Next Up Previous Contents Index
suivant: Table des matières **Table des matières** **Index**

Vers une architecture n-tiers

Rémi LEBLOND (remi.leblond@free.fr)

Oral probatoire soutenu le 02/12/1999 à 15h30

Document rédigé par Rémi LEBLOND (remi.leblond@free.fr)

Next Up Previous Contents Index

suivant: De l'informatique centralisée au **monter:** Table des matières **précédent:** Table des matières **Table des matières** **Index**

Sous-sections

Introduction

Rappel du sujet

Le client-serveur à deux niveaux fut, ces dernières années, la principale solution préconisée par les constructeurs d'applications.

Ayant acquis de la maturité, l'architecture n-tiers devient aujourd'hui une alternative.

Après avoir présenté brièvement les modèles de type un tiers, deux tiers et trois tiers, le candidat s'attardera sur les architectures logicielles n-tiers. Il présentera les technologies utilisées et les moyens à mettre en oeuvre. Il définira et expliquera les termes clients légers, clients lourds et middleware. Il étudiera les avantages et les inconvénients des différentes architectures. Enfin, il pourra compléter son probatoire par la présentation d'une solution concrète.

Composition du jury

- Président du jury : M. Claude KAISER,
- M. E. MAURICE,
- M. C. KLEINPETER,
- M. J. L. STEFFAN.

Remerciements

Je tiens à remercier vivement M. Emmanuel MAURICE, qui a proposé le sujet de cette étude, pour son soutien et sa disponibilité tout au long de mes recherches.

Je remercie également M. J.L. STEFFAN pour ses précieux conseils et tous les membres du jury, MM. Claude KAISER et C. KLEINPETER pour le temps qu'ils consacrent aux étudiants du CNAM.

Next Up Previous Contents Index

suivant: De l'informatique centralisée au **monter:** Table des matières **précédent:** Table des matières **Table des matières** **Index**

Document rédigé par Rémi LEBLOND (remi.leblond@free.fr)

Next Up Previous Contents Index
suivant: Les trois niveaux d'abstraction **monter:** Vers une architecture n-tiers
précédent: Introduction **Table des matières** **Index**

De l'informatique centralisée au client-serveur

Sous-sections

Document rédigé par Rémi LEBLOND (remi.leblond@free.fr)

Next Up Previous Contents Index
suivant: L'architecture un tiers **monter:** De l'informatique centralisée au **précédent:** De l'informatique centralisée au **Table des matières** **Index**

Les trois niveaux d'abstraction d'une application

En règle générale, une application informatique peut être découpée en trois niveaux d'abstraction distincts :

- **la couche de présentation**, encore appelée IHM[2.1], permet l'interaction de l'application avec l'utilisateur. Cette couche gère les saisies au clavier, à la souris et la présentation des informations à l'écran. Dans la mesure du possible, elle doit être conviviale et ergonomique.

- **la logique applicative, les traitements**, décrivant les travaux à réaliser par l'application. Ils peuvent être découpés en deux familles :
  - **les traitements locaux**, regroupant les contrôles effectués au niveau du dialogue avec l'IHM, visant essentiellement le contrôle et l'aide à la saisie,

  - **les traitements globaux**, constituant l'application elle-même [JFG99]. Cette couche, appelée *Business Logic* ou couche métier, contient les règles internes qui régissent une entreprise donnée [FAS99].

- **les données**, ou plus exactement l'accès aux données, regroupant l'ensemble des mécanismes permettant la gestion des informations stockées par l'application.

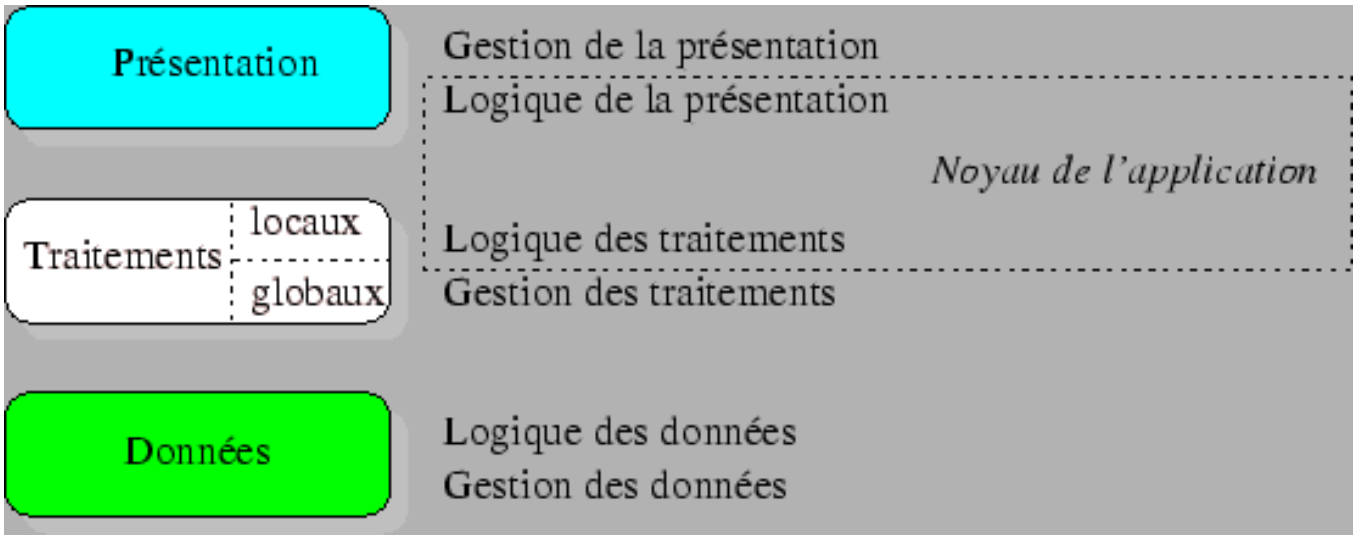

Figure 2.1: Les trois niveaux d'une application informatique [LEF94]

Ces trois niveaux peuvent être imbriqués ou répartis de différentes manières entre plusieurs machines physiques.

Le noyau de l'application est composé de la logique de l'affichage et la logique des traitements. Le découpage et la répartition de ce noyau permettent de distinguer les architectures applicatives suivantes :

- l'architecture 1-tiers[2.2],
- l'architecture 2-tiers,
- l'architecture 3-tiers,
- les architectures n-tiers.

Nous allons faire un rapide tour des premières architectures, en introduisant progressivement les notions utiles à leur compréhension, et nous attarder ensuite plus longuement sur les architectures n-tiers.

Next Up Previous Contents Index

suivant: L'architecture un tiers **monter:** De l'informatique centralisée au **précédent:** De l'informatique centralisée au **Table des matières** **Index**

Document rédigé par Rémi LEBLOND (remi.leblond@free.fr)

Next Up Previous Contents Index
suivant: Le schéma du Gartner **monter:** De l'informatique centralisée au **précédent:** Les trois niveaux d'abstraction **Table des matières** **Index**

Sous-sections

L'architecture un tiers

Présentation

Dans une application un tiers, les trois couches applicatives sont intimement liées et s'exécutent sur le même ordinateur. On ne parle pas ici d'architecture client-serveur, mais d'informatique centralisée.

Dans un contexte multi-utilisateurs, on peut rencontrer deux types d'architecture mettant en oeuvre des applications un tiers :

- des applications sur site central,
- des applications réparties sur des machines indépendantes communiquant par partage de fichiers.

Les solutions sur site central (*mainframe*)

Historiquement, les applications sur site central furent les premières à proposer un accès multi-utilisateurs. Dans ce contexte, les utilisateurs se connectent aux applications exécutées par le serveur central (le *mainframe*) à l'aide de terminaux passifs se comportant en esclaves. C'est le serveur central qui prend en charge l'intégralité des traitements, y compris l'affichage qui est simplement déporté sur des terminaux passifs.

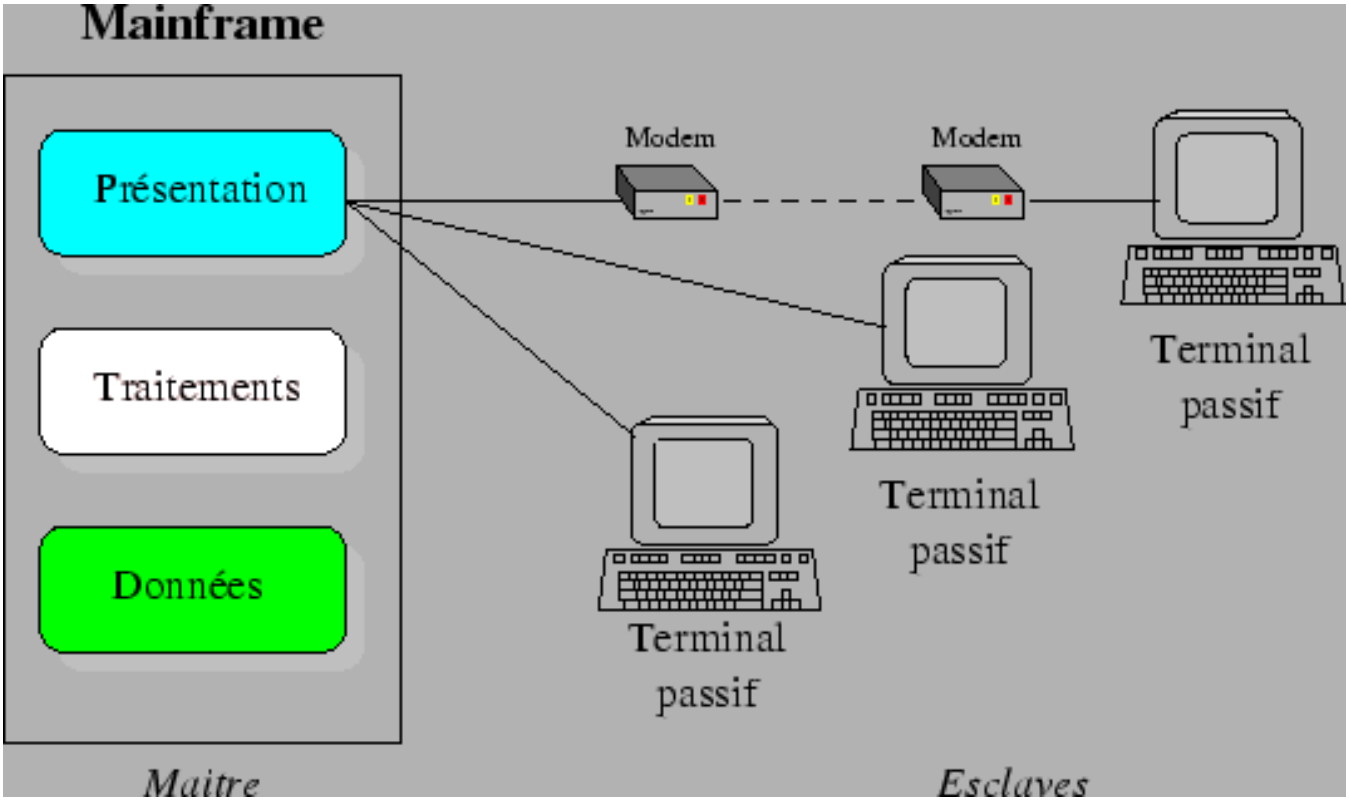

Figure 3.1: Architecture d'une application sur site central

Ce type d'organisation brille par sa grande facilité d'administration et sa haute disponibilité. Il bénéficie aujourd'hui d'une large palette d'outils de conception, de programmation et d'administration ayant atteint un niveau de maturité et de fiabilité leur permettant encore de soutenir la comparaison avec des solutions beaucoup plus modernes. De plus, la centralisation de la puissance sur une seule et même machine permet une utilisation optimale des ressources et se prête très bien à des traitements par lots[3.1] (en *batch*).

L'émergence des interfaces utilisateur de type *Windows - Macintosh*, démocratisées par les outils bureautiques sur micro-ordinateur, a fortement démodé les applications *mainframe*. Ces dernières exploitaient exclusivement une interface utilisateur en mode caractère jugée obsolète par les utilisateurs, qui réclamaient alors massivement une convergence entre la bureautique et le système d'information de l'entreprise.

Le ré-habillage d'application centralisée, ou *revamping*[3.2], permet de rajeunir ces dernières en leur faisant profiter d'une interface graphique moderne (*GUI*[3.3]), mais au prix d'une telle lourdeur[3.4] qu'il doit être considéré comme une simple étape vers une architecture plus moderne.

Les applications un tiers déployées

Avec l'arrivée dans l'entreprise des premiers PC en réseau, il est devenu possible de déployer une application un tiers sur plusieurs ordinateurs indépendants.

Ce type de programme est simple à concevoir et à mettre en oeuvre. Il existe pour cela de nombreux environnements de développement (*dBase, Ms Access, Lotus Approach, Paradox...*) qui sont souvent intégrés aux principales suites bureautiques. L'ergonomie des applications mises en oeuvre, basée sur celle des outils bureautiques, est très riche.

Ce type d'application peut être très satisfaisant pour répondre aux besoins d'un utilisateur isolé et sa mise en oeuvre dans un environnement multi-utilisateur est envisageable.

Dans ce contexte, plusieurs utilisateurs se partagent des fichiers de données stockés sur un serveur commun. Le moteur de base de données est exécuté indépendamment sur chaque poste client. La gestion des conflits d'accès aux données doit être prise en charge par chaque programme de façon indépendante, ce qui n'est pas toujours évident.

Lors de l'exécution d'une requête, l'intégralité des données nécessaires doit transiter sur le réseau et on arrive vite à saturer ce dernier. De plus, la cohabitation de plusieurs moteurs de base de données indépendants manipulant les mêmes données peut devenir assez instable. Il n'est pas rare de rencontrer des conflits lors de la consultation ou de la modification simultanée d'un même enregistrement par plusieurs utilisateurs. Ces conflits peuvent altérer l'intégrité des données. Enfin, il est difficile d'assurer la confidentialité des données.

Ce type de solution est donc à réserver à des applications non critiques exploitées par de petits groupes de travail (une dizaine de personnes au maximum).

Limitations

On le voit, les applications sur site central souffrent d'une interface utilisateur en mode caractères et la cohabitation d'applications micro exploitant des données communes n'est pas fiable au delà d'un certain nombre d'utilisateurs.

Il a donc fallu trouver une solution conciliant les avantages des deux premières :

- la fiabilité des solutions sur site central, qui gèrent les données de façon centralisée,
- l'interface utilisateur moderne des applications sur micro-ordinateurs.

Pour obtenir cette synthèse, il a fallu scinder les applications en plusieurs parties distinctes et coopérantes :

- gestion centralisée des données,
- gestion locale de l'interface utilisateur.

Ainsi est né le concept du client-serveur.

Next Up Previous Contents Index

suivant: Le schéma du Gartner **monter:** De l'informatique centralisée au **précédent:** Les trois niveaux d'abstraction **Table des matières** **Index**

Document rédigé par Rémi LEBLOND (remi.leblond@free.fr)

Next Up Previous Contents Index
suivant: L'architecture deux tiers **monter:** De l'informatique centralisée au **précédent:** L'architecture un tiers **Table des matières** **Index**

Le schéma du Gartner Group

Le Gartner Group, un cabinet de consultants américain, a publié un schéma des différents types de client-serveur existants.

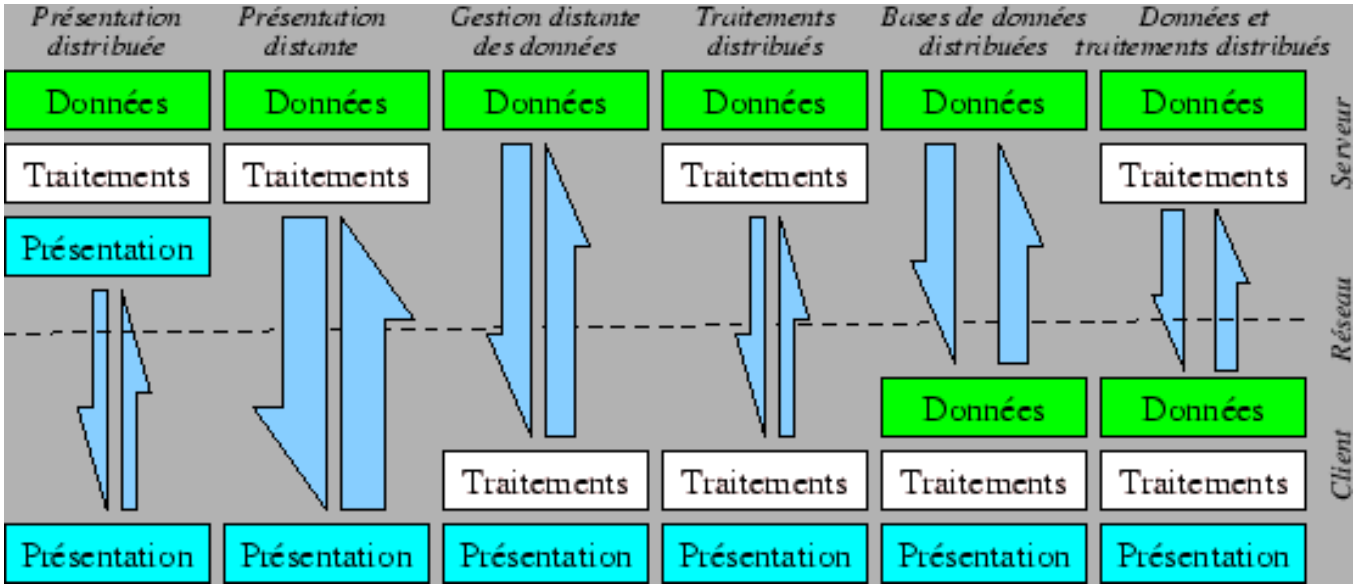

Figure 4.1: Les différents types de client-serveur selon la classification du Gartner Group [LEF94] [JFG99]

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 coeur 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[4.1], 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[4.2] centralisé.

   Cette architecture, de part sa souplesse, s'adapte très bien aux applications de type *infocentre*, 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.

Les solutions que nous venons de détailler sont indépendantes les unes des autres et, de ce fait, combinables à volonté.

Prises individuellement, on peut dire que les deux premières solutions proposées ne correspondent

pas à des applications client-serveur. En effet, la présentation distribuée comme l'affichage distant, dans l'application qu'en fait X-Window, ne considèrent pas le client en temps que tel et ce dernier reste dans une position d'esclave par rapport au serveur.

La première solution à mettre en oeuvre une relation client-serveur se retrouve au troisième niveau et correspond au client-serveur de données. Ce type d'application, encore appelé client-serveur de première génération, met en oeuvre une architecture deux-tiers.

Nous allons maintenant nous pencher plus en détails sur ce type d'application.

Next Up Previous Contents Index

suivant: L'architecture deux tiers **monter:** De l'informatique centralisée au **précédent:** L'architecture un tiers **Table des matières** **Index**

Document rédigé par Rémi LEBLOND (remi.leblond@free.fr)

Next Up Previous Contents Index
suivant: Les architectures distribuées **monter:** De l'informatique centralisée au **précédent:** Le schéma du Gartner
Table des matières **Index**

Sous-sections

L'architecture deux tiers

Présentation

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 l'application de gestion fonctionnant sous Ms-Windows 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[5.1], 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[5.2].

Le dialogue entre client et serveur se résume donc à l'envoi de requêtes et au retour des données correspondant aux requêtes. Ce dialogue nécessite l'instauration d'une communication entre client et serveur. Nous allons étudier de quoi elle se compose.

Le dialogue client-serveur

Le modèle client-serveur met en oeuvre une conversation entre deux programmes que l'on peut opposer à l'échange figé ``maître-esclave'' qu'entretiennent les applications sur site central avec leurs terminaux passifs [LEF94].

Dans une conversation client-serveur, on distingue donc les deux parties suivantes :

- **Le client**, c'est le programme qui provoque le dialogue,
- **Le serveur**, c'est le programme qui se contente de répondre au client.

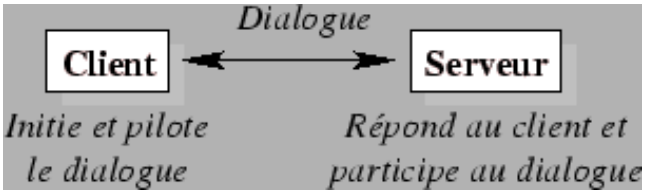

Figure 5.1:A la base du dialogue, un besoin du client
[LEF94]

Le client provoque l'établissement d'une conversation afin de d'obtenir des données ou un résultat de la part du serveur.

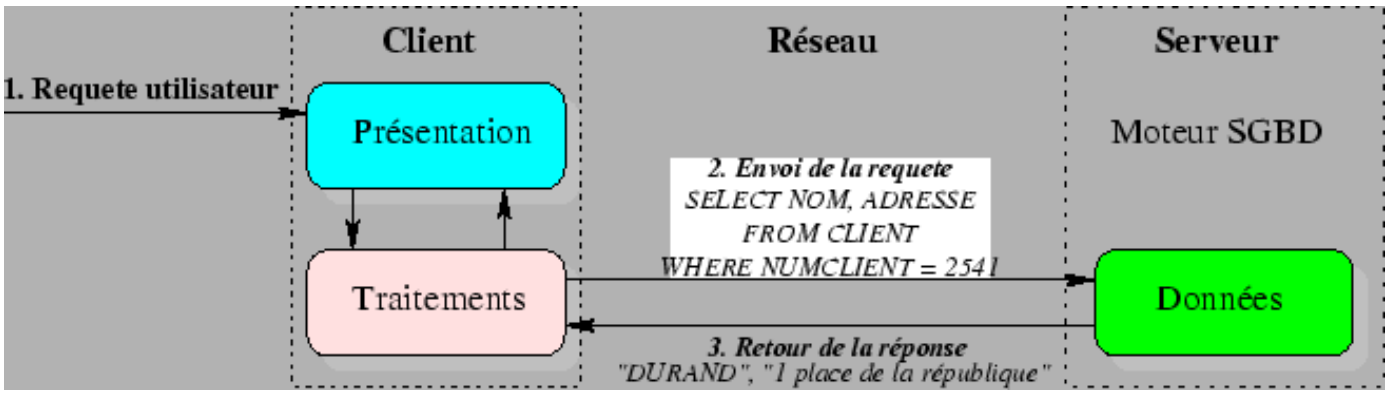

Figure 5.2: Accès aux données en mode deux tiers

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*.

Le Middleware

Définition

On appelle middleware, 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[5.3] 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.

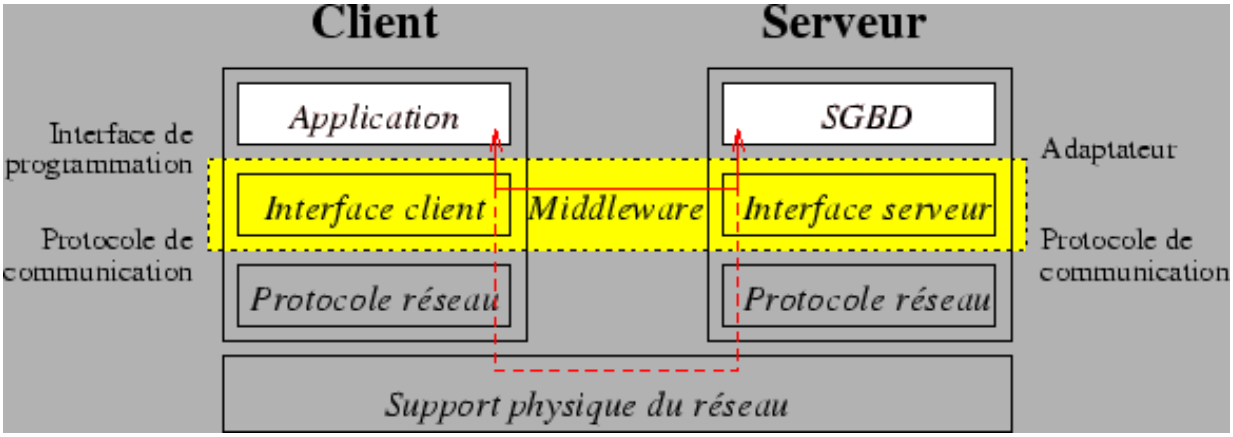

Figure 5.3: Positionnement du middleware entre client et serveur

Les services rendus

Un middleware est susceptible de rendre les services suivants [FIA96b]:

- **Conversion** : Service utilisé pour la communication entre machines mettant en oeuvre des formats de données différents, elle est prise en charge par la FAP5.4,
- **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, cette fonction doit faire appel aux services d'un annuaire.
- **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 extrêmement complexe et rigide.

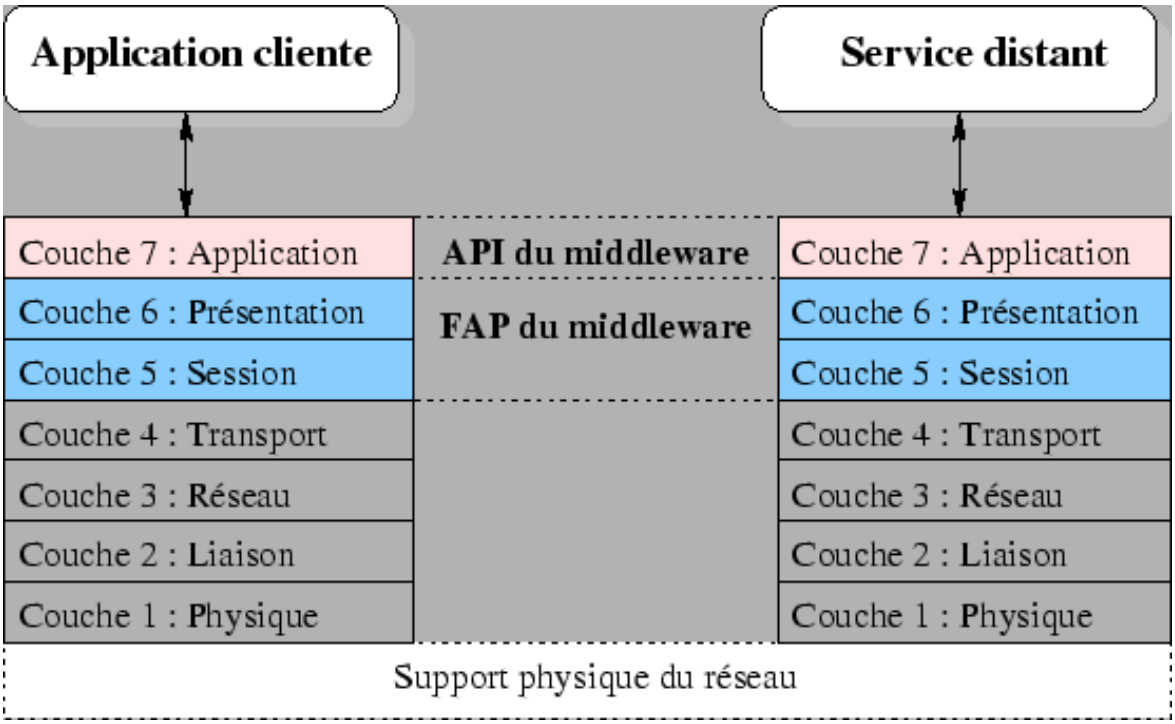

Figure 5.4: Le middleware par rapport au modèle OSI (Open Systems Interconnection)

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**[5.5] : Interface standardisée isolant le client du serveur de données. C'est l'implémentation par Microsoft du standard CLI[5.6] 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**[5.7] : Permet l'appel à des procédures distantes depuis une application.

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

Les middleware proposés par les fournisseurs de SGBD sont très performants et permettent de tirer profit de l'ensemble des fonctionnalités du serveur de données pour lequel ils ont été conçus. Par contre, ils ne permettent pas, le plus souvent, l'accès à d'autres sources de données.

Pour certaines applications devant accéder à des services hétérogènes, il est parfois nécessaire de combiner plusieurs middlewares. Dans ce cas, le poste client doit connaître et mettre en oeuvre plusieurs IPC[5.8], on en vient à la notion de client lourd.

Limites du client-serveur deux tiers : Le client lourd

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, ou *fat client*.

L'architecture client-serveur de première génération s'est heurtée à ce constat à l'heure des premiers bilans :

- 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,
- la conversation entre client et serveur est assez bruyante et s'adapte mal à des bandes passantes étroites. De ce fait, ce type d'application est souvent cantonné au réseau local de l'entreprise,
- 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.

Next Up Previous Contents Index

suivant: Les architectures distribuées **monter:** De l'informatique centralisée au **précédent:** Le schéma du Gartner

Table des matières **Index**

Document rédigé par Rémi LEBLOND (remi.leblond@free.fr)

Next Up Previous Contents Index

suivant: L'architecture trois tiers **monter:** Vers une architecture n-tiers **précédent:** L'architecture deux tiers **Table des matières** **Index**

Les architectures distribuées

Sous-sections

Document rédigé par Rémi LEBLOND (remi.leblond@free.fr)

Next Up Previous Contents Index
suivant: Les architectures n-tiers **monter:** Les architectures distribuées **précédent:** Les architectures distribuées **Table des matières** **Index**

Sous-sections

L'architecture trois tiers

Objectifs

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.

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.

Les premières tentatives

Les premières tentatives de mise en oeuvre d'architecture trois tiers proposaient l'introduction d'un serveur d'application centralisé exploité par les postes clients à l'aide dialogue RPC[6.1] propriétaire.

Les mécanismes nécessaires à la mise en place de telles solutions existent depuis longtemps :

- UNIX propose un mécanisme de RPC intégré au système NFS,
- NetDDE, puis DCOM de Microsoft permettent l'instauration d'un dialogue RPC entre client et serveur,
- certains environnements de développement facilitent la répartition des traitements entre le poste client et le serveur,
- des solutions propriétaires comme ForT ou Implicite permettent l'exploitation d'un serveur d'application par des clients simplement équipés d'un environnement d'exécution (runtime).

Cependant, l'application de ces technologies dans une architecture trois tiers est complexe et demande des compétences très pointues, du fait du manque de standards. Malgré ses avantages techniques évidents, ce type d'application fut souvent jugé trop coûteux à mettre en place.

Ce premier essai s'est soldé par un échec dû en grande partie à l'absence de standard et à la complexité des mécanismes mis en oeuvre.

Le serveur de transaction

Ce type de serveur héberge un moniteur transactionnel qui s'occupe de la mise en relation du client avec un ensemble de serveurs de données. Le serveur transactionnel, interrogé en utilisant une API unique, masque au client la complexité de l'organisation des serveurs de données.

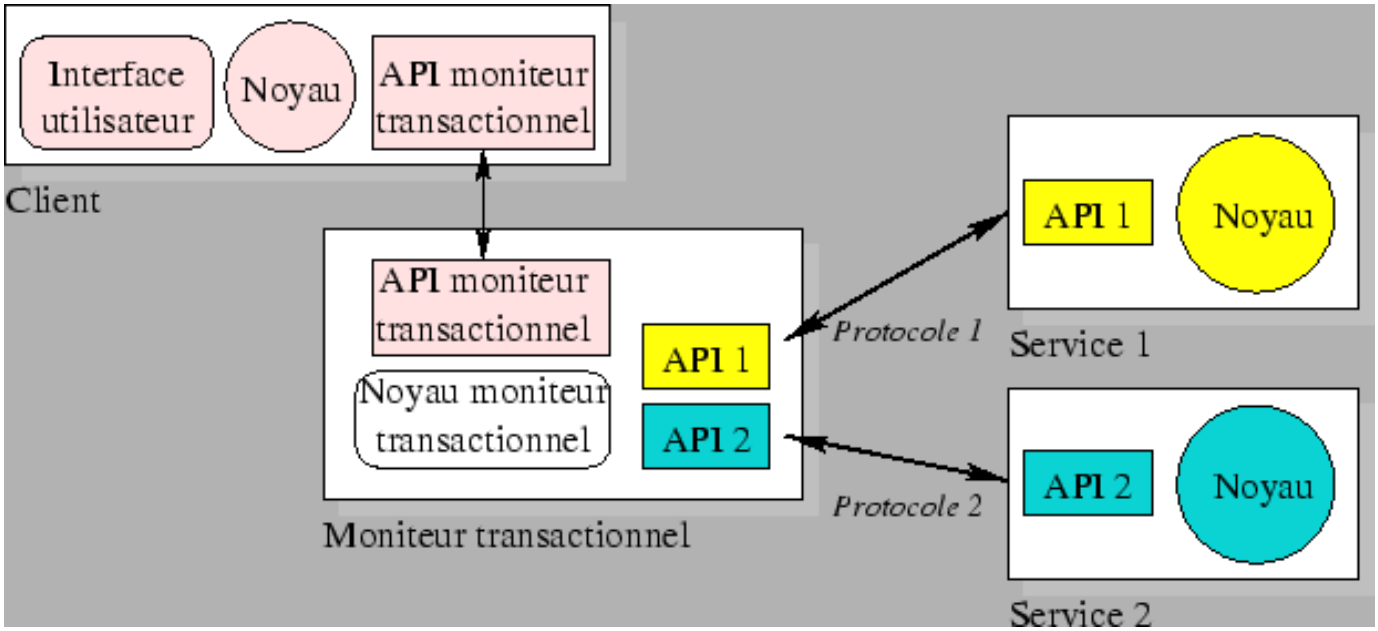

Figure 6.1: Fonctionnement d'un moniteur transactionnel [EL97]

Le moniteur transactionnel permet de garantir que toutes les transactions vérifient la règle ACID :

- **Atomicité** : La transaction ne peut être partiellement effectuée,
- **Cohérence** : Une transaction fait passer la base d'un état cohérent à un autre,
- **Isolation** : Une transaction n'est pas affectée par le résultat des autres transactions,
- **Durée** : Les modifications dues à une transaction sont durablement garanties.

En fait, soit une transaction a définitivement eu lieu, soit elle n'a jamais existé.

La notion de serveur transactionnel est citée brièvement ici car elle correspond, à mon avis, à une architecture trois tiers puisqu'une partie des traitements applicatifs est centralisée.

La révolution Internet

Introduction

S'il est un phénomène qui a marqué le monde de l'informatique ces dernières années, c'est bien celui d'Internet.

Ce réseau mondial, créé en 1969 par l'armée américaine, puis utilisé par les chercheurs et autres scientifiques, a connu une croissance phénoménale auprès du grand public avec l'introduction du *World Wide Web*[6.2] en 1989. Ce dernier permet de publier simplement des informations richement mises en forme et pouvant même, par la suite, contenir des données multimédia.

La véritable révolution du WWW réside dans son caractère universel, rendu possible par l'utilisation de standards reconnus.

Les standards d'Internet

L'universalité du Web repose sur des standards simples et admis par tous :

- HTML, pour la description des pages disponibles sur le Web,
- HTTP, pour la communication entre navigateur et serveur Web,
- TCP/IP, le protocole réseau largement utilisé par les systèmes Unix,
- CGI[6.3], l'interface qui permet de déclencher à distance des traitements sur les serveurs Web.

HTML (HyperText Markup Langage)

HTML est le langage de description de pages hypertexte utilisé par le World Wide Web, il est issu de SGML[6.4].

Une page HTML est composée de son contenu propre (du texte plus ou moins richement mis en forme) et de références statiques vers d'autres sources d'informations (images, liens vers d'autres documents...).

La consultation d'une page HTML n'implique donc que très rarement le chargement du seul fichier décrivant la page, il s'accompagne en général de celui de nombreux fichiers annexes. Ces chargements mettent en oeuvre le protocole HTTP.

HTTP

HTTP est un protocole réseau applicatif (dernier niveau du modèle OSI) sans connexion utilisé pour l'échange des données sur le Web. En fait, une connexion HTTP est créée pour chaque requête et ne dure que pendant l'exécution de cette dernière.

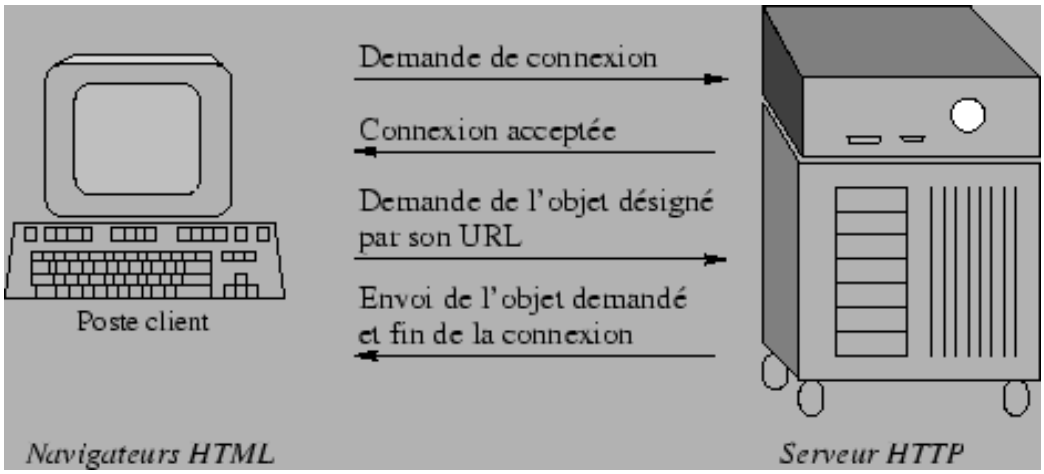

Figure 6.2: Le fonctionnement de base de HTTP[LEF97]

Le protocole HTTP, en temps que protocole réseau applicatif, s'appuie sur un protocole de transport indépendant qui, dans le cadre d'Internet, est TCP/IP[6.5].

TCP/IP (Transmission Control Protocol / Internet Protocol)

TCP/IP est un protocole réseau de niveau trois et quatre (réseau et transport) qui s'est largement imposé sur les systèmes Unix, puis sur Internet, et fait aujourd'hui figure de standard universel.

Si le fonctionnement de TCP/IP s'adapte bien à la topologie et à la qualité de service du réseau Internet, on ne peut en dire autant du mariage avec le protocole HTTP. En effet, TCP/IP utilise un mécanisme de ``démarrage lent'' afin d'éviter les engorgements du réseau. Ce mécanisme permet à la machine émettrice d'ouvrir progressivement une fenêtre de congestion en doublant le nombre de paquets émis à chaque aller-retour. En général, la courte durée des échanges HTTP ne permet pas à la fenêtre de congestion d'atteindre la largeur de bande fournie par le réseau local [LEF97].

CGI (Common Gateway Interface)

CGI est un standard permettant d'écrire des extensions compatibles avec la grande majorité des serveurs HTTP. Ces extensions permettent l'exécution d'une action par le serveur à la demande d'un client.

Ce mécanisme relativement simple, voire même rustique, entraîne l'exécution d'un processus propre à chaque invocation, ce qui est très consommateur de ressources.

De ce fait, des extensions comme ISAPI[6.6], NSAPI[6.7] ou les servlets Java sont souvent préférés au standard CGI.

Adaptation à l'entreprise : Intranet

Aucun des mécanismes mis en oeuvre par Internet n'est exempt de défaut et il est relativement simple de trouver plus performant. En fait, la force de l'ensemble repose essentiellement dans son universalité.

La notion d'Intranet est née de l'intégration des principes d'Internet et des technologies déployées dans l'entreprise :

- on utilise le réseau local de l'entreprise,
- les données sont toujours gérées par un SGBD,
- les mécanismes utilisés pour interroger le SGBD sont toujours les mêmes.

Répartition des traitements

L'architecture trois tiers, encore appelée client-serveur de deuxième génération ou client-serveur distribué, sépare l'application en trois niveaux de service distincts :

- **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.

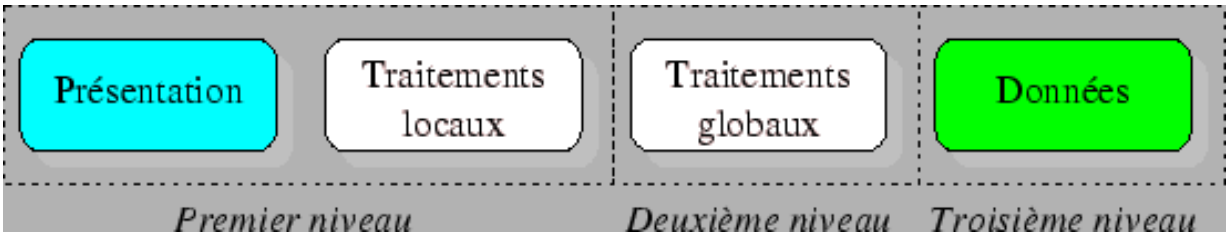

Figure 6.3: Le découpage d'une application en pavés fonctionnels indépendants

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 resources 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 le cadre d'un Intranet, 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.

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. On l'appelle **circuit froid** et n'est composé que de standards (principalement HTML et HTTP). Le serveur Web tient le rôle de ``façade HTTP'',

- le deuxième tronçon permet la collecte des données, il est aussi appelé **circuit chaud**. 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 impacter la configuration des postes clients.

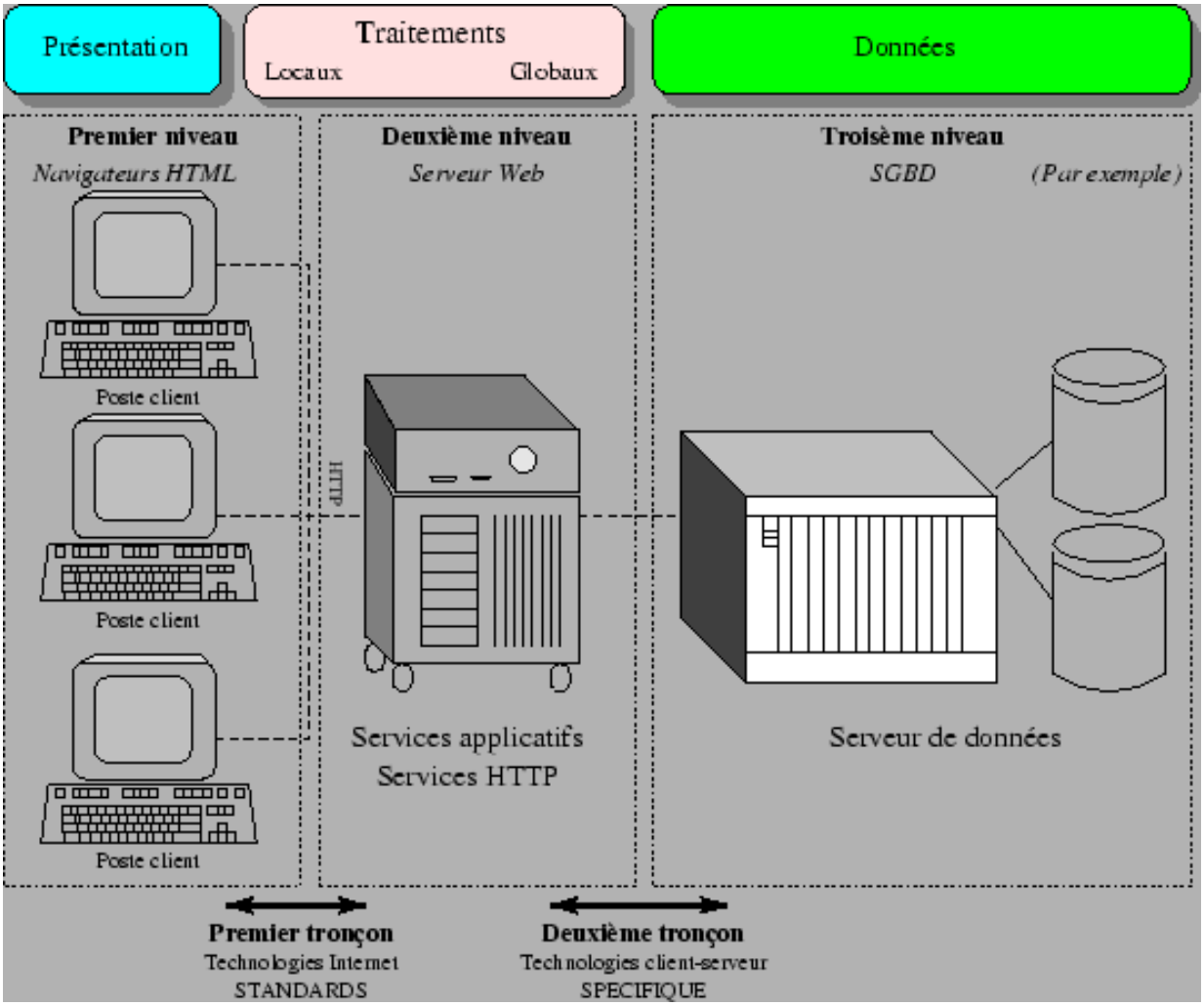

Figure 6.4:Répartition des couches applicatives dans une architecture trois tiers

Le client léger

Présentation

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. Il est souvent constitué d'un simple navigateur Internet.

Le poste client ne communique qu'avec la façade HTTP de l'application et ne dispose d'aucune connaissance des traitements applicatifs ou de la structure des données exploitées. Les évolutions de l'application sont donc possibles sans nécessiter de modification de la partie cliente.

Par exemple, un internaute se connectant à *www.yahoo.fr* pour effectuer une recherche provoque, sans même le savoir, l'exécution de traitements sur le serveur. Si ces traitements évoluent, ce qui doit arriver relativement souvent, le client continuera à utiliser le service sans se rendre compte des changements (sauf s'ils lui apportent de nouveaux services).

De plus, ce même internaute peut se connecter au serveur en utilisant tout type de poste client disposant d'un navigateur compatible HTML (PC sous Windows, Macintosh, Station Unix, WebPhone...).

On voit donc ici la force des architectures trois tiers par rapport au client-serveur de première génération. Le déploiement est immédiat, les évolutions peuvent être transparentes pour l'utilisateur et les caractéristiques du poste client sont libres.

Ergonomie

Utilisation d'HTML

Les pages HTML, même avec l'aide de langages de script, sont loin d'atteindre les possibilités offertes par l'environnement Windows :

- le multi-fenêtrage n'est pas facile à mettre en oeuvre,
- le déroulement de l'application doit se faire séquentiellement,
- les pages affichées sont relativement statiques,
- le développement multi-plateforme peut être contraignant[6.8].
- l'ergonomie de l'application est limitée aux possibilités du navigateur.

Pour ces raisons, certaines applications ne sont pas réalisables dans une architecture de type Intranet (infocentres, applications bureautiques...).

Dans les autres cas, l'appauvrissement de l'interface utilisateur est souvent le gage d'une plus grande facilité de prise en main de l'application.

Il est rare en effet de devoir suivre une formation pour apprendre à se servir d'un site Web particulier. La plupart du temps, une formation générale à l'ergonomie des sites Web suffit.

Cette perte de richesse peut donc se transformer en avantage, à condition de respecter une charte graphique et ergonomique cohérente pour toutes les applications Intranet d'une entreprise.

Il est possible d'aller au delà des possibilités offertes par le langage HTML en y introduisant des applets Java ou des contrôles ActiveX. Nous ne parlerons pas ici des modules d'extension du navigateur, encore appelés *plug-in*, qui étaient en vogue avant l'arrivée des solutions Java et ActiveX, car cette solution est trop contraignante à déployer.

Utilisation de Java

Java est un langage de développement orienté objet et multi-plateforme introduit par Sun en 1995. Il s'agit de l'adaptation à Internet du langage OAK[6.9], initialement étudié pour les environnements de petite taille. Il permet, entre autre, d'écrire de petites applications, appelées *applet*, pouvant être intégrées à des pages HTML pour en enrichir le contenu.

Le caractère multi-plateforme de Java se prête bien à une utilisation sur Internet, où les caractéristiques des postes clients ne sont pas maîtrisées. Il repose sur l'utilisation d'un interpréteur de pseudo-code Java, pompeusement appelé ``machine virtuelle''.

Les programmes Java ne sont pas compilés en code machine, mais en pseudo-code Java uniquement compréhensible par la machine virtuelle. Cette dernière interprète le code et se charge de lier les modules au moment de l'exécution, en les téléchargeant si nécessaire. La liaison dynamique des modules au moment de l'exécution permet d'optimiser le trafic réseau, puisqu'on ne charge que le strict nécessaire.

Pour ces raisons, un programme Java s'exécute plus lentement que son équivalent compilé. La compilation à la volée des programmes Java et plus encore, la technologie d'optimisation dynamique de code *HotSpot* de Sun, permettent de réduire l'écart de performance avec des programmes compilés.

Java propose aujourd'hui une large palette de composants graphiques[6.10] et multimédia qui permettent d'atteindre la richesse fonctionnelle des applications Windows.

Une applet Java est aussi capable d'exploiter directement un serveur de données en utilisant JDBC[6.11] ou de faire appel à des procédures distantes en utilisant RMI[6.12] ou CORBA[6.13]. Nous verrons ces mécanismes par la suite.

Utilisation d'ActiveX

Quand Microsoft entend parler de ``multi-plateforme'', il comprend ``fonctionnement en dehors de Windows'' et, par extension directe, ``danger''. De ce fait, il ne pouvait rester sans réponse devant le phénomène Java.

Après avoir essayé, sans succès, de confiner Java dans le rôle d'un simple langage de programmation dépendant de Windows[6.14], Microsoft appuie sa politique Intranet sur sa technologie ActiveX.

ActiveX est une adaptation Internet des composants OCX constituant la clef de voûte de l'architecture DNA[6.15]. Les composants ActiveX exploitent les possibilités de Windows et sont donc capables d'offrir une grande richesse fonctionnelle aux applications qui les utilisent.

Ils peuvent être intégrés à une page HTML mais, à la différence des applets Java, ils persistent sur le poste client après utilisation. Ils ne sont alors remplacés que si une nouvelle version du composant est utilisée. Ce mécanisme permet d'optimiser les transferts de composants sur le réseau mais rappelle grandement l'installation locale d'application, avec ses effets négatifs sur l'alourdissement du poste client.

Les composants ActiveX peuvent communiquer entre eux en utilisant la technologie DCOM[6.16] ou avec des bases de données, en utilisant ODBC.

En fait, l'utilisation des composants ActiveX rend les applications dépendantes de la plateforme Windows. On se prive alors des possibilités offertes par d'autres systèmes d'exploitation ou de postes clients plus simples et donc, de la possibilité de faire baisser le coût de possession des postes clients.

En prenant du recul, on s'aperçoit que l'utilisation d'objets ActiveX dans une application Intranet fait perdre une grande partie de l'intérêt de cette architecture et rappelle fortement le client-serveur deux tiers.

Exemples de clients légers

Le client léger peut prendre plusieurs formes :

- un poste Windows équipé d'un navigateur HTML,
- une station réseau du type NC[6.17]. Contrairement à un simple terminal X, ce type de station prend en charge des traitements locaux (affichage, contrôle de saisie, mise en forme de donnée...). Ce type de poste client se démarque par un coût de possession très largement inférieur à celui d'un PC sous Windows.
- un terminal Windows (NetPC[6.18]) correspondant à un PC minimal.

Le client léger à souvent été présenté comme le successeur du PC sous Windows. En fait, le client léger ne prétend pas remplacer tous les PC dans l'entreprise, il se présente plutôt comme une alternative à ce dernier pour certains besoins particuliers et cohabite très bien avec l'existant.

Le service applicatif

Présentation

Dans une architecture trois tiers, la logique applicative est prise en charge par le serveur HTTP. Ce dernier se retrouve dans la position du poste client d'une application deux tiers et les échanges avec le serveur de données mettent en oeuvre les mécanismes déjà vus dans ce type d'application.

Les développements mis en oeuvre sur le serveur HTTP doivent être conçus spécifiquement. Il peuvent mettre en oeuvre :

- CGI : le mécanisme standard et reconnu de tous, mais grand consommateur de ressources,
- NSAPI ou ISAPI : les API de Netscape et Microsoft permettant l'écriture d'applications multi-thread[6.19] intégrées au serveur HTTP,
- les scripts serveur comme ASP (Active Server Page pour IIS) ou PHP (l'équivalent pour Apache) sont interprétés par le serveur pour générer des pages dynamiquement,
- les servlets Java : qui appliquent le mécanisme des applets aux traitements réalisés sur le serveur.

Gestion des transactions

Comme le protocole HTTP n'assure pas de gestion d'états, les applications transactionnelles doivent gérer elles-mêmes le contexte utilisateur afin de contrôler :

- le cheminement de l'utilisateur dans l'application,
- les actions et saisies de l'utilisateur,
- l'identité de l'utilisateur,
- la gestion des transactions.

Il existe différentes méthodes de gestion de contexte :

- attribution d'un identifiant à chaque étape de la transaction,
- stockage de l'ensemble du contexte sur le poste client.

Ces méthodes nécessitent le stockage d'informations sur le poste client :

- dans un cookie[6.20] déposé sur le poste client,
- dans une URL[6.21] longue,
- dans des variables cachées au sein de la page HTML.

Limitations

L'architecture trois tiers a corrigé les excès du client lourd en centralisant une grande partie de la logique applicative sur un serveur HTTP. Le poste client, qui ne prend à sa charge que la présentation et les contrôles de saisie, s'est trouvé ainsi soulagé et plus simple à gérer.

Par contre, le serveur HTTP constitue la pierre angulaire de l'architecture et se trouve souvent fortement sollicité et 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[6.22]. Le phénoméne fait penser à un retour de balancier.

Le juste équilibrage de la charge entre client et serveur semble atteint avec la génération suivante : les architectures n-tiers.

Next Up Previous Contents Index

suivant: Les architectures n-tiers **monter:** Les architectures distribuées **précédent:** Les architectures distribuées **Table des matières** **Index**

Document rédigé par Rémi LEBLOND (remi.leblond@free.fr)

Next Up Previous Contents Index
suivant: Index **monter:** Les architectures distribuées **précédent:** L'architecture trois tiers
Table des matières **Index**

Sous-sections

Les architectures n-tiers

Présentation

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 [CN98] :

- 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.

Que de niveaux...

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.

Le rôle de l'approche objet

L'approche objet

Les évolutions successives de l'informatique permettent de masquer la complexité des mécanismes mis en oeuvre derrière une approche de plus en plus conceptuelle.

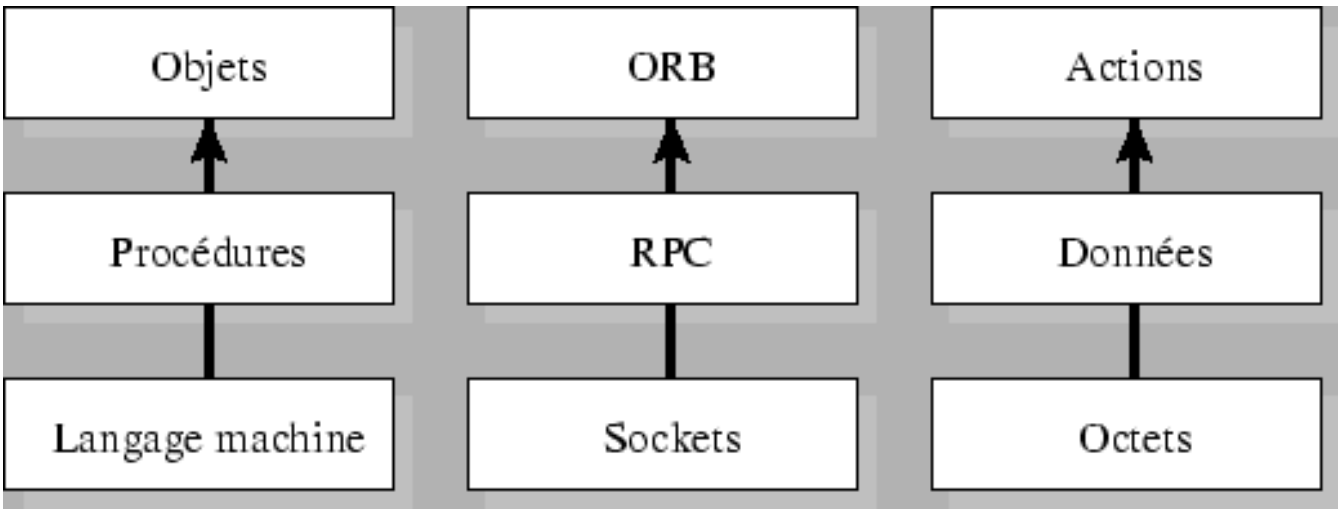

Figure 7.1: Evolution des technologies utilisées pour la mise en oeuvre d'applications distribuées[EL97]

Ainsi, les langages de programmation ont tout d'abord été très proches de la machine (langage machine de bas niveau), puis procéduraux et, enfin, orientés objets. Le succès du langage Java a véritablement popularisé ce mode de programmation.

Les protocoles réseau ont suivi le même type d'évolution. Ils furent d'abord très proches de la couche physique, avec les mécanismes de sockets orientés octets. Ensuite, la notion de RPC a permis de faire abstraction des protocoles de communication et, ainsi, a facilité la mise en place d'application client-serveur. Aujourd'hui, l'utilisation d'ORB[7.1] permet une totale transparence des appels distants et permet de manipuler un objet distant comme s'il était local. Le flux d'informations fut donc initialement constitué d'octets, puis de données et, enfin, de messages.

Les méthodes de conception orientées objet telles qu'UML ou OMT permettent une modélisation plus concrète des besoins et facilitent le passage de la conception à la réalisation.

Aucune de ces évolutions ne constitue en soit une révolution, mais elles rendent économiquement réalisables des architectures qui n'étaient jusqu'à présent que techniquement envisageables.

Les objets métier

Les Java Beans

Selon les spécifications de Sun, un Java Beans est un composant logiciel réutilisable qui peut être manipulé par un outil d'assemblage. Cette notion assez large englobe aussi bien un simple bouton qu'une application complète. Concrètement, les Java Beans sont des classes

Java utilisant des interfaces particulières.

Un Java Beans peut être utilisé indépendamment, sous la forme d'une simple applet, ou intégré à un développement Java. Il peut dévoiler son comportement à ses futurs utilisateurs à l'aide :

- des propriétés qu'il expose et rend accessible à l'aide d'accesseurs,
- des méthodes qu'il permet d'invoquer, comme tout objet Java,
- des événements qu'il peut générer pour avertir d'autres composants.

Le mécanisme d'introspection permet une analyse automatique du composant qui, ainsi, s'auto-décrit.

Les Java Beans proposent deux modes de fonctionnement :

- le mode de configuration permettant de paramètrer l'intégration du composant à l'aide d'un outil d'assemblage. Dans ce mode, le Java Beans propose un interface graphique de configuration,
- le mode d'exécution utilisé par l'application intégrant le Java Beans.

Les Java Beans sont gérés comme tout objet Java, notamment ce qui concerne son cycle de vie. Ils sont livrés sous forme de fichiers JAR[7.2].

Les EJB (Entreprise Java Beans)

Les Enterprise Java Beans sont des Java Beans destinés à s'exécuter côté serveur :

- ils ne comportent pas forcément de partie visible,
- ils prennent en charge des fonctions de sécurité, de gestion des transactions et d'état,
- ils peuvent communiquer avec des Java Beans côté client de façon indépendante du protocole (IIOP[7.3], RMI[7.4], DCOM[7.5]),
- ils s'exécutent dans un environnement ``Beanstalk'' s'appuyant sur l'API Java Server et offrant des fonctionnalités de gestion de la charge d'exécution, de la sécurité d'accès, de communication, d'accès aux services transactionnels.

Microsoft OLE-COM-ActiveX

Le modèle de communication COM permet de mettre en place une communication orientée objet entre des applications s'exécutant sur une même machine.

DCOM est une extension de ce modèle pour les architectures distribuées. Il repose sur le modèle DCE[7.6] défini par l'OSF[7.7] et met en oeuvre un serveur d'objets situé sur chaque

machine.

Les contrôles ActiveX, anciennement dénommés OCX, sont des composants logiciels basés sur le modèles COM. Ils peuvent être intégrés à des applications où à des documents sous Windows.

La communication entre objets

Principes

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[7.8] préconisée par l'OMG[7.9] et DCOM développée par Microsoft.

Ces middlewares sont constitués d'une série de mécanismes permettant à un ensemble de programmes d'interopérer 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 ORB[7.10] se chargeant d'aiguiller les messages vers les objets (locaux ou distants).

Il est possible d'encapsuler des applications ``non objet'' existantes pour les rendre accessibles via le middleware objet. Ainsi, on préserve l'existant sans alourdir les nouveaux développements, qui ne voient que l'interface de l'application encapsulée. La vision du système s'effectue ainsi de manière unifiée, chaque composant étant accessible via le middleware. Un client peut désormais accéder de la même façon à un EJB ou à une transaction sur mainframe.

L'appel de procédure distantes (RMI)

RMI permet à une application Java, s'exécutant sur une machine virtuelle, d'invoquer les méthodes d'un objet hébergé par une machine distante. Pour cela, le client utilise une représentation locale de l'interface de l'objet serveur. Cette représentation locale est appelée *stub* et représente l'interface de l'objet serveur, appelée *skeleton*.

Un objet distribué se caractérise par son interface et son adresse (URL). La mise en relation des objets est assurée par un serveur de noms.

Le modèle CORBA

Le système CORBA permet, au travers du protocole IIOP, l'utilisation d'objets structurés dans un environnement hétérogène. Cette communication, orchestrée par l'ORB, est indépendante des contraintes systèmes des différentes plates-formes matérielles.

Une application accède à un objet distant en utilisant une télécommande locale, appelée *proxy*. Ce proxy lui permet de déclencher les méthodes de l'objet distant à l'aide de primitives décrites avec le langage IDL[7.11].

L'OMG effectue toute une série de recommandations connues sous le nom de CORBA services visant à proposer des interfaces génériques pour chaque type de service usuel (nommage, transaction, cycle de vie, ...).

Enfin une vision cohérente du système d'information

Avec les applications n-tiers, on dispose enfin d'une vision cohérente du système d'information. Tous les services sont représentés sous la forme d'objets interchangeables qu'il est possible d'implanter librement, en fonction des besoins.

Le potentiel de ce type d'application est très important et permettrait enfin l'utilisation d'objets métiers réutilisables.

Serions-nous en présence de l'architecture parfaite, nous permettant de mettre en oeuvre les applications les plus ambitieuses au moindre coût et de les faire évoluer librement en fonction des besoins ?

Si, sur le papier, on pourrait répondre par l'affirmative, l'expérience des précédentes ``révolutions'' de l'informatique nous pousse à plus de modération. L'architecture n-tiers propose effectivement un potentiel énorme, reste à voir comment il sera utilisé, à quel coût et, surtout, comment sera gérée la transition depuis les environnements actuels.

Next Up Previous Contents Index

suivant: Index **monter:** Les architectures distribuées **précédent:** L'architecture trois tiers

Table des matières **Index**

Document rédigé par Rémi LEBLOND (remi.leblond@free.fr)

Next Up Previous Contents Index

suivant: À propos de ce **monter:** Vers une architecture n-tiers **précédent:** Index **Table des matières** **Index**

Bibliographie

19999

Micromax Information Services Ltd. 1999.
N-Tiers Background Articles.
www.n-tiers.com, Octobre 1999.

CN98

Frédéric NAJMAN Cédric NICOLAS, Christophe AVARE.
Java client-serveur.
Eyrolles, 1998.

EL97

Thierry RUIZ Emmanuel LIGNE.
Corba : Objectifs et architecture.
www.etu.info.unicaen.fr/ cliquet/dess/corba/doc-fr/node3.html, Avril 1997.

FAS99

Will FASTIE.
Du client-serveur aux architectures n-tiers.
PC Expert, 1999.
Adapté de l'américain par Laurent DELATTRE.

FIA96a

Bernard FIAT.
CORBA Objectifs et architecture.
Computer channel, Septembre 1996.
En collaboration avec l'AFPA.

FIA96b

Bernard FIAT.
Le client-serveur (1) : Concepts de base.
Computer channel, Septembre 1996.
En collaboration avec l'AFPA.

Ing98

Sql Ingenierie.
Formation Intranet.
SQLI, 1998.

INM93
William-H INMON.
Le développement des applications clients-serveurs.
Masson, 1993.

JFG99
Philippe USCLADE Jean-François GOGLIN.
Du client-serveur au web-serveur.
Hermès Sciences, 1999.

LEF94
Alain LEFEBVRE.
L'Architecture client-serveur.
Armand COLIN, 1994.

LEF97
Alain LEFEBVRE.
Intranet client-serveur universel.
Eyrolles, 1997.

MAR99a
Daniel MARTIN.
Architecture des applications réparties.
http://worldserver2.oleane.com/, Octobre 1999.
/dmartin/Architecture applications reparties.htm.

MAR99b
Daniel MARTIN.
Les services d'un middleware.
http://worldserver2.oleane.com/, Octobre 1999.
dmartin/Middleware.

MOR88
Pierre MORVAN.
Dictionnaire de l'Informatique.
Larousse, 1988.

Tri96

Bud Tribble.
Java Computing, l'informatique Java.
Sun Microsystem, octobre 1996.

Document rédigé par Rémi LEBLOND (remi.leblond@free.fr)

Next Up Previous Contents Index
monter: Vers une architecture n-tiers **précédent:** Bibliographie **Table des matières**
Index

À propos de ce document...

Vers une architecture n-tiers

This document was generated using the **LaTeX**2HTML translator Version 99.2beta8 (1.42)

The command line arguments were:
latex2html `-transparent -image_type gif -local_icons -split 3 -address '`

`Document rédigé par Rémi LEBLOND (remi.leblond@free.fr)'`
`probatoire.tex`

The translation was initiated by remi on 2001-05-10

Document rédigé par Rémi LEBLOND (remi.leblond@free.fr)

Next Up Previous Contents

suivant: Bibliographie **monter:** Vers une architecture n-tiers **précédent:** Les architectures n-tiers **Table des matières**

Index

Document rédigé par Rémi LEBLOND (remi.leblond@free.fr)