

Présentation du cours

Ce cours se veut un aperçu de l'environnement de développement et d'exécution actuel des applications. Il va nous amener à fréquenter de nombreuses notions et matières que nous n'aurons guère l'occasion que d'effleurer. Cette année, nous aborderons le concept d'applications distribuées et nous nous attacherons à quelques notions liées au développement Web et d'outils génériques à ce type d'applications (les autres notions seront abordées l'an prochain). Pour l'exemplification, nous faisons le choix de nous diriger vers les solutions liées à Java – c'est dans ce domaine que vous, et nous, avons le plus l'habitude de travailler – mais nous aurions pu nous orienter vers les solutions .NET à forte connotation MicroSoft.

Plan du cours

- *Historique de la distribution*
- *Éléments intervenant dans une page reçue par un navigateur : HTML, CSS, javaScript, Ajax*
- *Technologies XML : XML, DTD, XSL*
- *Génération de pages JSP et servlets sous Java EE*

Remarque : ce cours ne s'intéressera aucunement aux problématiques liées à la conception et la mise en œuvre des réseaux nécessaires à la distribution.

Historique et problématique

Applications monolithiques

Au commencement existèrent les applications « monolithiques » (d'une seule pièce : cf. p.ex. vos prg Cobol sur l'AS/400 ou sur le mainframe).

Avantages :

- Cohérence
- Sécurité
- Environnement homogène

Inconvénients :

- Environnement autiste
- Environnement propriétaire (dépendance à un fournisseur : cf. hégémonie IBM)
- Limitation des possibilités d'échange d'information ou de services avec d'autres systèmes (difficulté d'utilisation de package, d'outils end-user, ...)

Application 2-tiers

L'apparition d'un modèle de SGBD 'universel' favorisa la répartition des gestions sur deux composants distincts (éventuellement deux machines). La gestion de la persistance était confiée au SGBD (généralement relationnel) l'autre tiers s'occupant de l'accès aux données, la logique métier et la gestion de l'interface utilisateur.

L'architecture Client/Serveur était née !

Cette architecture favorisa la séparation de la logique de persistance de la logique métier et donc la réutilisation de définition de données et de traitements, la séparation de notions liées à la gestion des performances (indépendance physique), des moyens de facilitations de conception et maintenance (indépendance logique). L'apparition de ce type de SGBD offrit aussi des outils génériques de sécurisation (physique, d'accès et logique [outils de gestion de transaction et de concurrence]).

Avantages

- Séparation de la gestion de persistance et de l'application
- Première possibilité de travailler avec des environnements hétérogènes

Inconvénients

- Les différents inconvénients présentés précédemment persistent pour la partie amont de l'application.
- Cette partie fait explicitement référence au modèle relationnel.

Notions sous-jacentes à la distribution

Avant de poursuivre l'historique de la distribution, voyons déjà quelles notions s'imposent déjà.

Client-Serveur

Même si la gestion de persistance fut un des premiers incitants au développement de cette architecture, ce n'est pas le seul environnement d'utilisation.

Remarque : les définitions fournies ici proviennent de « Le Client-Serveur », Georges et Olivier Gardarin, Eyrolles.

Client (client)

Processus demandant l'exécution d'une opération à un autre processus par envoi d'un message contenant le descriptif de l'opération à exécuter et attendant la réponse à cette opération par un message en retour

Serveur (server)

Processus accomplissant une opération sur demande d'un client et transmettant la réponse à ce client.

Requête (request)

Message transmis par un client à un serveur décrivant l'opération à exécuter pour le compte d'un client.

Réponse (reply)

Message transmis par un serveur à un client suite à l'exécution d'une opération contenant les paramètres de retour de l'opération.

Pour être envoyé à une machine distante, nécessitant souvent un codage spécifique, on doit réaliser l'« **assemblage** » (marshalling) du message. Lors de la réception du message, on doit en assurer le « **désassemblage** » (unmarshalling).

Dans une application monolithique l'appel d'une procédure s'effectue par un débranchement de l'appelant vers l'appelé. Dans un environnement distribué cela va se complexifier mais doit, idéalement, ne pas alourdir le code de l'appelant.

Appel de procédure à distance (Remote Procedure Call - RPC)

Technique permettant d'appeler une procédure distante comme une procédure locale, en rendant transparents les messages échangés et les assemblages/désassemblages de paramètres.

Les RPC seront implémentés au travers de la notion suivante :

Souche (stub)

Représentant d'une procédure sur un site client ou serveur capable de recevoir un appel de procédure du client et de transmettre en format adapté à l'implémentation ou à son représentant.

Remarque : les notions présentées jusqu'ici ne tiennent aucun compte de l'OO apparue postérieurement.

Les médiateurs (middleware)

Cette architecture Client-serveur impose un dialogue entre deux parties suivant des conventions qui dépendent des interlocuteurs. Il est évident que nos programmes seront « pollués » par du code ou des spécifications liés au produit utilisé (pour reprendre le cas de la gestion de persistance par un sgbd relationnel, le langage de requête sera généralement SQL qui se veut normalisé mais dont chacune des implémentations propose ses propres spécificités, de plus chaque sgbd risque d'avoir ses propres représentations des types élémentaires).

Pour pallier ce manque d'indépendance, des produits de généralisations sont apparus : les middleware. Leur rôle est de permettre l'utilisation de produits similaires suivant des spécifications générales. L'exemple, que vous avez fréquenté, le plus représentatif des middleware est ODBC.

Médiateur (Middleware)
Ensemble de services logiciels construit au-dessus d'un protocole de transport afin de permettre l'échange de requêtes et des requêtes associées entre client et serveur de manière transparente.

Le médiateur assurera donc la transparence aux réseaux (interface standard), aux serveurs (uniformisation des requêtes – ex uniformisation de SQL), aux langages (utilisable avec tout langage e.a. doit assurer les conversions de types).

Avantages :

- Facilitation de développement et déploiement d'applications dans des milieux hétérogènes
- Dégagement du code de particularités liées à un produit
- Facilité de développement de solutions génériques

Inconvénients :

- Overhead
- Difficulté d'utiliser des fonctionnalités tout à fait spécifiques au produit utilisé même si elles peuvent parfois se révéler très utiles.

Application 3-tiers

Assez rapidement, le besoin de répartition en 3 tiers s'est fait sentir :

Présentation – Logique métier – Accès aux données persistantes.

L'architecture 2-tiers présente comme inconvénient majeur de laisser la possibilité de mêler le code de présentation au code 'Métier'. Les applications 3-tiers permettent de faire exécuter la logique métier, la gestion de persistance et la gestion de la présentation sur des machines séparées (il est à noter que cette séparation devrait être présente dans l'architecture d'une application même si elle est exécutée sur une seule machine).

Le terminal utilisé comme interface homme-machine devient le serveur de présentation et n'assure que la gestion de la présentation (validations de saisies, adaptation de la présentation aux saisies, ...) . Il réalisera des appels au serveur assurant la gestion de la logique métier. Ce dernier sera le seul à accéder au serveur de données.

Exemple : Prêt d'un livre :

UI	Logique métier	Données
Sélection du lecteur. du livre	Recherche du lecteur. du livre	Trouve lecteur. livre
Saisie de l'emprunt	Transaction	
	Peut emprunter ?	
	Peut être emprunté	
	Entérine l'emprunt	
	Fin transaction	
Message Ok		

Avantages :

- Accroissement de la productivité et de la maintenabilité,
- Possibilité de distribution des tâches des développeurs,
- Optimisation de l'utilisation des ressources matérielles,
- Plus grandes possibilités d'utiliser les outils les plus appropriés à telle ou telle tâche.

Inconvénients :

- Conception paraissant plus 'lourde' a priori,
- Complexité croissante de l'architecture.

Application n-tiers

Il s'agit uniquement ici d'une généralisation du modèle 3-tiers : les données peuvent être réparties sur plusieurs SGBD, les livres peuvent être disponibles au travers de plusieurs bibliothèques, des serveurs spécialisés dans certains types de traitements (calcul scientifique...) peuvent être sollicités.

Client léger

De plus en plus souvent, l'interface homme-machine est délivrée par un navigateur. Dans ce cas, un serveur Web aura à générer dynamiquement les pages à afficher à l'utilisateur. Ces pages sont idéalement dépourvues de tout code exécutable, le terminal étant essentiellement 'passif'.

Remarque : En pratique, le client recevra souvent du code à exécuter pour permettre de réduire le nombre de requêtes au serveur (validation de certaines saisies, affichage d'éléments dynamiques, ...). L'exécution de ces codes se fait généralement au travers d'utilisation de plug-in. Ces codes sont généralement fournis sous forme de code JavaScript intégré au code Html (des choses plus complexes peuvent être réalisées au travers d'applets).

Nous essayerons cette année de parcourir le minimum de notions nécessaires au développement d'applications 'serveur' pour des clients légers.

Avantages :

- Déploiement facilité
- Terminaux peu gourmands en ressources

Inconvénients :

- Si on veut limiter le code exécuté sur le terminal, les interfaces sont relativement pauvres
- Sécurisation difficile

Client riche

On l'a dit, si on se limite aux balises HTML une page web propose un interface relativement pauvre. Le client riche enrichi l'interface des pages webs pour les rapprocher de ce qui est possible avec un client lourd (onglets, arborescence, calendrier, ...). Ajax est une technologie parmi d'autres pour enrichir les clients légers.

L'Orientation Objet

Toute la présentation précédente fait l'impasse sur le paradigme actuel de développement et exécution d'application : l'orientation objet.

Chacun des composants de l'application peut être développé en OO mais ce que nous avons abordé jusqu'ici ne permet pas qu'ils partagent des objets.

Pour disposer de la notion d'objets répartis, il faut un ORB (Object Request Broker), middleware par lequel les objets communiquent à distance.

L'OMG (Object Management Group) définit des standards d'architecture à objets répartis pour assurer l'interopérabilité entre des applications orientées objets sous le nom de CORBA (Common Object Request Broker Architecture).

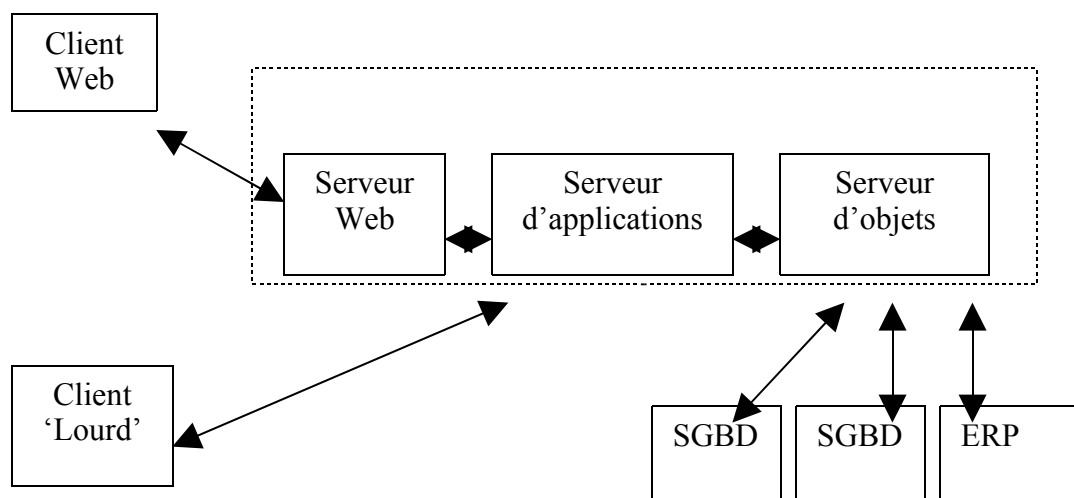
Les ORB les plus courants sont COM/DCOM (Components Objects Model/ Distributed Components Objects Model) de Microsoft, l'API RMI (Remote Method Invocation) de java, ...

Nous ne mettrons en œuvre ceci, et encore de manière entièrement transparente, que l'année prochaine.

Serveur d'application (Application server)

Un serveur d'application est l'environnement dans lequel les applications vont s'exécuter et offrir leurs fonctionnalités à un ensemble de clients. Un tel serveur gérera la montée en charge des applications (éventuellement en étant lui-même réparti sur plusieurs machines) et offrira des possibilités de reprises après incidents.

Cette notion est relativement controversée mais peut être étendue à la notion de server Web (génération de page HTML), gestionnaire de la logique métier (sens premier de serveur d'application), et serveur d'objets.



Il s'agit donc de l'environnement dans lequel peuvent s'exécuter la partie médiane des applications n-tiers.

En fonction du fait qu'un serveur d'application recouvre ou non les parties 'serveur d'objets' et 'serveur Web' plus ou moins de services et facilités seront offerts :

- Gestion de transactions,
- Gestion de montée en charge,
- Gestion de persistance,
- Gestion de sécurité,
- Transparence de la distribution,
- ...

Java 5 EE

Nous utiliserons dans le cadre de ce cours le *Sun Application Server* de Sun qui implémente les spécifications j2ee¹, appelées maintenant Java 5 EE, de Sun. D'autres implémentations de cette norme existent.

Java 5 EE est orienté Java et présente de multiples spécifications couvrant l'ensemble des problématiques présentées précédemment. Cette année nous n'aborderons que certains éléments liés au serveur Web.

¹ Java 5 EE est un framework de frameworks: Un framework est un ensemble de classes abstraites collaborant entre elles pour faciliter la création de tout ou partie d'un système logiciel. Un framework fournit un guide architectural en partitionnant le domaine visé en classes abstraites et en définissant les responsabilités de chacune ainsi que les collaborations entre classes. Un framework est habituellement implémenté à l'aide d'un langage à objets, bien que cela ne soit pas strictement nécessaire. Le déploiement à grande échelle de bibliothèques d'objets exige un framework ; celui-ci fournit un contexte où les composants sont ré-utilisés. (cf. Wikipedia)