L’architecture client serveur

PLAN

Bibliographie

- Client serveur : guide de survie (ORF0ALI, HARKEY, EDWARDS : International Thomson Publishing)
- Client serveur (G. et O. GARDARIN : Eyrolles)
- Intranet client-serveur universel (Alain Lefebvre : Eyrolles)
- Décision micro & réseaux (N° 380 avril mai 1999)

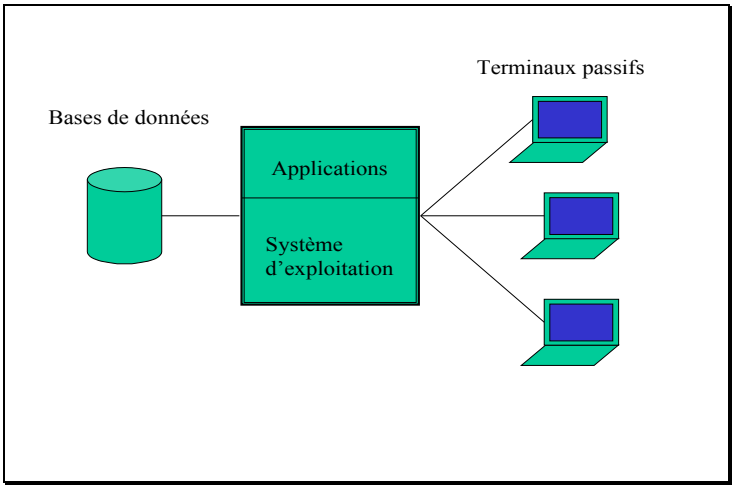
I. Introduction

En quelques années, l'architecture informatique a évolué des grands systèmes centralisés vers une architecture client serveur, dernier cri en matière de plate-forme ouverte.

A. Caractéristiques des grands systèmes

- Machine énorme comportant un système d'exploitation unique.
- Réseau frontal tentaculaire raccordant tous les terminaux.
- Stockage de masse par disques et bandes magnétiques.
- Organes de surveillance et de maintenance.
- Méthodes de développement et AGL propriétaires.

Cet ensemble était fourni par un seul constructeur qui constituait l'interlocuteur unique. Le métier d'informaticien était relativement confortable : les évolutions étaient simples et planifiées. Cette architecture assurait emploi et plan de carrière.



Exemple d'architecture centralisée

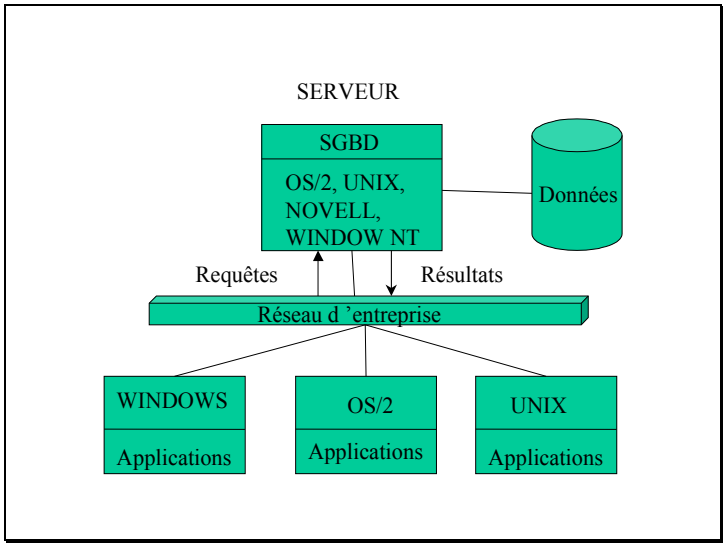
B. Caractéristiques de l'architecture client serveur

Cette architecture ce distingue par la faculté de pouvoir mélanger et apparier des composants à tous les niveaux. L'élaboration d'une solution informatique est entièrement 'à la carte'. Aussi, l'architecte est-il confronté à des choix très complexes.

- quelle plate forme serveur ?
- quelle plate forme client ?
- quels protocoles réseau ?
- quel serveur de bases de données ?
- quel middleware ?
- quels outils de gestion du système ?
- quelle technologie utiliser pour bâtir des applications ?
  - serveur de bases de données,
  - moniteur transactionnel ?

- groupware ?
- objets répartis ?
- internet, intranet ?

Contrairement aux solutions entièrement propriétaires, l'architecte est seul (ou presque) pour assembler le tout et le faire fonctionner : il est l'intégrateur du système.



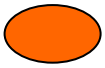
Architecture moderne des années 80

II. Qu'est-ce que le modèle client serveur ?

A. La classification du Gartner Group

L'architecture client serveur est un mode de dialogue entre deux processus : le client demande l'exécution de services au serveur qui retourne les réponses. Le serveur doit bien sûr être en mesure de traiter les requêtes de plusieurs clients.

La classification du Gartner Group dépend de la répartition de trois fonctions :

- graphiques : affichage et saisie des données 
- gestion des données : accès aux fichiers ou aux bases de données. 
- Exécution du code applicatif. 

1. Le client serveur de présentation

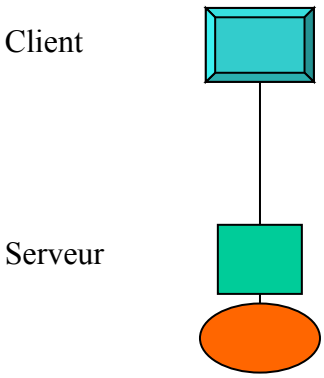
Ce type d'architecture permet d'assurer une meilleure qualité du dialogue homme machine.

Définition :

Type d'architecture client serveur dans lequel un processus exécute seulement les fonctions de dialogue avec l'utilisateur, l'autre gère les données et exécute le code applicatif.

Remarque : la définition correspond exactement à la description du schéma, sauf que l'on ne précise pas qui est le client, qui est le serveur. Toute l'ambiguïté réside dans le fait que l'on peut appliquer le raisonnement client serveur aux machines ou au processus.

Le processus qui exécute le code applicatif s'adresse à un autre pour les entrées / sorties graphiques. Selon la définition, il est le processus client mais est hébergé par la machine serveur. On parle alors de dialogue client serveur inversé.



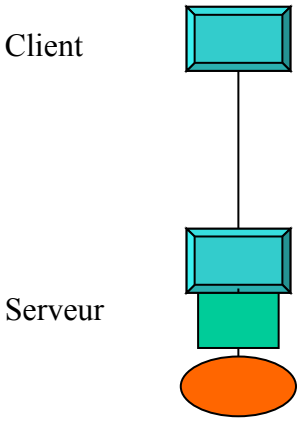
2. Le rhabillage ou revamping

Ce type d'architecture caractérise la transformation des anciennes applications centralisées : une machine cliente intelligente capable d'exécuter une interface graphique sophistiquée vient remplacer un terminal mode caractère. L'objectif est bien entendu de minimiser les modifications de l'application initiale

Définition :

Type d'architecture client serveur dans lequel un processus exécute seulement la fonction de dialogue sophistiqué avec l'utilisateur. L'autre processus gère les données, exécute le code applicatif et assure le dialogue simplifié avec l'utilisateur.

Là encore, le dialogue client serveur est qualifié inversé.



3. Client serveur de données

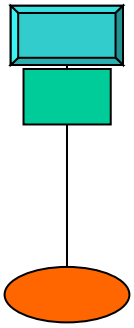
C'est l'architecture la plus répandue. Exemple : un PC accède à des données partagées gérées par un serveur SQL.

Définition : type d'architecture dans lequel un programme d'application, contrôlé par une interface de présentation sur une machine cliente, accède à des données sur une machine serveur par des requêtes.

Cette architecture est qualifiée de première génération.

Client

Serveur



4. Client serveur de procédures

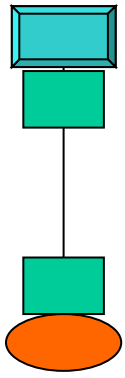
C'est une évolution de l'architecture précédente. La base de données intègre des procédures stockées : procédures applicatives recevant des paramètres d'entrée et retournant des paramètres de sortie.

Définition : type d'architecture client serveur dans lequel un programme applicatif contrôlé par une interface de présentation sur une machine cliente, sous-traite l'exécution de procédures applicatives à une machine serveur.

Remarque : le serveur de procédures inclue un serveur de données basé sur SQL. Le client peut donc accéder aux données directement via SQL ou appeler des procédures qui manipulent les données. Le client serveur de procédures est en fait une architecture client serveur de données et procédures.

Client

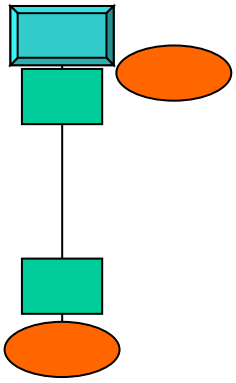
Serveur



5. Système réparti

L'architecture système réparti est une architecture client serveur de données et procédures dans laquelle le client peut accéder à des données qui sont réparties sur plusieurs serveurs mais peuvent également être gérées en local.

Client



Serveur

B. Caractéristiques des systèmes client serveur.

Ce paragraphe présente les éléments qui caractérisent une architecture client serveur.

1. Service

Le modèle client serveur est une relation entre des processus qui tournent sur des machines séparées. Le serveur est un fournisseur de services. Le client est un consommateur de services.

2. Partage de ressources

Un serveur traite plusieurs clients et contrôle leurs accès aux ressources.

3. Protocole asymétrique

Conséquence du partage de ressources, le protocole de communication est asymétrique : le client déclenche le dialogue ; le serveur attend les requêtes des clients.

4. Transparence de la localisation.

L'architecture client serveur doit masquer au client la localisation du serveur (que le service soit sur la même machine ou accessible par le réseau).

Transparence par rapport aux systèmes d'exploitation et aux plates-formes matérielles.

Idéalement, le logiciel client serveur doit être indépendant de ces deux éléments.

5. Messages

Les messages sont les moyens d'échanges entre client et serveur.

6. Encapsulation des services.

Un client demande un service. Le serveur décide de la façon de le rendre une mise à niveau du logiciel serveur doit être sans conséquence pour le client tant que l'interface message est identique.

7. Evolution

Une architecture client serveur doit pouvoir évoluer horizontalement (évolution du nombre de clients) et verticalement (évolution du nombre et des caractéristiques des serveurs).

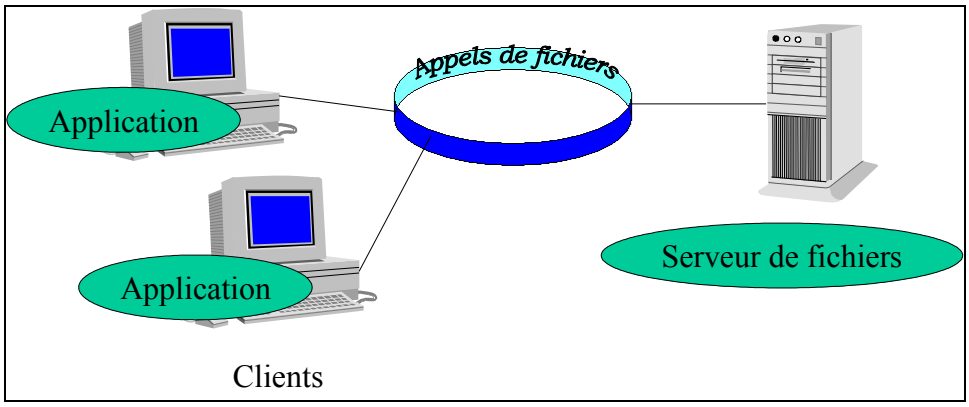
L'architecture client serveur correspond à la répartition de l'intelligence sur le réseau.

III.Les différentes catégories de client serveur

Les différentes architectures client serveur peuvent être classées en fonction du service rendu aux utilisateurs.

A. Serveur de fichier

Le client (généralement un PC) requiert des enregistrements de fichiers en émettant des requêtes sur le réseau en direction d'un serveur de fichier.



Client / serveur avec serveur de fichiers

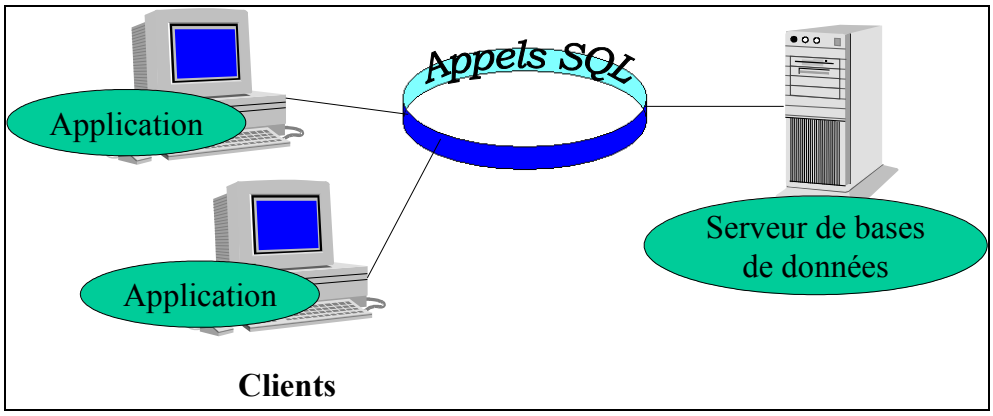
Caractéristiques.

- Très utilisé à ce jour (partage de fichiers sur le réseau).
- Forme primitive de service de données.
- Nombreux échanges de messages sur le réseau pour obtenir le résultat.
- Indispensable pour les banques de documents, d'images etc.

B. Serveur de bases de données

Le client émet des requêtes SQL sous forme de message.

Le serveur renvoie le résultat de chaque requête.



Client / serveur avec serveur de bases de données

Caractéristiques.

- Meilleure répartition de la puissance : le serveur utilise sa capacité de traitement (SGBD) pour sélectionner les réponses.
- Nécessité d'écrire du code pour l'application cliente.
- C'est la base des systèmes d'aide à la décision.

C. Serveur de transactions

Le client invoque des procédures distantes résidant sur le serveur qui comporte un moteur de bases de données.

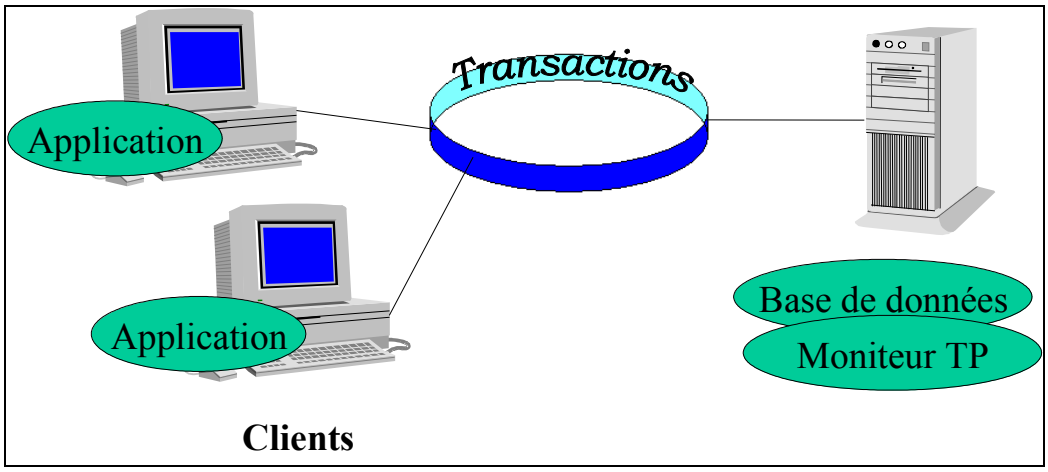
Chaque procédure correspond à un ensemble de requêtes SQL appelé transaction.

L'échange consiste en un message requête / réponse, .contrairement au cas précédent ou il y a un message par instruction SQL.

Le code applicatif est réparti sur le client et le serveur. Coté serveur, il porte le nom de 'traitement de transaction en ligne OLTP (On Line Transaction Processing).

Il existe deux formes de transactionnel :

- transactionnel léger : les procédures sont fournies par l'éditeur du SGBD.
- Transactionnel lourd : moniteurs transactionnels fournis par l'éditeur d'applications OLTP.



Client / serveur avec serveur de transactions

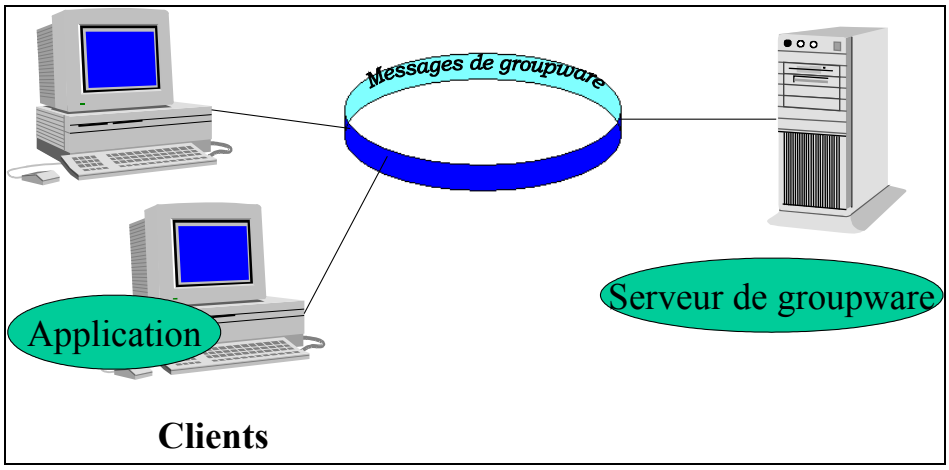
D. Serveur de groupware

Le groupware est fondé sur 5 technologies de base :

- gestion de documents multimédia
- workflow (Annexes page 37)
- courrier électronique
- gestion de conférences
- planification de réunions

Il n'y a pas de produit groupware regroupant toutes ces technologies qui ont une particularité commune : tout se passe comme s'il s'agissait d'une relation de client à client et non une relation client serveur.

Les applications sont créées à l'aide d'un langage de script et des gabarits d'interface fournis par le vendeur. Le middleware est également spécifique de l'éditeur.



Client / serveur avec serveur de transactions

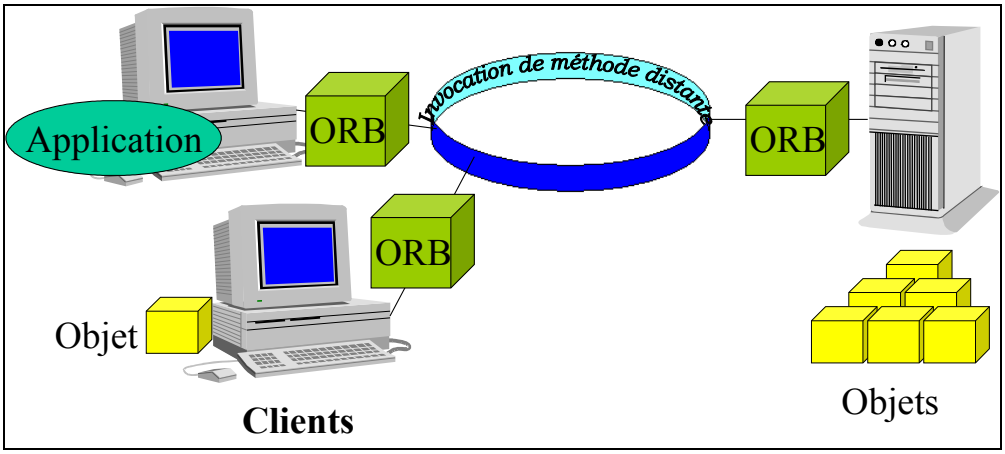
E. Serveur d'objets

L'application client serveur est écrite sous forme d'objets communiquants. Les objets client communiquent avec les objets serveur au moyen d'un négociateur de requêtes objet ou ORB (Object Request Brocker).

Fonctionnement : (Annexes page 36)

- Le client appelle une méthode appartenant à une classe du serveur objet.
- L'ORB localise une instance de la classe, appelle la méthode demandée et renvoie le résultat à l'objet client.

Les serveur d'objets doivent bien sûr traiter le partage des objets. Quelques ORB sont conforme au standard CORBA de l'OMG.



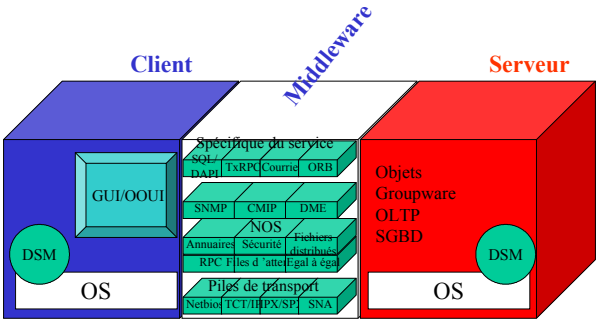
Client serveur avec objets distribués

F. Serveur Web

Il s'agit d'une révolution dans l'architecture client serveur. Elle a été possible grâce à la croissance de la bande passante et à l'apparition des systèmes d'exploitation multithread (Annexes page 37) dotés de fonctionnalités réseau.
C'est le pays de cocagne du client serveur : toute machine située sur une autoroute de l'information peut être à la fois client et serveur.
Un graphique complet sera présenté plus loin.

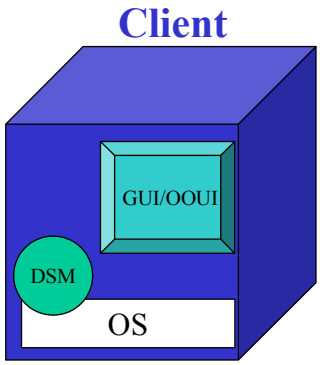
IV. Les cubes du jeu de construction

A. Présentation



Bien que les chapitres qui suivent ne traitent que du middleware SQL et Web, ce paragraphe traitera l'architecture client serveur en général.

1. Le cube client

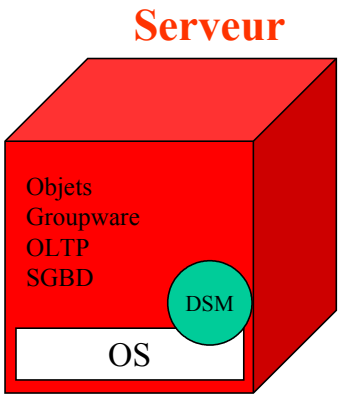


Il exécute la partie cliente de l'application et tourne sous un système d'exploitation qui fournit l'interface graphique utilisateur. On distingue deux sortes d'interfaces graphiques :

- GUI (graphic user interface)
- OOGUI (object oriented graphic user interface)

Le cube client accède à des services répartis et passe la main au middleware pour les services non locaux. En général, il exécute également un composant de l'administration du système complet. (DSM : gestionnaire distribué de système)

2. *Le cube serveur*

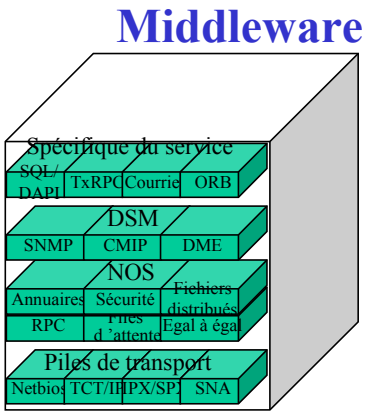


Le cube serveur exécute la partie serveur de l'application. Celle-ci s'installe par dessus un logiciel serveur spécifique au type d'application :

- objets,
- web,
- groupware,
- OLTP,
- SGBD.

Le système d'exploitation assure l'interface avec le cube middleware. Le cube serveur exécute également un composant DSM : soit un agent de celui-ci, soit la partie serveur de ce logiciel.

3. *Le cube du middleware*



Il se trouve sur la partie client et serveur d'une application. Le middleware peut être décomposé en trois parties :

- piles de transport,
- système d'exploitation réseau,
- middleware spécifique au service rendu.

A ces trois parties, il faut ajouter les composants du DSM.

4. Les relations entre serveurs

Ces relations sont par nature de type client serveur. Un serveur est client d'un autre serveur. Cependant, certaines actions nécessitent un middleware spécifique : on peut citer par exemple le protocole de validation à deux phases (two phase commit) qui intervient dans la validation des requêtes multi-serveur.

B. Anatomie d'un client

1. Les différents clients

Le cube client fournit l'aspect, la présentation des services offerts par le système. Toutes les applications clientes ont une **fonction commune** : elle réclament les services d'un serveur. Elles se différencient par contre par la façon dont elles déclenchent leurs requêtes et par l'interface graphique.

On distingue ainsi

- les clients sans interface graphique
- les clients à interface graphique
- les clients à interface graphique orientée objet.

1.1. Clients sans interface graphique

Testeurs, fax, stations service intelligentes, téléphones cellulaires, robots, lecteurs de codes barres, ardoises intelligentes se caractérisent par un minimum d'interaction humaine. Certaines de ces applications peuvent nécessiter des services multitâches.

1.2. Clients à interface graphique

Les requêtes au serveur résultent de l'interaction entre l'utilisateur et l'interface graphique. Ces applications graphiques clientes sont en général la transcription graphique des dialogues qui s'instauraient auparavant sur des terminaux passifs. Le dialogue relève du modèle objet/action dans lequel l'utilisateur choisit l'objet à traiter puis la commande à lui appliquer. Il s'agit de l'interface courante pour les applications OLTP et serveur bases de données. Ce modèle d'interaction avec l'utilisateur est appelé modèle graphique CUA 89 (exemple : Windows 3.X, OSF Motif).

1.3. Client à interface graphique orientée objet

Ces applications sont utilisées par des personnes qui effectuent des tâches multiples, variables et sans ordre prédéfini. Les objets du bureau communiquent entre eux et avec les serveurs externes par des dialogues en temps réel, interactifs et simultanés.

En général, on ne sait pas où commence l'application et où finit le bureau du système. Les objets et programmes du bureau peuvent être réunis pour accomplir une tâche (exemple : le dépôt de l'objet réservation sur l'icône fax provoque la transmission du fax de réservation). Dans ce type d'interface, l'application est transparente à l'utilisateur ou du moins ses limites sont floues : le bureau est une collection d'objets qui coopèrent et de fenêtres associées à ces objets. A l'inverse, dans une interface GUI, l'application est lancée en cliquant sur une icône. A partir de là, il faut utiliser les menus et sous menus : la structure de travail est rigide, les contours de l'application nets.

2. L'OS du client

Le tableau ci-dessous présente les caractéristiques de l'OS d'une machine cliente. Le cas de l'interface non GUI n'est pas présenté car il s'éloigne du contexte de ce cours.

Dispositif de l'OS	*Client GUI*	*Client OOUI*
Mécanisme de requête / réponse (de préférence avec transparence local / distant)	OUI	OUI
Transfert de fichiers textes, d'images, d'extraits de bases de données	OUI	OUI
Multitâche préemptif (1)	Préférable	OUI
Prioritisation des tâches	Préférable	OUI
Communications interprocessus	Préférable	OUI
Threads (2) pour communications d'arrière plan avec le serveur et réception des appels	OUI (sinon : sablier)	OUI
OS robuste avec protection intertâches et appels OS réentrants (3)	Préférable	OUI
Modèle graphique CUA 89	OUI	OUI
Interface OOUI	NON	OUI

(1) Préemptif : lorsqu'une une ressource est attribuée à un processus, elle peut lui être retirée (pour attribution à un autre processus) sans provoquer de dysfonctionnement si le système est préemptif.

(2) En utilisant des threads différents pour l'interface utilisateur et les tâches d'arrière plan, le programme peut répondre aux actions de l'utilisateur alors qu'un thread séparé prend en charge la relation avec le serveur.

(3) Réentrants : les primitives du noyau peuvent être utilisées par plusieurs processus applicatifs.

3. Evolution

L'intelligence et les données se déportent de plus en plus vers le cube client. Aussi, dans cette évolution, les OS clients doivent être en mesure de fournir une fonction de serveur « léger » car :

- les clients bases de données conservent localement des extraits de tables,
- les clients moniteurs transactionnels coordonnent les transactions multiserveur,
- les clients groupware gèrent les files d'attente,
- les clients multimédia enregistrent les dossiers en entrée et en sortie
- et les clients objets répartis acceptent des requêtes d'objets situés n'importe où.

Par opposition à un serveur complet, un serveur léger n'a pas besoin de supporter l'accès simultané à des ressources partagées, l'équilibrage de la charge et les communications multithread.

C. Anatomie d'un serveur

1. Les activités d'un serveur

Le rôle d'un serveur est de fournir des services à de multiples clients intéressés parses ressources partagées. Voici les principales activités qui en découlent.

1.1. Attente des requêtes émises par les clients

Un serveur passe le plus clair de son temps à attendre les requêtes de ses clients qui lui parviennent sous forme de messages.

Plusieurs techniques peuvent être mises en œuvre pour traiter ces requêtes :

- assignation d'une session particulière à chaque client,
- création d'un jeu dynamique de sessions réutilisables,
- solution mixte.

La performance d'un serveur sera liée à son aptitude à répondre à tous ses clients en affrontant les heures de pointe.

1.2. Exécution simultanée de nombreuses requêtes

Un client ne doit pas monopoliser les ressources d'un serveur et celles-ci doivent rester intègres. Pour cela, un serveur doit être multitâches, multithread et protéger ses ressources partagées.

1.3. Attribution de priorités

Un serveur doit être en mesure de proposer différents niveaux de priorité à ces clients. Une impression, un traitement par lots peuvent être différés lorsque survient un traitement transactionnel.

1.4. Lancement et exécution de tâches de fond

Certaines tâches indispensables mais non liées à un besoin immédiat doivent pouvoir être lancées et exécutées en tâches de fond. Citons par exemple, le rafraîchissement H+24 d'une base de données.

1.5. Fonctionnement ininterrompu

Un programme serveur est une application critique devant fonctionner 24 heures sur 24 sans rupture de service. Le serveur et son environnement doivent donc être particulièrement robustes.

1.6. Voracité

L'inflation des besoins en mémoires et puissance est permanente. Le programme serveur et son environnement doivent être évolutifs.

2. L'OS d'un serveur

Les services de base font partie d'un système d'exploitation (serveur) de base tandis que les services étendus sont des composants logiciels modulaires complémentaires. La limite entre les deux est floue : les extensions d'aujourd'hui seront vraisemblablement intégrées demain.

2.1. Services de base

Pour offrir un haut niveau de simultanéité, il est souhaitable d'assigner une tâche à chaque client servi par le serveur à un instant donné. De plus, les applications complexes peuvent être décomposées en un ensemble de tâches concurrentes et logiquement distinctes ce qui a pour effet d'améliorer les performances, le débit, la modularité et la réactivité du serveur. En allant encore plus loin, le code du serveur sera plus efficace si les tâches sont allouées à différentes parties d'un même programme plutôt qu'à des programmes différents. Ces tâches, unité élémentaire d'exécution, sont alors appelées thread.

Tous les services de base sont liés à la nécessité de disposer d'une gestion multitâches voire multithread et aux mécanismes associés à cette gestion.

2.1.1. Préemption des tâches

Stratégie qui autorise la suspension de l'exécution d'un processus au profit d'un autre. Elle s'oppose à la stratégie d'exécution jusqu'à achèvement des premiers systèmes. Lorsque le système d'exploitation gère lui même cet aspect de préemption des tâches, les programmes serveurs sont beaucoup plus sûrs et plus simples à écrire.

2.1.2. Priorité des tâches

Le système d'exploitation doit ordonnancer les tâches selon leur niveau de priorité.

2.1.3. Sémaphores

Le système d'exploitation doit offrir un mécanisme simple pour éviter que des tâches s'exécutant « simultanément » soient en conflit lors d'accès à des ressources partagées. Ces mécanismes, les sémaphores, permettent de synchroniser les actions de plusieurs tâches serveur indépendantes et de les alerter si une erreur survient.

2.1.4. Communications interprocessus

Le système d'exploitation doit fournir les outils nécessaires à l'échange et au partage de données entre processus indépendants.

2.1.5. Communications entre processus locaux ou distatnts

Le système d'exploitation doit être capable d'étendre les communications interprocessus au-delà de la machine et ce de façon transparente pour l'application.

2.1.6. Threads

Unités élémentaires d'exécution. Ils sont utilisés pour créer des programmes événementiels à traitements simultanés. Chaque événement en attente peut être

affecté à un thread qui attend que cet événement survienne. Pendant ce temps d'attente, d'autres threads du même programme peuvent être activés.

2.1.7. Protection intertâches

Une tâche ne doit pas provoquer l'écroulement du système. L'OS doit prévenir les interférences entre tâches sur les mêmes ressources.

2.1.8. Système de fichier multi-utilisateurs à haute performance

Le système de gestion de fichiers doit accepter le multitâche et offrir les verrous nécessaires au maintien de l'intégrité des données. Le système doit pouvoir ouvrir un grand nombre de fichiers sans conséquences sur les performances.

2.1.9. Gestion efficace de la mémoire

La gestion de la mémoire doit pouvoir prendre en compte des blocs de données et programmes volumineux en lecture, écriture, par blocs de taille adaptée.

2.1.10. Extensions liées dynamiquement à l'exécution

Les services du système d'exploitation doivent être extensibles. En effet, lors de l'installation du noyau du système, l'administrateur a le choix d'installer tout ou partie de certains services. L'OS doit donc fournir un mécanisme d'extension d'un service pendant son exécution.

2.2. Services étendus

Ces services permettent de faciliter la gestion du système et son évolution. Certains seront intégrés à terme aux systèmes d'exploitation. On peut citer rapidement :

- piles de protocoles : pour permettre la communication avec le plus grand nombre de clients et de serveurs.
- Extension réseau du système d'exploitation : plus particulièrement pour les services de fichiers et d'impression. Pour une application, la localisation de ce genre de périphérique doit être transparente.
- Gestion des gros objets binaires (blobs) : protocoles d'échanges de blobs et d'affectation de ceux-ci aux programmes adaptés.
- Annuaires globaux et pages jaunes : les clients doivent pouvoir localiser les ressources du réseau par leur nom. Ces annuaires doivent faire l'objet d'une mise à jour dynamique par les serveurs qui proposent les ressources.
- Services d'authentification et d'autorisation
- Gestion intégré du réseau et du système : configuration, performances de tous les éléments, alertes en cas d'erreur, distribution de logiciels, détection de virus et d'intrus et mesure du temps d'utilisation des ressources payantes.
- Synchronisation des horloges
- SGBD intégré avec gestion du transactionnel léger voire lourd.
- Services Internet

- Services orientés objets : négociateur, référentiel, échange d'objets.

V. Les briques de base du middleware

Le middleware peut être défini de façon très générale comme étant l'ensemble des logiciels répartis nécessaires à l'interaction entre un client et un serveur.

Son domaine d'action couvre l'API coté client (demandeur du service), la transmission de la requête sur le réseau.

Par contre, le logiciel fournissant le service demandé, l'interface utilisateur et la logique de l'application ne sont pas de son ressort.

Dans ce chapitre, nous découperons le middleware en deux grandes parties et citerons des exemples de produits correspondants puis nous aborderons les principes des techniques de communication.

A. A chacun son middleware

Tenter de découper le middleware en plusieurs parties relève de l'exercice intellectuel purement formel. Il n'y a pas de stricte vérité en la matière : les produits qui seront cités ci-dessous et dans la suite de ce chapitre remplissent un certain nombre de fonctions qu'il est souvent difficile de cataloguer. Il ne s'agit donc là que d'une proposition de découpage qui a le mérite de fixer les idées mais qui peut être facilement contestée.

De façon très globale, on peut considérer qu'il existe un middleware général et le middleware propre à un service.

1. Le middleware général

Il comporte :

- les piles de communication,
- les annuaires répartis,
- les services d'authentification,
- la synchronisation horaire sur le réseau,
- l'appel de procédures distantes,
- la gestion des files d'attentes.

Citons quelques produits qui remplissent tout ou partie de ces fonctions : DCE, ONC+, Ntware, Named Pipes, Lan server, Lan manager, Vines, TCP/IP, APPC, NetBios, MOM (Message Oriented Middleware) …

2. Les middleware propres à un service

- Pour les bases de données
  ODBC, DRDA, EDA/SQL, SAG/CLI, et Clue (Oracle).
- Transactionnel
  ATMI et Ws/Tuxedo.
  Transactionnel RPC (Encina).
  Tx RPC et XATMI (X/OPEN)
- Groupware
  MAPI, VIM, VIC, SMTP et les appels à Lotus Notes.

- Objets répartis
  CORBA (OMG), Netwotk OLE/DCOM (Microsoft)
- Internet
  HTTP, S-HTTP, SSL
- Administration du système
  SNMP, CMIP et les ORB

B. Principes des techniques de communication

Comment font un client et un serveur pour communiquer ?

Comment sont synchronisées les requêtes et réponses ?

Comment traiter les différences de représentation des données entre machines ?

Que se passe-t-il quand un interlocuteur est indisponible ?

Voilà les principales difficultés qui doivent être résolues par les techniques de communication. De plus, le NOS doit assurer la transparence de l'informatique répartie et nous éviter ainsi de traiter avec les protocoles de communication, les réseaux et piles de transport.

Le NOS offre trois types d'interfaces de communication :

- poste à poste,
- RPC
- MOM.

Les communications poste à poste s'effectuent selon une sémantique d'envoi / réception que l'on qualifie 'au raz du câble'. RPC, l'appel de procédures distantes, donne le sentiment que tout serveur est accessible par un appel de fonction. MOM pratique l'échange de messages au travers des files d'attentes.

1. Communication de poste à poste : l'exemple des sockets

Les sockets sont supportées pratiquement par tous les systèmes d'exploitation. L'API de Windows appelée Winsock standardise l'utilisation de TCP/IP sous Windows.

Les sockets constituent un standard de fait pour les fournisseurs d'applications sur les réseaux TCP/IP.

Principe

Pour établir une communication entre processus par sockets, le programmeur fait appel à des services définis par le RFC. Chaque service est identifié pour tous les systèmes d'exploitation par une adresse sur 16 bits. Sur le système d'exploitation Windows NT4, le fichier 'services' donne la liste des services disponibles, leur adresse et le type de protocole utilisé : TCP ou UDP. Avec TCP, l'envoi est effectué avec assurance de réception. UDP est qualifié de mode non connecté : la communication est plus rapide mais la réception n'est pas certaine.

Dans tous les cas, la communication s'établit par l'ouverture d'un canal de communication en précisant l'adresse IP de la machine distante (32 bits) ainsi que l'adresse du service utilisé.

La communication est constituée de trames d'information dont le formalisme est particulier à chaque service.

L'application WTNVT a été développée en utilisant les services sockets.

2. L'appel de procédures distantes (RPC)

RPC utilise l'appel de procédure connu des programmeurs : un processus client appelle une fonction d'un serveur et suspend son exécution jusqu'à obtention du résultat..

Principe de fonctionnement

Le logiciel exécutable RPC rassemble les valeurs des paramètres de communication, constitue un message et l'expédie au serveur distant.

Celui-ci reçoit la requête, désassemble les paramètres, appelle la procédure cible et retourne la réponse au client.

Derrière cette description de principe se cachent de nombreuses difficultés qui doivent être résolues par RPC :

- la localisation et le lancement des procédures du serveur,
- le formatage et la transmission des paramètres,
- le traitement des pannes (non réponse d'un correspondant),
- la sécurité
- la localisation du serveur,
- le codage des données pour assurer une communication indépendante du codage de chaque correspondant.

3. MOM

Il s'agit de la méthode de communication la plus simple si l'application tolère un niveau de réponse indépendant du temps.

On parle alors d'architecture client serveur nomade : il est possible d'accumuler les transactions en attendant que la connexion soit établie. Le client et le serveur peuvent fonctionner à des moments différents.

Principe de fonctionnement

Les applications communiquent par dépôt et retrait de message au travers de files d'attente.

VI.Le middleware SQL

A. Rappels SQL

SQL est un langage ensembliste puissant spécialisé pour les bases de données relationnelles.

Il permet essentiellement la manipulation, la définition et le contrôle des données.

Il est caractérisé par la séparation nette de l'aspect physique et de la représentation logique des données l'accès aux données est logique, sans aucune considération du support physique utilisé.

1. Les normes.

SQL 89 correspond à l'unification des différents produits existants à l'époque.

SQL 92 (SQL 2) est un sur ensemble de SQL 89. L'ISO suggère trois étapes pour effectuer la transition de SQL 89 à SQL 92. Ces trois étapes correspondent aux trois niveaux que constituent SQL 92 :

- Le niveau entrée permet une transition simple avec peu de reprise des applications.
- Niveau intermédiaire
- Compatibilité totale.

SQL 3 est la future version. Le texte préliminaire est déjà en circulation.

Il ne faut jamais perdre de vue que SQL reste une norme sa seule existence physique à ce titre est un support papier. Les éditeurs s'efforcent de coller au plus près de la norme mais fournissent toujours leur propres particularités. En utilisant un SGBD à la norme SQL, il faut donc effectuer le choix de se limiter strictement aux possibilités de la norme ou d'utiliser la puissance des particularités du SGBD avec en contre partie, une perte de portabilité.

2. ESQL

Depuis la version SQL 92, ce standard propose embedded SQL, c'est à dire l'inclusion d'instructions SQL dans les langages de troisième génération (C, COBOL, FORTRAN, PASCAL, MUMPS, ADA …

Des marqueurs propres à chaque langage permettent de délimiter les instructions SQL au sein des programmes.

Le SQL source doit être traité par un pré-compilateur pour obtenir un fichier de code reonnu par le compilateur du langage.

L'utilisation de embedded SQL pose quelques problèmes évoqués ci-dessous :

- la base cible doit être connue au moment de l'écriture du programme.
- A l'installation, les applications doivent être liées au serveur auquel elles se connectent.
- Les pré-compilateurs sont associés au SGBD : il faut donc re-compiler le code SQL incorporé pour chaque serveur venant d'un autre éditeur.

Tout ceci constitue un handicap pour les fournisseurs d'applications client serveur clef en main.

B. Interface SQL à appel direct (CLI)

Exemple de la CLI du SAG devenue la CLI de X/OPEN.

44 éditeurs de bases de données ont créé le consortium SAG dans le but de proposer des standards d'accès aux bases de données distantes.

C'est une API SQL qui ne nécessite pas de pré_compilateur.

L'API permet de créer et exécuter les instructions SQL au moment de l'exécution ce qui permet de développer des applications portables indépendantes de tout produit SGBD.

Les fonctions SQL correspondent au standard SQL 89.

On peut dégager les grands principes suivants :

- sémantique et syntaxe SQL communs
- codification des types de données
- notification et traitement des erreurs
- définition d'un ensemble de catalogues système
- service de gestion des connexions (Exemple Busines Object) qui permet au client de spécifier les connexions aux SGBD distants.

Plus précisément, les API du SAG permettent de réaliser les opérations suivantes :

- connexion à une base de données à travers un pilote local (3 appels)
- préparation de requêtes (5 appels)
- exécution de requêtes(2 appels)
- récupération des résultats (7 appels)
- fin de commandes (3 appels)
- fin de connexion

Par opposition à l'exécution de requêtes, la préparation permet de mémoriser des requêtes au niveau du serveur SQL de n'effectuer qu'une seule fois leur traitement par le serveur de données puis d'y faire appel pour exécution.

C. L'interface CLI ODBC de Microsoft

ODBC est l'API SQL de Windows. C'est une version améliorée de la CLI du SAG.

ODBC comporte 54 appels, dont 23 constituant le noyau sont basés sur la CLI du SAG.

Il existe 3 niveaux de conformité :

- **le noyau** comporte 23 appels pour assurer la connexion, déconnexion, l'extraction de résultats, la validation et l'annulation de transctions.
- **Le niveau 1** complète le noyau par 19 appels qui permettent :
  - L'exploitation du catalogue d'une base de données,
  - l'extraction de gros objets (BLOB),
  - l'utilisation de fonctions spécifiques des pilotes.
- **Le niveau 2** comporte 19 appels pour assurer l'extraction de données sous contrôle de curseurs avec défilement avant et arrière.

La plupart des éditeurs de serveurs supportent l'API ODBC en plus de leur API SQL native. On constate à ce jour une utilisation fréquente de ODBC dans les outils d'aide à la décision.

ODBC comporte trois ensembles : le gestionnaire de pilotes, les pilotes et les sources.

Le gestionnaire de pilotes (accessible depuis le système d'exploitation) est une dll (librairie dynamique) qui effectue le routage des appels de fonctions vers le pilote concerné.

Le pilote (également une librairie dynamique) est spécifique au SGBD cible le gestionnaire de pilotes gère autant de pilotes que de SBGBD auxquels Microsoft souhaite s'interfacer.

Il réceptionne les appels de fonction et les traduit en langage d'accès natif du SGBD cible.

La source. La mise en place d'une configuration de travail avec ODBC nécessite la création d'une source qui est en fait la duplication du pilote approprié, complété par les paramètres de la base de données cible.

VII.Client serveur et INTERNET

A. Présentation

1. INTERNET : le plus grand réseau public du monde

30 000 réseaux interconnectés sur 70 pays ; ce chiffre double chaque année.

Les mêmes protocoles, interfaces, et infrastructures réseau peuvent être utilisés pour des réseaux privés (INTRANET).

2. Web : world wide web

L'application qui a fait connaître INTERNET au grand public. C'est la plus grande application C/S du monde.

Quelques chiffres du juin 1996.

- plus de 100 000 serveurs
- 500 000 pages d'accueil
- 30 millions d'utilisateurs (1 million de plus par mois)
- 3 000 nouveaux sites par jour
- web double sa taille tous les 53 jours.

Les protocoles applicatifs du web tournent tels quels sur INTERNET.

On peut considérer le web comme l'addition des protocoles d'INTERNET.

Tous sont des services qui tournent sous TCP/IP.

Num service	*Sigle*	*Protocole*
25	SMTP	Simple Mail Transfert Protocol
23	Telnet	
21	FTP	File transfert Protocol
119	NNTP	Network News Transfert Protocol
70	Gopher	

B. Technologies C/S derrière le Web : les fondations

Dans une première génération, le web a été limité à la lecture : les documents ne sont pas modifiables mais les principales difficultés sont contournées.

On peut considérer le web comme un système hypertexte global.

1. Hypertexte

Mécanisme logiciel qui lie un document à d'autres sur la même machine ou à travers le réseau. Les documents liés peuvent eux-mêmes contenir des liens vers d'autres documents.

Plutôt que de parler de documents liés, il faut raisonner en ressources liées telles que

- images,
- sons,
- exécutables

Avec le web, le monde devient un gigantesque document hypertexte.

2. Les URL (Uniform Ressource Locator)

Toutes les ressources du web sont nommées selon le même protocole URL.

URL définit ou se trouve une ressource et comment y accéder.

URL supporte les protocoles web les plus récents (HTTP) et les plus anciens (FTP, Goffer etc.). URL est une chaîne de caractères ASCII imprimable comprenant quatre parties.

Nom du protocole	*@ serveur*	*N° du port*	*Ressource*
HTTP (natif) ***FTP (le + ancien)*** ***Gopher (précurseur)*** ***News*** ***Mailto*** ***WAIS***	://www.adresse.com Nom du site ou adresse IP	N° spécifié ou à défaut port standard (80, 21, 70)	:/path/dir/file Certains protocoles acceptent le point d'interrogation pour rechercher une ressource (HTTP, Gopher, WAIS)

3. HTTP: Hypertext transfert protocol)

Protocole semblable au RPC qui permet d'accéder aux ressources désignées par leur URL.

HTTP est un protocole applicatif (dernier niveau du modèle OSI) indépendant du protocole de transport (en général TCP / IP). L'échange entre un client et un serveur du web est sans session donc sans synchronisation. Le demon HTTP du serveur tourne en écoute de son port de communication. A la réception d'une demande, il

- établit une connexion C/S

- reçoit ou transmet les paramètres (dont la ressource désignée par son URL)
- coupe la connexion C/S.

Pour un serveur HTTP, tous les clients sont égaux : ils ne distinguent pas un client MAC d'un PC. Ce sont les navigateurs qui s'occupent des détails propres à chaque plate-forme en informant les serveurs des données qu'ils reconnaissent.

La requête émise par le client peut faire appel essentiellement à 4 méthodes (il en existe 13 dans la version HTTP 1.0) reconnues par le serveur HTTP : GET, HEAD, POST, PUT

GET : demande l'extraction de l'URL désignée.

HEAD : est une demande identique à GET. Le serveur ne renvoie pas l'URL désignée mais seulement les en-têtes de réponse ce qui permet au client de tester la validité des liens hypertexte.

POST : transmet des données du client à l'URL désignée du serveur.

PUT : permet de transmettre une page HTML complète au serveur.

Ces deux dernières méthodes sortent du domaine de la simple consultation pour offrir une possibilité d'interaction entre client et serveur.

4. HTML : Hypertext Markup Language

Langage qui permet d'inclure des liens hypertexte et de décrire la structure logique des documents. Entièrement portable sur tout système d'exploitation, toutes architectures CPU.

Le web est une collection de documents HTML liés les uns aux autres.

Un document du web est un fichier texte ASCII dans lequel on imbrique des commandes HTML pour :

- décrire la structure du document
- donner des informations sur les polices d'affichage et les graphiques,
- lier d'autres ressources.

Les documents HTML vivent dans des serveurs HTTP : les serveurs web.

5. Le navigateur web : client universel

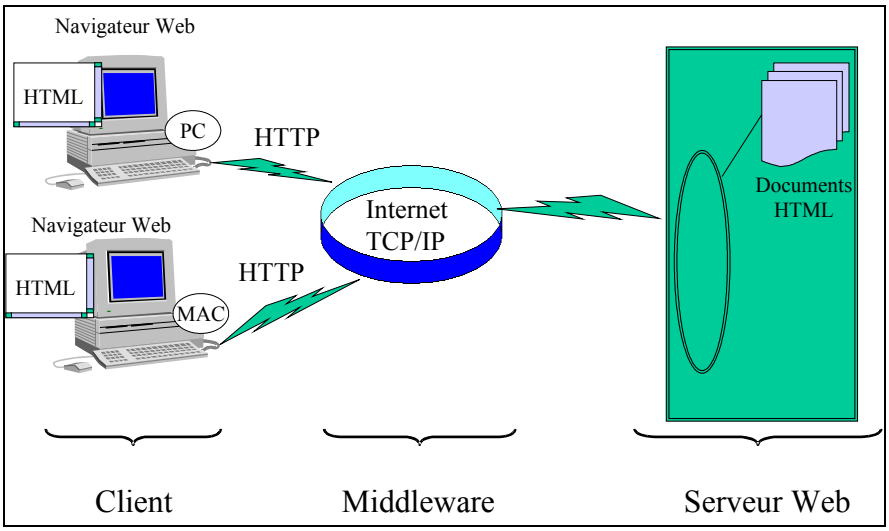
Il interprète les commandes du serveur et présente le contenu d'une page HTML en exécutant des commandes HTML.

Exemples : Netscape, Spyglass sont des interpréteurs de commandes HTML.

Le navigateur utilise les liens hypertexte pour aller d'une page à l'autre et s'occupe des détails propres à chaque plate-forme en informant les serveurs des données qu'ils sont en mesure de reconnaître. C'est ainsi que pour un serveur HTTP, tous les clients semblent égaux et que l'on qualifie un navigateur web de client universel.

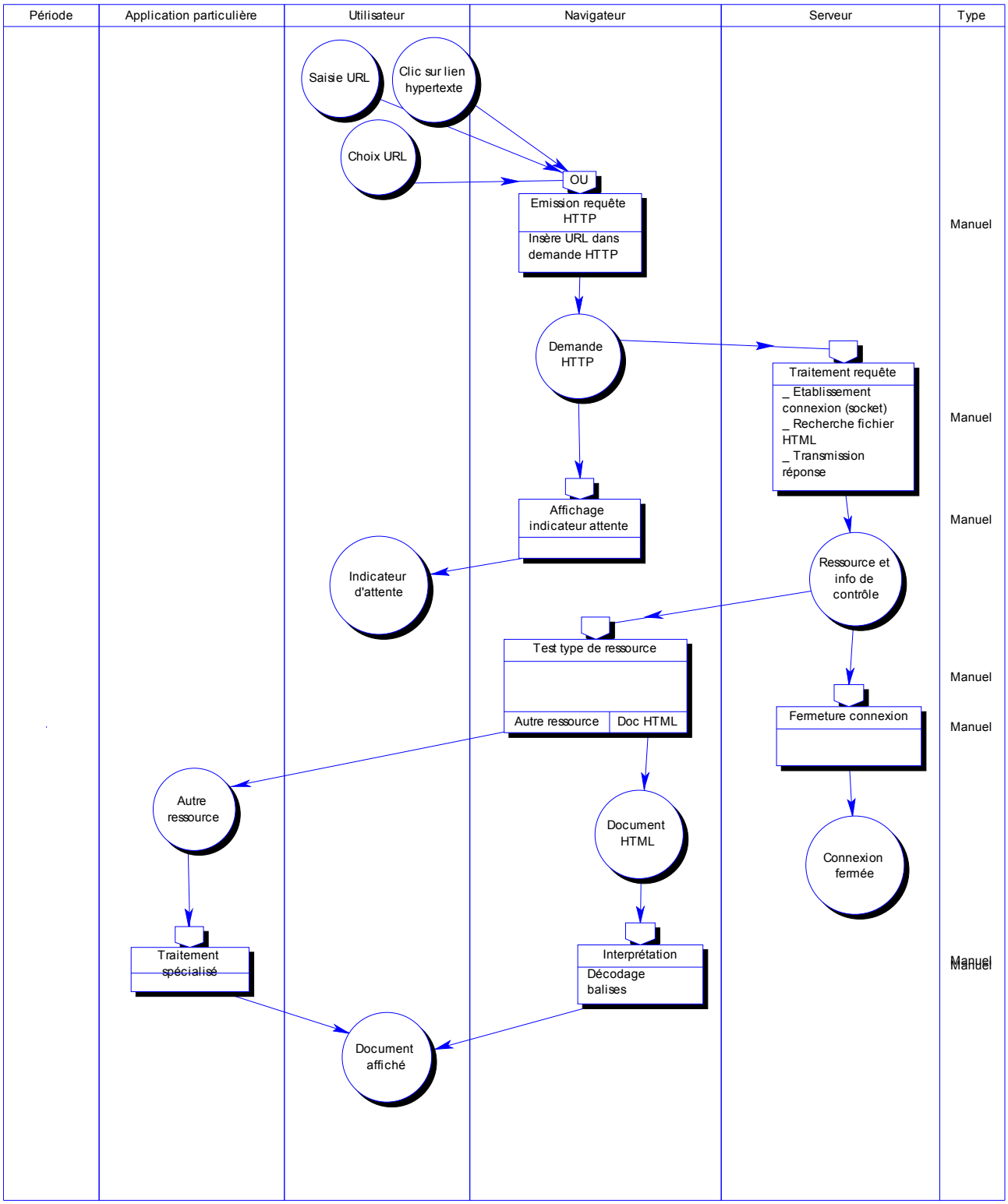
C. Interaction entre un client et un serveur web

1. Architecture



Interaction client / serveur sur le Web : consultation

2. *MOT de l'interaction*



D. Les formulaires, le protocole CGI

Jusqu'à présent, notre raisonnement s'est limité à une première génération dans laquelle les applications web sont limitées à la lecture. L'objectif est maintenant de dépasser cette limite et d'offrir au client la possibilité d'accéder à n'importe quel service (SGBD, OLTP …) avec des actions de mise à jour. Comment peut-on passer d'une navigation passive (lecture simple) à un support C/S interactif (mise à jour) ?

Les formulaires et le protocole CGI sont les deux points clefs de cette évolution.

1. Les formulaires

Un formulaire est une page HTML contenant un ou plusieurs champs à remplir et un bouton «soumettre » qui permet d'envoyer au serveur Web les données contenues dans le formulaire.

Le navigateur collecte les données du formulaire, les rassemble dans un message HTTP comportant une des méthodes de mise à jour (GET ou POST) ainsi que la désignation du programme CGI à exécuter.

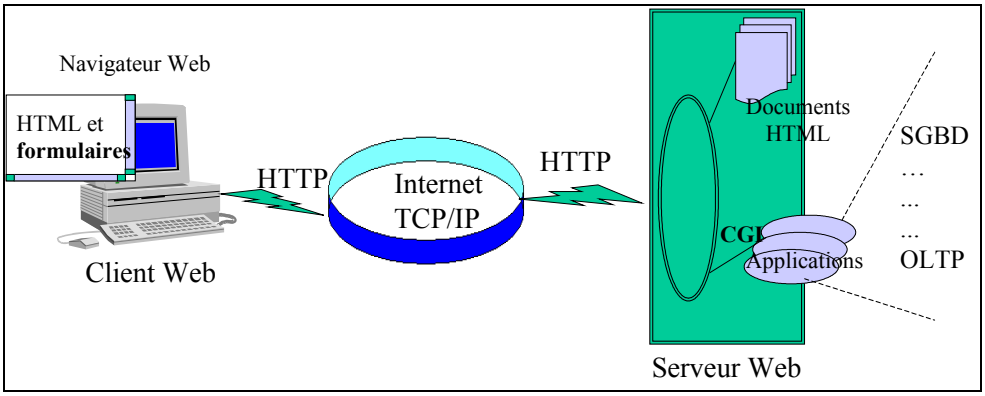
2. Le protocole CGI

CGI (Common Gateway Interface) est un standard de relativement bas niveau qui permet d'écrire des programmes compatibles avec quasiment tous les serveurs HTTP.

Un programme CGI doit être en mesure :

- de récupérer les données du formulaire d'entrée,
- interagir avec une ressource d'arrière plan (un SGBD, une transaction),
- prendre en compte les réponses issues de cette ressource pour les mettre au format HTML,
- transmettre ce résultat au serveur HTTP qui se charge de le retourner au client.

Le diagramme d'architecture précédent se complète ainsi.



Architecture client serveur Web : forme interactive

3. Inconvénients du protocole CGI

Chaque invocation provoque le lancement d'un exécutable CGI qui ne peut rien conserver entre deux appels. En effet, ce protocole est sans état : il oublie tout après avoir fournit sa réponse au client. Il existe des solutions pour contourner ce problème ce qui est heureux car la plupart des actions d'un client peuvent difficilement se concevoir à l'aide d'un seul formulaire (exemple : formulaire de sélection de marchandises, suivi du formulaire qui présente la facture et demande les éléments de paiement).

De plus, les exécutables CGI ne sont pas partageables entre plusieurs clients simultanés : il faut en lancer autant d'instance que de clients.

E. Conclusion

Malgré quelques inconvénients, la simplicité des techniques mise en œuvre sur le Web a assuré leur succès. Le protocole CGI a ouvert la voie à une architecture de forme client serveur de présentation dans laquelle le client, muni d'un navigateur Web (client universel disponible sous tous les environnements) ne supporte pas le code applicatif. On évite ainsi les problèmes liés à la configuration des machines clientes, à la diffusion et à la mise à jour des applications qui représentent un frein considérable à l'architecture client serveur classique.

VIII.Glossaire

API
Application programming Interface. Interface de programmation d'applications.

CLI
Client Interface ? Call Level Interface ?

DSM
Gestionnaire distribué de système. L'application DSM s'exécute sur chaque nœud d'un réseau. La station gestionnaire récupère les informations auprès de tous les agents et les présente à l'administrateur sous forme généralement graphique. La station gestionnaire peut aussi déléguer des actions à l'un de ses agents : il s'agit d'un réseau dans le réseau.

DLL
Dynamic link library : bibliothèque de fonctions liées dynamiquement à un programme (lors de son chargement en mémoire) par opposition aux bibliothèque ajoutées à un exécutable lors de l'édition des liens.

GARTNER GROUP

ISO
International Standard Organisation. Organisme intégré aux nations unies qui entérine les standards internationaux. L'ANSI est sa branche US.

MOM
Message Oriented Middleware.

NOS
Network Operating System. Système d'exploitation réseau.

ODBC
Open DataBase Connectivity

OSI
Open Systems Interconnect : une initiative de l'ISO pour standardiser les protocoles réseaux. Il ne reste de cette initiative que le modèle à 7 couches.

RFC
Request For Comments

RPC
Remote Procedure Call. Appel de procédure distante.

SAG
Sql Access Group. Groupe de constructeurs indépendants. Ces travaux dans le cadre des médiateurs ont été repris par X/OPEN. Exemple : l'interface d'application standard : l'API CLI.

X/OPEN
Association internationale à but non lucratif (1984) composée de constructeurs essentiellement Américains. Définit et diffuse les technologies et standards en matière de systèmes ouverts. Leurs documents de spécifications précisent et complètent les standards.

IX.Annexes

A. NOTIONS D'OBJETS

1. A l'origine :

Tout programmeur a passé une partie de son temps à écrire des procédures qu'il avait déjà traitées sous une autre forme. De ce problème est née la volonté de réaliser des applications avec des modules réutilisables donc génériques.

L'apparition des langages permettant au programmeurs de créer des structures a été déterminante surtout lorsque ces structures ont été responsables des méthodes qui leur sont associées.

Une classe est une structure composée de propriétés (les différents éléments de la structure) et de méthodes. Les données ne peuvent être manipulées que par les méthodes de la classe.

La programmation par classes (programmation objets) consiste à définir des classes (structures) et les méthodes associées pour manipuler les données.

Exemple
La classe VOITURE, contenant les **propriétés** :
PROPRIETAIRE
ETAT
IMMATRICULATION
et les **méthodes** qui agissent sur les propriétés :
Changement de propriétaire
Changement d'immatriculation
Accélérer,
Freiner

Une **instance** de classe est une variable déclarée comme étant du type de la structure.

2. Héritage :

Il est possible de bâtir une classe à partir d'une autre.

Cette nouvelle classe hérite des propriétés et des méthodes de la classe d'origine. Elle peut bien sûr être complétée par ses propres propriétés et méthodes.

La conception d'une application objet est donc radicalement différente. Il faut faire l'inventaire de toutes les entités que l'on devra manipuler et les regrouper par propriétés et méthodes communes tout en utilisant pleinement le principe d'héritage.

Exemple :

la classe des chaises comportant ses propriétés et ses méthodes peut être utilisée pour créer une classe des chaises à hauteur et dossier réglable. On peut imaginer comme méthodes propres à cette nouvelle classe, le réglage de la hauteur et l'inclinaison du dossier.

3. Utilisation

Après de longues considérations philosophiques on peut admettre que dans l'univers, tout est objet communiquant par des messages. Ainsi, une imprimante en action est un objet imprimante ayant reçu un message d'impression ayant comme paramètre le document à imprimer.

La programmation par classe est ainsi devenue la programmation objet.

L'utilisation d'un objet se fait en précisant l'instance, la méthode et les arguments nécessaires. Méthode et arguments constituent le message.

Instance.methode (arguments)

4. Que signifie « localiser une instance de la classe » ?

Considérons

- la classe EMPLOYE avec ses propriétés et la méthode CALCULSALAIRE ;
- La classe COMMERCIAL qui hérite d'employé et dont la méthode CALCULSALAIRE utilise la méthode du même nom de la classe employé en y ajoutant les commissions.

Localiser une instance de la classe signifie que pour calculer le salaire de M. Dupond, le système devra déterminer à quelle classe appartient l'instance afin de lui appliquer la bonne méthode.

Cette difficulté n'est pas spécifique à une architecture client serveur (elle est traitée par les SGBDO) mais ce type d'architecture la rend encore plus complexe.

B. Workflow

Le workflow est l'acheminement automatique des événements et des tâches d'un programme au programme suivant.

Un progiciel de workflow doit être capable de prendre en compte la description de QUI fait QUOI, QUAND, dans quel but, les itinéraires parallèles (plusieurs acteurs peuvent réaliser la même tâche), la logique de construction dynamique des itinéraires au moment de l'exécution ainsi que les exceptions aux règles. Cette description peut être comparée à celle d'un MOT qui ne serait pas seulement graphique mais également dynamique.

Un workflow permet ainsi de distribuer des rôles, suivre et acheminer le travail en cours, veiller aux dates d'échéance, suivre l'exécution des tâches (QUI, QUAND).

C. Multithreading

Le multithreading correspond au découpage d'une même application en plusieurs tâches, dans le but de tirer partie des machines multiprocesseurs. L'application doit être conçue de façon particulière : elle doit faire appel à des tâches élémentaires, indépendantes les unes des autres, qu'il sera possible d'exécuter en parallèle. Des bibliothèques de fonctions sont disponibles sur le marché. Le principal problème à résoudre est la synchronisation entre threads : il faut éviter par exemple que deux threads réalisent simultanément une écriture dans un fichier. Une des

façons de résoudre cette difficulté est l'utilisation des sémaphores. Le noyau du système d'exploitation LINUX gère l'ordonnancement des threads.