifier, pour les applications, l'accès et la manipulation de l'ensemble des services disponibles sur le réseau, afin de rendre l'utilisation de ces derniers presque transparente.



2.6.2 Les services des middleware

Un middleware est susceptible de rendre les services suivants :

- **Conversion** : Service utilisé pour la communication entre machines mettant en oeuvre des formats de données différents
- **Adressage** : Permet d'identifier la machine serveur sur laquelle est localisé le service demandé afin d'en déduire le chemin d'accès. Dans la mesure du possible
- **Sécurité** : Permet de garantir la confidentialité et la sécurité des données à l'aide de mécanismes d'authentification et de cryptage des informations.
- **Communication** : Permet la transmission des messages entre les deux systèmes sans altération. Ce service doit gérer la connexion au serveur, la préparation de l'exécution des requêtes, la récupération des résultats et la dé-connexion de l'utilisateur.

Le middleware masque la complexité des échanges inter-applications et permet ainsi d'élever le niveau des API utilisées par les programmes. Sans ce mécanisme, la programmation d'une application client-serveur serait complexe et difficilement évolutive.

2.6.3 Exemples de Middleware :

- **SQL*Net** : Interface propriétaire permettant de faire dialoguer une application cliente avec une base de données Oracle. Ce dialogue peut aussi bien être le passage de requêtes SQL que l'appel de procédures stockées.
- **ODBC** : (Object Data Base Connexion) Interface standardisée isolant le client du serveur de données. C'est l'implémentation par Microsoft d'un standard défini par le SQL Access Group. Elle se compose d'un gestionnaire de driver standardisé, d'une API s'interfaçant avec l'application cliente (sous Ms Windows) et d'un driver correspondant au SGBD utilisé.

- **DCE** : (Distributions Computing environment) Permet l'appel à des procédures distantes depuis une application. Correspond à RPC (Remote Procedure Call) qui permet d'éxécuter des procédures distantes.

Le choix d'un middleware est déterminant en matière d'architecture, il joue un grand rôle dans la structuration du système d'information.

Pour certaines applications devant accéder à des services hétérogènes, il est parfois nécessaire de combiner plusieurs middlewares. On en vient à la notion de client lourd.

2.6.4 Les Middleware objet

Pour permettre la répartition d'objets entre machines et l'intégration des systèmes non objets, il doit être possible d'instaurer une communication entre tous ces éléments. Ainsi est né le concept de middleware objet qui a donné naissance à plusieurs spécifications, dont l'architecture CORBA 5common Object Request Broker Architecture) préconisée par l'OMG (Object Management Group) et DCOM développée par Microsoft.

Ces middlewares sont constitués d'une série de mécanismes permettant à un ensemble de programmes d'inter opérer de façon transparente. Les services offerts par les applications serveurs sont présentés aux clients sous la forme d'objets. La localisation et les mécanismes mis en oeuvre pour cette interaction sont cachés par le middleware.

La communication entre objets gomme la différence entre ce qui est local ou distant. Les appels de méthodes d'objet à objet sont traités par un mécanisme se chargeant d'aiguiller les messages vers les objets (locaux ou distants).

3 Le cas de l'Internet

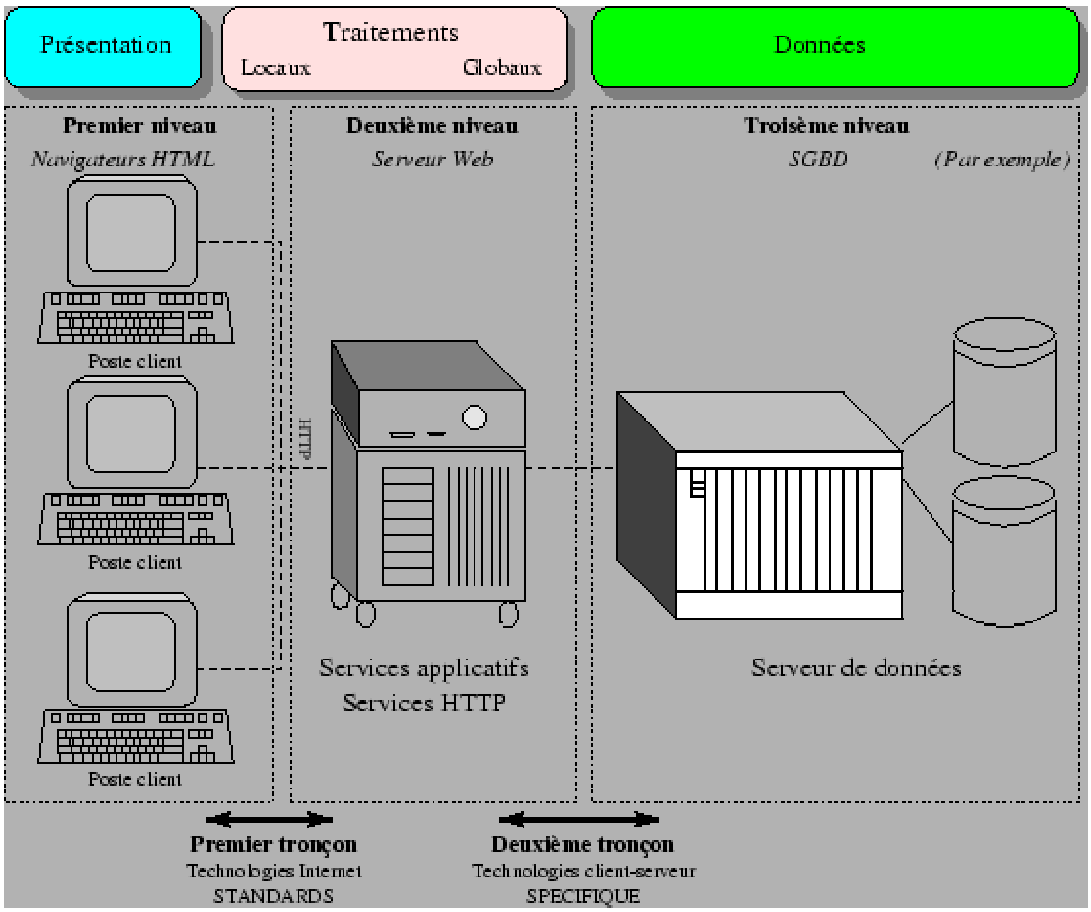
Dans le cadre d'un Intranet ou d'un Extranet, le poste client prend la forme d'un simple navigateur Web, le service applicatif est assuré par un serveur HTTP et la communication avec le SGBD met en oeuvre les mécanismes bien connus des applications client-serveur de la première génération.

3.1 Répartition des tâches

Ce type d'architecture fait une distinction nette entre deux tronçons de communication indépendants et délimités par le serveur HTTP :

- le premier tronçon relie le poste client au serveur Web pour permettre l'interaction avec l'utilisateur et la visualisation des résultats. Ce premier tronçon n'est composé que de standards (principalement HTML et HTTP) et est basé sur un simple navigateur Web. Le serveur Web tient le rôle de ``frontal HTTP'',

- le deuxième tronçon permet la collecte des données. Les mécanismes utilisés sont comparables à ceux mis en oeuvre pour une application deux tiers. Ils ne franchissent jamais la façade HTTP et, de ce fait, peuvent évoluer sans influence sur la configuration des postes clients.



3.2 Le client universel

Ce type d'architecture est définie par certain comme le client-serveur universel dans la mesure où il s'appuie sur des standards existant sur toutes les plate-formes.

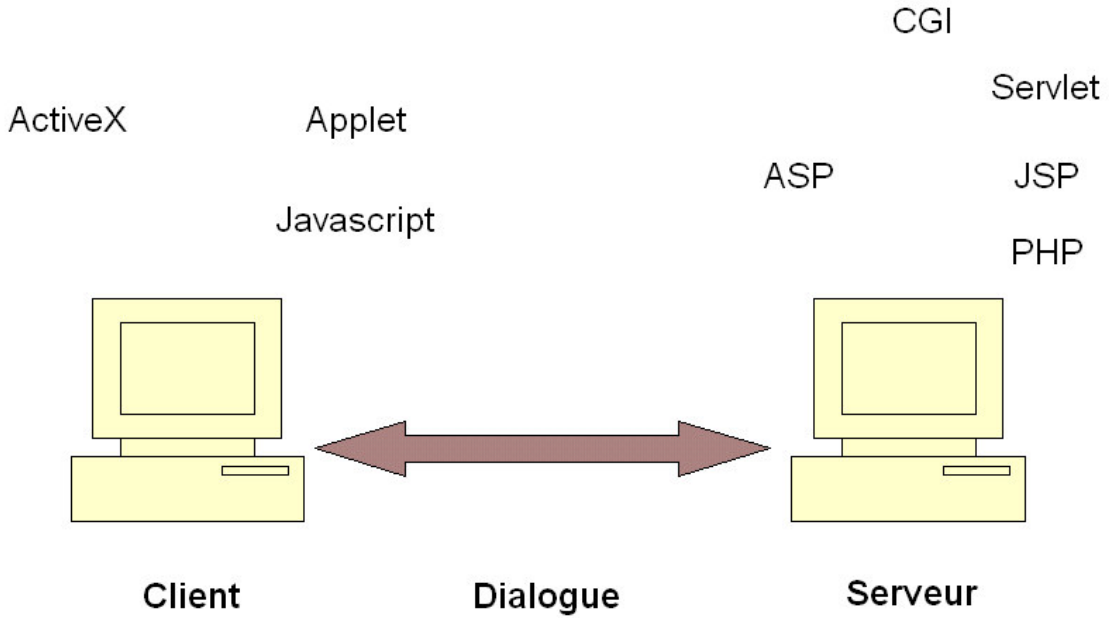
De plus, dans la même approche, on peut dire au vu de l'explosion des architectures Internet que le navigateur Web est le client universel, puisque de plus en plus utilisé pour tout type de développement, ou d'interface.
L'avantage de ce client universel est qu'il est aujourd'hui présent sur tout type de machine :

- Postes de bureau
- Portables
- Pockets PC
- Téléphones portables
- …

De plus, tous les constructeurs de produits embarqués embarquent dans leurs applicatifs un serveur http qui permet d'administrer facilement à distance leurs machines.

3.3 Les technologies coté client ou serveur

Pour revenir aux architectures 3 tiers de l'Internet, cette solution peut s'appuyer sur toutes les nouvelles technologies, et permet de répartir en fonction des besoins les traitements coté client ou coté serveur.



- Les technologies coté client font appel à :
  - du javascript
  - des applets JAVA
  - des composants ACTIVEX (Microsoft)

- Coté serveur, toutes les techniques permettant de faire du Web dynamique à savoir :
  - des scripts CGI (Perl, C, C++
  - des servlets écrit en JAVA : JSP (Java Server Pages)
  - des traitements en ASP (Active Server Pages) - technologie Microsoft
  - du code PHP

3.4 Le futur

Le futur appelé par certains Web 2.0 s'oriente néanmoins vers des clients plus riche afin de pouvoir bénéficier d'interfaces plus conviviaux, tout en gardant le coté universel du poste client. Un exemple en est donné aujourd'hui à travers certains clients de messagerie (Yahoo, MSN).

Alors, si vous voulez rester à la page, suivez de prés ces nouvelles technologies…

Pour en savoir plus : http://www.framasoft.net/article3991.html