Université des sciences et technologies de Lille (USTL)
Master mention Informatique – M1
Module : Conception d'applications réparties (CAR)

Juin 2008

Examen – 2nde session

Conception d'applications réparties (CAR)

Tous documents papier autorisés. Téléphones et ordinateurs interdits.
Le barème est donné à titre indicatif. 3 heures.
Ce sujet comporte 4 pages.

1 Conception d'un protocole client/serveur (12 points)

On s'intéresse à la conception d'un protocole client/serveur de transfert de fichiers. Les serveurs gèrent un ensemble de fichiers. Les clients se connectent sur le serveur pour récupérer un fichier.

1.1 Le serveur est-il avec ou sans état permanent ? Si la réponse est avec état, quel est l'état ? Sinon, pourquoi n'y a-t-il pas d'état ? (1 point)

1.2 Plusieurs clients peuvent-ils transférer simultanément le même fichier ? Justifier. (1 point)

En cas d'interruption du transfert, suite à une panne réseau ou à une panne de serveur, on souhaite ne pas avoir à reprendre le transfert depuis le début lorsque le service redevient opérationnel (pour que les clients ne perdent pas le contenu des fichiers déjà transférés). On se base pour cela sur un mécanisme de « point de reprise majeur ». Tous les 100Ko transférés, le serveur envoie au client un message `PRep` (point de reprise majeur). Un point de reprise majeur est aussi envoyé au début, avant tout transfert de donné. Chaque point est numéroté à partir de 0. A la réception d'un message `PRep`, le client acquitte la réception en envoyant au serveur un message `PRepAck` pour indiquer au serveur qu'il a bien reçu le point de reprise.

1.3 Lors d'une demande de transfert de fichier par un client, quelle modification cette fonctionnalité entraîne-t-elle au niveau du protocole ? Que doit indiquer en plus le client ? (1 point)

1.4 Lors de l'envoi d'un message `PRep`, le serveur attend d'avoir reçu un acquittement (`PRepAck`) avant de continuer. Citer deux fonctionnalités qu'un tel mécanisme permet de réaliser. (1 point)

On introduit maintenant un nouveau mécanisme dit « point de reprise mineur ». Entre deux points de reprise majeurs, le serveur envoie un message `PRepMin` (point de reprise mineur) après 50 Ko transférés. Les points de reprise mineurs ne sont pas acquittés (i.e. les clients ne renvoient rien suite à leur réception).

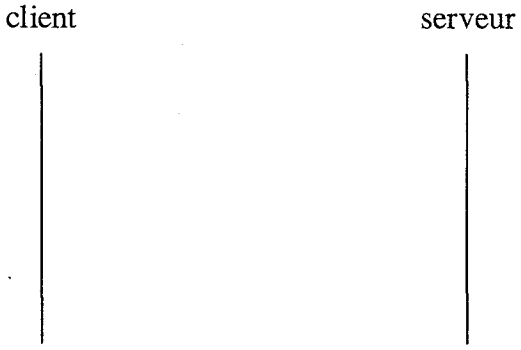
Etude des messages échangés par le protocole client/serveur

On considère les **primitives** suivantes :

- `ouvrirCx(serveur)` primitive de demande d'ouverture de connexion vers `serveur`
- `attendreCx()` primitive d'attente bloquante et acceptation d'une demande de connexion

- `envoyer(message)` primitive d'envoi d'un message
  `message` peut prendre une des valeurs suivantes :
  - `TRANSFERT` *`nom i j`* demande de transfert du fichier *`nom`*
    message envoyé 1 fois à chaque demande d'un nouveau transfert
    si `j=0` demande de transfert à partir du point de reprise majeur `i`
    si `j=1` `demande` de transfert à partir du point de reprise mineur suivant le point de reprise majeur `i`
  - `DATA` *`i j`* envoi d'un bloc de données de au plus 50 Ko
    si `j=0` envoi à partir du point de reprise majeur `i`
    si `j=1` envoi à partir du point de reprise mineur suivant le point de reprise majeur `i`
  - `PRep` *`i`* envoi du point de reprise majeur numéro `i`
  - `PRepAck` envoi de l'acquittement d'un point de reprise majeur
  - `PRepMin` envoi du point de reprise mineur
  - `FIN` pour signaler la fin du transfert
- `recevoir()` primitive d'attente bloquante et réception d'un message
  (`message = recevoir()`)
- `fermerCx()` fermeture de connexion
- `nbBloc100K(fic)` retourne le nombre de bloc de au plus 100 Ko du fichier `fic` (3 pour un fichier de 225 Ko)

1.5 Dans le cas particulier où le fichier s'appelle `fic` et contient 225 Ko, indiquer sur un diagramme la séquence de **messages** échangés par le client et le serveur pour un transfert complet du fichier. Les messages sont ceux définis ci-dessus et uniquement ceux-là. On ne demande pas de faire apparaître les primitives sur ce diagramme. (2 points)



Implantation des services

On se replace dans le cas où la taille du fichier est quelconque. On considère les primitives suivantes :
- `client(fic,i,j)` exécutée par le client pour demander le transfert du fichier `fic` à partir de :
  si `j=0` du point de reprise majeur `i`
  si `j=1` du point de reprise mineur suivant le point de reprise majeur `i`
- `serveur()` exécutée par le serveur pour envoyer le fichier au client.

Dans un but de simplification, on suppose que :
- en cas de réception d'un message non attendu, les procédures sont interrompues et une erreur signalée à l'utilisateur,
- on ne s'intéresse pas à la façon dont le client écrit les données du fichier qu'il reçoit.

1.6 En utilisant les primitives de la question précédente, écrire le pseudo-code ou le code Java des primitives `client(fic,x,y)` et `serveur()`. (6 points)

2 CORBA – Gestion d'un parking (8 points)

On considère un parking dont on souhaite automatiser la gestion en l'administrant à distance avec un *middleware* CORBA. Le parking peut accueillir au maximum 50 véhicules. Une barrière gère les entrées et les sorties de véhicules : elle délivre un ticket à l'entrée d'un véhicule et elle contrôle le ticket fourni par l'automobiliste à la sortie. Un automate de paiement permet sur présentation du ticket, de régler le montant correspondant au temps de stationnement. L'automobiliste règle le montant du stationnement à l'automate avant de franchir la barrière pour sortir. La barrière, l'automate et le parking sont représentés chacun par un objet CORBA.

L'interface `BarriereItf` de la barrière fournit deux méthodes :
- `entrer` : correspond à l'entrée d'un véhicule dans le parking. Retourne vrai si le véhicule peut rentrer (si place libre), faux sinon (et dans ce cas le véhicule ne peut pas entrer). Cette méthode fournit en paramètre de sortie une structure `Ticket` comprenant un numéro de ticket de type entier et une heure d'entrée sous la forme d'une chaîne de caractères.
- `sortir` : correspond à la sortie d'un véhicule. Retourne vrai si l'automobiliste a réglé le montant du stationnement, faux sinon (et dans ce cas le véhicule ne peut pas sortir). Cette méthode prend en paramètre d'entrée une structure `Ticket`.

L'interface `AutomateItf` de l'automate fournit deux méthodes :
- `payer` : correspond au paiement en fonction de la durée de stationnement. Retourne vrai si le montant fourni par l'automobiliste est suffisant, faux sinon. Cette méthode prend en paramètre d'entrée une structure `Ticket` et un montant de type réel double. Cette méthode fournit également en sortie une valeur de type réel double représentant la monnaie à rendre à l'utilisateur.
- `cestpaye` : à partir d'une structure `Ticket` fournie en entrée, retourne vrai si le montant du ticket a été réglé, faux sinon.

L'interface `ParkingItf` du parking fournit trois méthodes :
- `nbPlacesLibres` : retourne un entier indiquant le nombre de places libres dans le parking.
- `entreeVehicule` : signale l'entrée d'un véhicule sur le parking. Ne prend aucun paramètre, ne retourne rien.
- `sortieVehicule` : signale la sortie d'un véhicule du parking. Ne prend aucun paramètre, ne retourne rien.

2.1 Définir la structure `Ticket` et les interfaces IDL correspondant à ces trois objets CORBA dans un module `ParkingPkg`. (2 points)

Arrivée d'un véhicule

2.2 Lorsqu'un automobiliste se présente à la `Barrière`, l'objet CORBA barrière interroge l'objet `Parking` avant de laisser entrer l'automobiliste. Pourquoi ? Quelle méthode invoque-t-il ? (0,5 point)

2.3 À l'issu de cela, il se peut que l'objet `Barrière` invoque la méthode `entreeVehicule` de l'objet `Parking`. Cette méthode n'a pas de paramètres et a pourtant un rôle essentiel. Lequel ? (0,5 point)

Sortie d'un véhicule

2.4 Décrire le scénario de la sortie d'un véhicule du parking : quel(s) objet(s) appelle(nt) quelle(s) méthode(s) de quel(s) autre(s) objet(s) et pourquoi ? En particulier, on s'attachera à fournir un scénario de fonctionnement permettant à l'objet `Barrière` d'autoriser ou non la sortie du véhicule. (1 point)

Implémentation du parking

2.5 Ecrire le code de la classe `ParkingImpl` implémentant l'interface `ParkingItf`. (1 point)

Ajout d'une barrière

2.6 On souhaite ajouter une deuxième barrière d'entrée/sortie au parking. Cela modifie-t-il les interfaces définies à la question 1 ? Si oui comment, si non pourquoi. (0,5 point)

2.7 La classe `ParkingImpl` de la question 5 est modifiée suite à l'ajout d'une barrière. Après avoir rappelé les méthodes invoquées lors de l'arrivée d'un véhicule, expliquez en quoi consiste cette modification (on ne vous demande pas de la programmer). (1 point)

2.8 Même question pour la sortie d'un véhicule. (1 point)

Evolution du parking

2.9 On souhaite maintenant connaître les heures d'entrée et de sortie des véhicules. Sans donner le code, expliquez comment vous pourriez faire cela. (0,5 point)

Université des sciences et technologies de Lille (USTL)
Master mention Informatique – M1
Module : Conception d'applications réparties (CAR)

Janvier 2008

Examen

Conception d'applications réparties (CAR)

Tous documents papier autorisés. Téléphones et ordinateurs interdits.
Le barème est donné à titre indicatif. 3 heures.
Ce sujet comporte 3 pages.

1 Questions de cours (3 points)

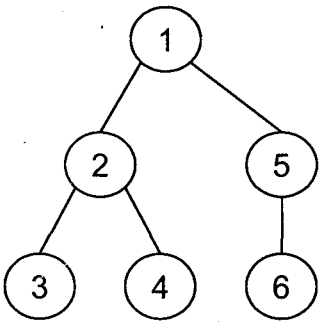
1.1 En Java RMI, que permet de faire la méthode `Naming.lookup` ? Quelle interface est implémentée par l'objet retourné par `Naming.lookup` ? Quelle est la classe de l'objet retourné par `Naming.lookup` ?

1.2 La transmission d'objets en Java RMI se fait-elle par référence (si oui, dans quel cas) ? par copie (si oui, dans quel cas) ?

1.3 Lorsque l'on souhaite transmettre un graphe d'objets en Java RMI, à quoi sert l'interface `Serializable` ?

2 Diffusion de messages (11 points)

On considère un protocole client/serveur qui permet de diffuser des messages à un ensemble de sites organisés selon une topologie en arbre. Chaque nœud de l'arbre propage le message à ses fils. Par exemple, dans la figure ci-dessous, pour diffuser son message, le site 1 l'envoie en parallèle à 2 et 5, puis 2 l'envoie en parallèle à 3 et 4 tandis que 5 l'envoie à 6.



2.1 Quel peut-être l'avantage d'une diffusion en arbre par rapport à une diffusion habituelle (par exemple de type Multicast IP) ? (1 point)

On considère que les sites communiquent en UDP sur le port 5000 en envoyant des messages de 2048 octets (contenu quelconque), et que chaque site a :

- une variable `pere` qui contient l'adresse de son père dans l'arbre (`null` si pas de père),
- un tableau `fils` qui contient les adresses de ses fils (tableau vide si pas de fils).

2.2 Ecrire en Java le code d'un nœud intermédiaire (dans l'exemple 2 et 5 sont des nœuds intermédiaires). (2 points)

On suppose maintenant que les nœuds feuille (3, 4 et 6 dans l'exemple) renvoient à l'émetteur un accusé de réception (message de 8 octets contenant des données quelconque) lorsque le message arrive. Lorsqu'un nœud intermédiaire a reçu les accusés de notification de tous ses fils, il renvoie à son propre père (1 dans l'exemple) un accusé de réception (message de 8 octets contenant des données quelconque).

2.3 Compléter le code Java de la question précédente en y incluant ce comportement. (1 point)

RMI

On considère maintenant que les nœuds de l'arbre ne communiquent plus par UDP mais via RMI. On considère également que les diffusions ne se font plus uniquement à l'initiative du site racine (1), mais que n'importe quel site de l'arbre peut initier une diffusion. Chaque site a une variable `pere` qui est maintenant une référence d'objet RMI et un tableau `fils` qui est un tableau d'objets RMI. Les messages diffusés sont des tableaux d'octets. Les accusés de réception sont des booléens.

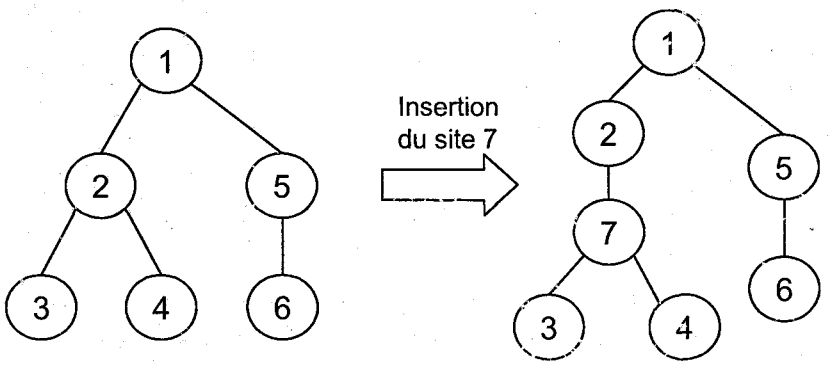
2.4 Proposer en français un algorithme qui permet de diffuser un message à tous les sites de l'arbre à partir de n'importe quel site (on rappelle qu'aucun site ne connaît la topologie complète de l'arbre, mais que chaque site ne connaît que son père et ses fils). (1 point)

2.5 Proposer une interface RMI (`SiteItf`) et sa classe d'implémentation (`SiteImpl`) mettant en œuvre cet algorithmique. L'interface et la classe doivent pouvoir servir pour tous les sites de l'arbre. (2 points)

2.6 Votre solution permet-elle de supporter plusieurs diffusions initiées simultanément par différents sites de l'arbre. Si oui pourquoi ? Si non, quelle modification faudrait-il apporter à la solution ? (1 point)

Servlet

On souhaite mettre en place une servlet pour gérer l'arbre de diffusion. La servlet doit permettre d'insérer et de retirer un site dans l'arbre. La servlet interagit avec les sites de l'arbre via Java RMI. À titre d'exemple, on souhaite que la servlet puisse insérer un site 7 comme illustré ci-dessous.



2.7 Que faut-il modifier au niveau de l'interface `SiteItf` et de la classe `SiteImpl` pour permettre à la servlet de fonctionner ? (0,5 point)

2.8 On suppose que l'on dispose d'un formulaire permettant de saisir le nom de l'objet RMI (paramètre `nouveauSite`) que l'on veut insérer dans l'arbre et le nom de l'objet père (paramètre `sitePere`) auquel on veut attacher cet objet. Écrire le code Java de la servlet qui réalise l'insertion. (1,5 point)

2.9 Que se passe-t-il en cas de modification concurrente par différentes personnes de la structure de l'arbre ? Que faudrait-il faire pour résoudre le problème ? (1 point)

3 Ping-pong CORBA (4 points)

Soient deux objets CORBA A et B :
- l'objet B implémente l'interface `BItf` qui fournit une méthode `ping` prenant en paramètre d'entrée une référence d'objet CORBA de type `AItf`. La méthode `ping` ne retourne rien (type de retour `void`). La méthode `ping` appelle la méthode `pong` sur la référence passée en paramètre.
- l'objet A implémente l'interface `AItf` qui fournit une méthode `pong` (aucun paramètre, type de retour `void`). La méthode `pong` est vide.

3.1 Écrire le code CORBA IDL des interfaces `AItf` et `BItf`. (1 point)

3.2 Écrire des classes `AImpl` et `BImpl` qui implémentent respectivement, les interfaces `AItf` et `BItf`. Écrire le code de la méthode main de la classe Client qui instancie les objets A et B et appelle la méthode `ping` sur l'objet A. (1 point)

3.3 On souhaite mesurer le temps mis par les invocations de méthodes pour faire l'aller-retour (appel de `ping` suivi de `pong`). À titre indicatif, la méthode Java `System.currentTimeMillis()` retourne un entier long qui correspond à l'heure courante. Quelle méthode faut-il modifier pour réaliser cette mesure ? Donner le code de cette modification. (0,5 point)

3.4 On considère maintenant que les méthodes `ping` et `pong` sont `oneway`. Peut-on toujours mesurer le temps pour faire l'aller-retour (appel de `ping` suivi de `pong`) de la même façon que précédemment ? Justifier votre réponse. En cas de changement, proposer une modification du code. (1,5 point)

4 IDL CORBA (2 points)

Indiquer quelles sont les erreurs contenues dans l'interface IDL suivante. Expliquer brièvement en quoi consistent les erreurs et proposer une correction. (2 points)

```
public interface Carte {
    const long echelle;
    typedef enum {nord,sud,est,ouest} Cardinaux;
    void orientation( Cardinaux direction );
    interface Michelin {
        string numero();
        long echelle() { return echelle; }
    };
};
```