Cours 1 Java EE

Introduction aux architectures des systèmes d'information

Dr. Hafedh Nefzi / 2IM

1. Introduction

Un système réparti permet à des utilisateurs, situés à des endroits différents, de coopérer et de mettre en commun leurs ressources. L'accès à distance aux ressources requiert un modèle d'interaction entre les utilisateurs et les ressources réparties. Le partage des ressources est intéressant parce qu'il permet, entre autres, de réduire les coûts, d'échanger de l'information, de diffuser rapidement des données et de collaborer à distance. Un gestionnaire de ressources est un logiciel responsable de l'administration d'un type de ressources. Il possède une interface de télécommunications au travers de laquelle s'effectuent l'accès et la mise à jour des ressources par les utilisateurs. Le gestionnaire met également en œuvre les politiques d'accès propres à chaque type de ressources (ex.: les imprimantes).

Les trois concepts de base des systèmes répartis (ressource, gestionnaire de ressources et utilisateur de ressources) peuvent être structurés et mis en relation suivant deux grands modèles, le modèle client-serveur et le modèle des composants répartis.

2. Les systèmes d'information

Le système d'information (SI) est un élément central d'une entreprise ou d'une organisation. Il permet aux différents acteurs de véhiculer des informations et de communiquer grâce à un ensemble de ressources matérielles, humaines et logicielles. Un SI permet de collecter, stocker, traiter et diffuser l'information à la bonne personne et au bon moment sous le format approprié.

- Collecter : c'est à partir de là que naît la donnée, qu'on acquière les informations provenant de l'environnement interne ou externe à l'entreprise.
- Stocker : dès que l'information est acquise, le système d'information la conserve. Elle doit pouvoir être disponible et doit pouvoir être conservée dans le temps.
- Traiter : cette phase permet de transformer l'information et choisir le support adapté pour traiter l'information.
- Diffuser : le SI transmet ensuite l'information dans son environnement interne ou externe.

En raison de l'importance cruciale des systèmes d'information dans le fonctionnement des entreprises ainsi que l'évolution des applications informatiques dans plusieurs domaines d'activité : comptabilité, administration, etc. Il semble nécessaire d'obtenir une seule et même application qui gère l'ensemble du système d'informations dans l'ensemble des secteurs d'activité d'une entreprise.

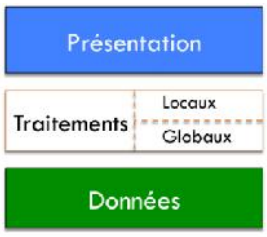
Une telle application peut être découpée en trois niveaux d'abstraction distincts : Présentation, Logique applicative et Données.

La couche de présentation, encore appelée IHM (Interface Homme Machine), permet l'interaction de l'application avec l'utilisateur, elle gère les saisies au clavier et à la souris et la présentation des informations à l'écran.

La logique applicative décrit les traitements à réaliser par l'application pour répondre aux besoins des utilisateurs. Ces traitements peuvent être divisés en 2 types : les traitements locaux qui correspondent aux

contrôles effectués au niveau du dialogue avec l'IHM (formulaires, champs, boutons radio…) et les traitements globaux qui correspondent aux règles de l'application, appelées aussi logique métier (Business Logic).

L'accès aux données, regroupe l'ensemble des mécanismes permettant la gestion des informations stockées par l'application telles que la définition de données, la manipulation de données, la sécurité de données, la gestion de transactions, etc.



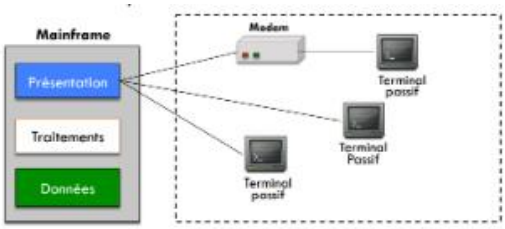
3. Architecture 1-tiers

Dans une approche de type 1-tiers, les trois couches sont fortement et intimement liées, et s'exécutent sur la même machine. Dans ce cas, on ne peut pas parler d'architecture client-serveur mais d'informatique centralisée.

Dans un contexte simple utilisateur, la question ne se pose pas, mais dans un contexte multiutilisateurs, on peut voir apparaître deux types d'architectures mettant en œuvre des applications 1-tiers : des applications sur site central et des applications réparties sur des machines indépendantes communiquant par partage de fichiers.

a) Applications sur site central

Les utilisateurs se connectent aux applications exécutées par le serveur central (mainframe) à l'aide de terminaux passifs, le serveur central prend en charge la gestion des données et des traitements, y compris l'affichage qui est transmis sur des terminaux passifs.



b) Applications 1-tiers déployées:

Dans ce type d'architecture, les utilisateurs se partagent des fichiers de données stockés sur un serveur commun, où la gestion de données et des conflits d'accès à ces données sont prises en charge par chaque client de façon indépendante.

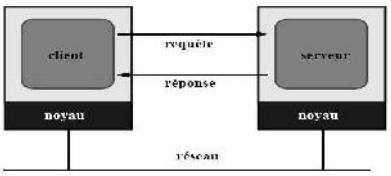
4. Architectures Client/Serveur

L'architecture client/serveur désigne un mode de communication entre plusieurs ordinateurs d'un réseau qui distingue un ou plusieurs postes clients du serveur : dans ce modèle, le dialogue entre le client et le serveur se fait par échange de messages plutôt que par mémoire partagée.

Pour le client, un serveur est une boite noire. Seuls les services rendus par le serveur sont connus du client par leurs noms ; les paramètres à fournir et les paramètres qui lui seront rendus après exécution du service.

Le dialogue avec le serveur est à l'initiative du client, il est réalisé par échange de deux messages :

- Une requête (demande) du client pour l'exécution d'un service par le serveur.
- Une réponse envoyée par le serveur et qui contient le résultat du service.



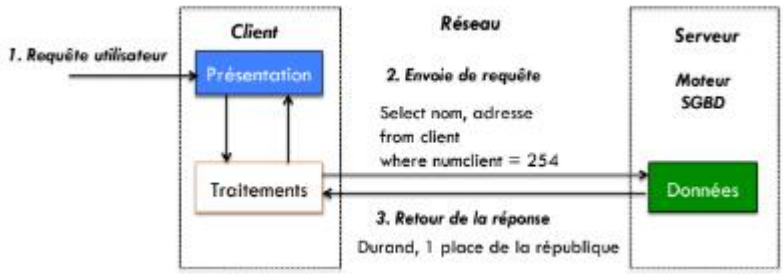
Modèle Client/Serveur

a) Architecture client/serveur traditionnelle

Le modèle d'architecture client/serveur le plus répandu est celui de 2-tiers, où l'application cliente (interface homme/machine et traitements) est située sur le poste client. Le lien entre le client et le serveur (serveur de BD par exemple) est direct. L'application cliente accède aux données de la base via des requêtes SQL (requêtes de sélection, d'ajout, de modification ou encore de suppression de données, etc.). Peu de traitements sont localisés au niveau du serveur.

Les limites de ce système sont :

- Problèmes de performances lorsqu'un grand nombre d'utilisateurs se connecte en parallèle.
- Impossibilité d'écrire des traitements complexes au niveau du serveur.
- Sollicitation importante du réseau pour pallier les insuffisances du serveur, notamment en matière de traitements.
- Mauvaise tolérance aux pannes et peu de mécanismes de répartition de charge : le lien entre client et serveur est direct, ce qui provoque un blocage complet du système lorsque l'un des éléments est défaillant.
- Difficulté de déploiement et d'installation systématique de l'application et des couches d'accès au SGBD sur chaque poste utilisateur, ce qui implique une taille et une complexité importantes du client.



b) Architecture 3-tiers

Le client n'accède pas directement au serveur de bases de données. Il émet des requêtes à un serveur d'application qui exécute les traitements et transmet les requêtes au serveur de données. On parle d'architecture à 3 niveaux (ou tiers) :

Dans cette architecture, on parle de niveaux ou de tiers pour identifier les différents éléments que constituent client, serveur d'application et serveur de base de données. A chaque niveau correspondent un rôle et des activités précises :

- **Client** : où est placée la couche présentation (interface homme/machine)
- **Serveur d'application :** On y trouve l'ensemble des traitements applicatifs séparés des données physiques. La logique de l'application est déplacée ici. Ce poste est un relais entre le client et le

SGBD, mais il n'est pas restreint à un simple rôle de transmetteur/récepteur puisque la logique applicative y est localisée. On y trouve de vrais traitements et règles de gestion qui peuvent n'avoir aucun lien direct avec les données physiques : ils sont écrits dans n'importe quel langage (Java, C++, Delphi, etc.) et, ainsi, ne souffrent d'aucune limitation quant à leur réalisation.

- **Serveur de base de données :** Comme en client/serveur de première génération, ce serveur sert au stockage de données et à l'exécution des requêtes d'accès à la base données.

Les avantages de cette seconde génération de client/serveur sont multiples :

- Centralisation des traitements, ce qui facilite la maintenance et les évolutions.
- Partage possible entre des traitements communs à plusieurs applications.
- Dissociation entre le nombre d'utilisateurs et les connexions serveur : un serveur de traitement peut multiplexer les accès utilisateur et partager une même connexion au SGBD.
- Prise en charge de la communication entre un client et plusieurs SGBD différents au sein d'une même application.
- Facilité du déploiement sur le poste client. Seul un module doit être déployé et la couche d'accès aux données est localisée sur le serveur de traitement.

A coté de ces avantages, d'autres problèmes sont toujours à résoudre :

- La communication entre les différents éléments n'est pas standardisée, ou alors à un bas niveau (par exemple, TCP/IP), ce qui alourdit la charge de développement.
- Le concept objet n'apparaît pas ici : on appelle à distance des traitements sous la forme de RPC (Remote Procedure Call), comme s'ils étaient locaux. Absence de principes de réutilisabilité, de maintenance, de fiabilité fournis et mis en évidence par la programmation orientée objet.
- Il manque une couche commune, un ensemble d'outils à notre disposition pour simplifier les développements et, surtout, pour nous éviter une réécriture systématique de certains modules utilitaires.

c) Architecture n-tiers

Dans les applications distribuées, les fonctions sont réparties entre plusieurs systèmes. On les appelle aussi architectures multi-tiers ou modèle client/serveur multi-niveaux. L'architecture n-tiers a été pensée pour pallier aux limitations des architectures 3-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 3-tiers met en œuvre une approche objet pour offrir une plus grande souplesse d'implémentation et faciliter la réutilisation des développements. Théoriquement, ce type d'architecture :

- Permet l'utilisation d'interface utilisateurs riches
- Sépare nettement tous les niveaux de l'application
- Offre de grandes capacités d'extension
- Facilite la gestion des sessions

L'appellation « n-tiers » pourrait faire penser que cette architecture met en œuvre un nombre indéterminé de niveaux de service, alors que ces derniers sont au maximum trois. 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 objet. Elle permet de tirer pleinement partie 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.