Université des sciences et technologies de Lille (USTL) Mars 2007
Master mention Informatique – M1
Module : Conception d'applications réparties (CAR)

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 Questions de cours (2 points)

1.1 Quels sont les modes d'invocation de méthodes définis par CORBA ? Décrire pour chacun d'eux le comportement du client.
1.2 Parmi les modes d'invocation de méthodes définis par CORBA, quels sont ceux qui ont un équivalent en Java RMI ?

2 Ping-pong (11 points)

Soient deux objets A et B. L'objet A envoie un message `PING` à l'objet B qui lui répond par un message `PONG`. La 1ère partie de l'exercice met en œuvre cet échange avec des *sockets* Java, la 2ème avec Java RMI et la 3ème avec XML-RPC.

Sockets Java

Les messages échangés via les *sockets* ont la structure suivante :
- 1 octet pour identifier le message. L'octet vaut 0 pour `PING` et 1 pour `PONG`.
- 1 octet pour le nombre de paramètres associés au message.
- Les octets suivants contiennent les paramètres associés au message.
  Pour chaque paramètre, on trouve :
  - 1 octet pour son type. L'octet vaut 0 si le paramètre est un entier et 1 si le paramètre est une chaîne de caractères.
  - La valeur du paramètre.
    - Les entiers sont codés sur 2 octets. L'octet de poids fort est placé en tête.
    - Les chaînes de caractères sont précédées d'un octet qui fournit la taille de la chaîne. Chaque caractère occupe 1 octet.

Le message `PING` est associé à 5 paramètres. Chaque paramètre est un entier. Les 4 1ers correspondent à l'adresse IP de la machine qui envoie le message `PING` (par exemple 163.172.139.14). Le 5ème paramètre correspond au numéro de port TCP sur lequel la machine attend la réception d'un message `PONG`. Le message `PONG` est associé à un paramètre de type chaîne de caractères.

On suppose que la machine A connaît l'adresse IP et le port TCP sur laquelle la machine B reçoit les messages `PONG`. Soit l'échange suivant :

- la machine A envoie PING 163.172.139.14 2048 à la machine B,
- la machine B reçoit ce message, en extrait l'adresse IP et le port TCP et utilise ces informations pour envoyer à la machine A un message `PONG` avec la chaîne de caractères « Ok ».
- la machine A reçoit le message `PONG`.

2.1 Donner la séquence d'octets transmis par la machine A à la machine B. (1 point)
2.2 Donner la séquence d'octets transmis par la machine B à la machine A. À titre d'information, le code ASCII du caractère O est 79 et celui du caractère k est 107. (1 point)
2.3 Donner le code Java du programme qui s'exécute sur la machine A. (1,5 point)
2.4 Donner le code Java du programme qui s'exécute sur la machine B. (1,5 point)

Java RMI

Les machines A et B communiquent maintenant via Java RMI. La machine A implémente l'interface `AItf` et la machine B implémente l'interface `BItf`.

2.5 Précédemment le message `PING` était associé à 5 paramètres (adresse IP + numéro de port TCP). Dans le contexte de Java RMI, par quoi suggérez-vous de remplacer ces paramètres ? (1 point)
2.6 Donner le code Java des interfaces `AItf` et `BItf`. (1 point)

On suppose que l'objet RMI s'exécutant sur la machine B est enregistré dans l'annuaire rmiregistry sous le nom « machineB ». On suppose que la machine A possède une méthode `main` qui démarre l'échange en invoquant la méthode `PING`.

2.7 Donner le code Java des classes `AImpl` et `BImpl` implémentant respectivement les interfaces `AItf` et `BItf`. La méthode `PONG` affiche à l'écran le message qu'elle reçoit. (2 points)

XML-RPC

2.8 Écrire le code Java des classes implémentant l'échange des messages PING et PONG en XML-RPC. (2 points)

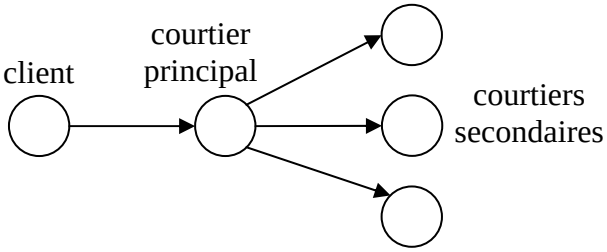
3 CORBA – Moteur de recherche d'objets (7 points)

On souhaite définir un moteur de recherche d'objets CORBA. Il s'agit donc de définir un service qui sera implémenté par un ou plusieurs objets CORBA appelés courtiers, et qui permettra de retrouver à partir d'une requête soumise par un utilisateur, la ou les adresses des objets CORBA correspondant au service demandé. Pour cela, l'objet CORBA courtier définit deux méthodes `enregistrer` et `rechercher` prenant les paramètres suivants :

- `enregistrer` : à partir d'une référence d'objet CORBA et d'un tableau de taille quelconque de couples propriété, valeur (propriété est une chaîne de caractères et valeur peut prendre n'importe quel type CORBA), cette méthode retourne un tableau de références d'objets CORBA.
- `rechercher` : à partir d'une chaîne de caractères correspondant à une expression régulière sur les caractéristiques des objets recherchés et d'un identifiant de requête de type entier fourni par le client, retourne un tableau de taille quelconque de références d'objets CORBA.

3.1 Donner en IDL CORBA l'interface `CourtierItf` définissant ces deux méthodes. (1 point)

Un courtier peut ne pas avoir la réponse à une requête de recherche. On prévoit dans ce cas qu'il délègue la recherche à des courtiers dits secondaires.



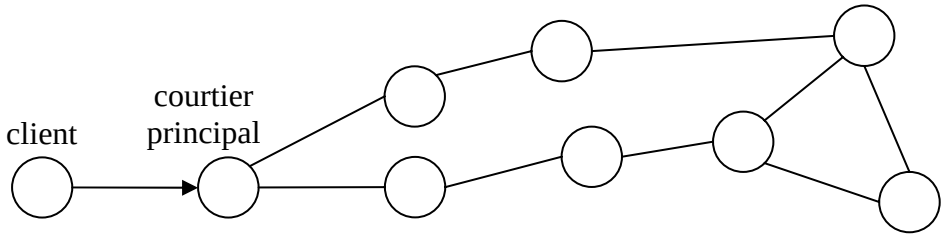
3.2 L'interface `Courtier2Itf` des courtiers secondaires peut-elle être identique à celle du courtier principal (`CourtierItf`) ou doit-elle être différente ? Justifier. (0,5 point)

3.3 Les courtiers secondaires peuvent-ils être invoqués concurremment par le courtier principal ou doivent-ils être invoqués les uns après les autres ? Justifier. (0,5 point)

3.4 Le courtier principal gère une liste de courtiers secondaires. On souhaite pouvoir ajouter et retirer des courtiers secondaires de cette liste. Proposer une interface IDL `GestionItf` permettant de faire cela. (0,5 point)

3.5 Etant données les interfaces `CourtierItf` et `GestionItf` implémentées par le courtier, quelle notion du langage IDL permet de définir le plus simplement possible l'interface complète `TotalItf` du courtier ? Donner en la définition en IDL. (0,5 point)

On considère maintenant que les courtiers secondaires peuvent eux-mêmes ne pas avoir la réponse à la requête. On offre donc la possibilité de relier les courtiers (secondaires et principal) à d'autres courtiers créant ainsi un graphe quelconque d'interconnexion entre courtiers.



Chaque courtier du graphe possède une liste de références vers ses courtiers voisins et implémente les interfaces `CourtierItf` et `GestionItf`. On considère que les arcs de ce graphe sont bidirectionnels (i.e. si un courtier peut dialoguer avec son voisin, le voisin peut aussi dialoguer avec lui).

3.6 Vis-à-vis de l'interface `GestionItf` quel problème nouveau ce graphe bidirectionnel pose-t-il ? Parmi les services CORBA vus en cours, quel est celui qui offre les propriétés permettant de résoudre un tel problème ? (1 point)

On suppose que chaque courtier :
- gère un tableau `voisins` de références CORBA implémentant l'interface `CourtierItf` (chaque élément de ce tableau est une référence vers un voisin),
- possède une méthode locale `idDejaTraite` prenant en paramètre un identificateur de requête de type entier et retournant vrai si la requête a déjà été traitée localement par ce courtier, faux sinon,
- possède une méthode locale `idTraite` prenant en paramètre un identificateur de requête de type entier permettant d'indiquer que la requête est traitée localement par ce courtier,

- possède une méthode locale `reponsesLocale` prenant en paramètre une chaîne de caractères correspondant à l'expression régulière d'une requête et retournant un tableau de références CORBA (le tableau est égal à la référence nulle si aucune réponse n'est disponible en local, sinon est égal aux réponses locales),

Lorsqu'il est interrogé, soit un courtier peut répondre localement et dans ce cas il fournit ses réponses et uniquement celles-là, soit il ne peut pas répondre localement et dans ce cas il délègue le traitement de la requête à ses voisins.

3.7 Donner le code Java de la méthode rechercher. (3 points)