Test du cours Génie Logiciel

20 décembre 2001

Durée 2h

Documents autorisés

Partie I Questions rapides / réponses rapides (4 points) :

A/ Expliquer les principes de fonctionnement de la liaison statique

B/ Expliquer les principes de fonctionnement de la liaison dynamique

C/ Quel est le principe de la liaison virtuelle ?

D/ Donner un exemple du polymorphisme.

Partie II : Application - Modèle Client / Serveur (16 points)

Définitions : Un **service** est un ensemble de fonctions mises à la disposition des programmes d'application et généralement partageable entre ces programmes. Très souvent, un service logiciel gère des ressources. La notion de service n'est pas nouvelle, ce qui est nouveau c'est la possibilité de dissocier la localisation de l'application appelante et la localisation d'une partie ou de la totalité des fonctions de service appelé. On dira que l'on est en mode **client / serveur** lorsque le service appelé incorpore un dispositif logiciel qui lui permet d'être exécuté localement ou à distance et de façon transparente pour l'application qui a recours à ce service. On observe que le service comporte en fait deux parties : une partie proche de l'application appelante nommée **demandeur** et une partie éloignée chargée de gérer une ressource appelée **gestionnaire de ressource**. Par extension, on appelle **client** le processus qui contient l'application appelante et la partie **serveur**, le processus qui contient la portion déportée du service.

Le **middleware par échange de messages** constitue une des approches qui permet d'échanger des données à travers le réseau sans avoir à connaître son fonctionnement et la localisation des différentes applications (client et serveur).

L'idée de base est de prendre en compte un bus unique de communication auquel chaque application se connecte. Lors d'un dialogue entre deux applications (client et serveur), chacune voit l'autre à travers le bus de communication de façon logique et ne connaît pas son adresse physique. Dans cette structure les applications communiquent en échangeant des messages. Elles fonctionnent indépendamment les unes des autres et indépendamment du bus de communication. Cette indépendance est obtenue par l'utilisation de files d'attente. Une file d'attente constitue un tampon qui permet à une application d'envoyer un message sans se préoccuper de la disponibilité du bus et de l'application destinatrice, ainsi que de sa localisation. La correspondance "nom logique de l'application" -> "nom physique" & "localisation" est à la charge du bus.

Dans ce modèle à chaque application sont associées deux files d'attente (une d'entrée, l'autre de sortie). Le dialogue entre deux applications fonctionne de la façon suivante :

1. Chaque application s'attache à ses deux files d'attente (d'entrée et de sortie) par l'opération Attacher_Files. Ces files représentent l'accès au bus de communication.
2. L'application initiatrice de dialogue dépose le message qu'elle veut envoyer dans sa file de sortie (primitive Dépose_message). Ce message contient en plus des données à transmettre le nomde l'application destinatrice
3. Le programme de gestion du bus scrute les files de sortie des applications et pour chaque message déposé traduit le nom logique de l'application destinatrice en son adresse physique et transfère le message à travers le réseau dans la file d'attente d'entrée de cette application.
4. L'application destinatrice consulte de temps en temps sa file d'entrée (par Lire_message) et récupère le message.
5. Après l'avoir traité, elle peut envoyer en retour un message de résultat, ou plus généralement de compte-rendu, en utilisant le même mécanisme (Dépose_message).
6. Ce message est transporté par le bus dans la file d'attente d'entrée de l'application initiatrice du dialogue qui la récupère par Lire_message.

Trois schémas sont envisageables :

- le cas simple dans lequel une application fait appel à un seul serveur (base de données par exemple),
- le cas où l'application fait appel à plusieurs services qui peuvent être proposés ou non par le même serveur (accès aux fichiers, à la base de données et à l'imprimante). On parle de **configuration logique en parallèle**,
- le cas de services imbriqués où un service est lui-même client d'un autre service (accès à la base de données qui fait appel à des services d'accès aux fichiers). On parle alors de **configuration logique en série**.

L'intérêt de l'approche client/serveur consiste en particulier dans le fait qu'un serveur peut être sollicité « en même temps » par plusieurs clients. Il est donc important d'assurer le bon fonctionnement du serveur par rapport aux multiples sollicitations. Cela peut être assuré par différentes approches :

a. par attente dans une file la disponibilité du service,
b. par dédoublement (démultiplication) du serveur sur un multi-processeur (ou sur machines différentes) en créant plusieurs processus remplissant le même service pouvant fonctionner en même temps,
c. par l'orientation du travail vers un dispositif disponible ou avec une attente moindre (pool d'imprimantes équivalentes),
d. par la possibilité de choisir le service le moins sollicité ou le moins coûteux (imprimante laser, jeu d'encre, matricielle,…).

On vous demande de proposer un simulateur de ce type de client-serveur (par échange de messages) et des différentes configurations (simple, parallèle, séquentielle) à l'aide de tâches ADA et/ou multithreading JAVA. L'objectif du simulateur est de tester les différentes configurations des serveurs par rapport à un ensemble de demandes des clients pour se rendre compte si potentiellement la configuration proposée est viable ou non. Il s'agit de :

1. donner un exemple d'utilisation pour chacun des trois cas,
2. donner une description simplifiée du programme client (contenant en particulier les différentes sollicitations des serveurs),
3. indiquer avec précision ce qui transite entre le client et le serveur,
4. dégager l'architecture globale du simulateur montrant graphiquement sa structure (tâches et paquetages la composant, ainsi que les liens de dépendance),
5. décrire les interfaces des composants identifiés (tâches et paquetages ADA et/ou objets Java) et donner le principe de fonctionnement se trouvant dans leurs corps,
6. fixer les objectifs à la simulation,
7. montrer comment gérer le temps dans la simulation.