Université de Lille 1 Mai 2015
Master mention Informatique – M1
Module : Conception d'applications réparties (CAR)

Examen

Construction des applications réparties (CAR)

Tous documents papier autorisés. Téléphones, calculatrices et ordinateurs interdits.
Le barème est donné à titre indicatif. 3 heures.
Ce sujet comporte 2 pages.

1 Questions de cours (3 points)

1.1 Donner le nom et la définition des deux modes de détection de panne qui existent dans les applications réparties.
1.2 Outre le fait qu'ils se basent sur Java, quel est le point commun, du point de vue de l'exécution des requêtes, entre RMI et les servlet ?
1.3 Donner le nom du format d'encodage de données utilisé d'un part par SOAP et d'autre part par RMI.

2 Interactions client/serveur (3 points)

Soit le protocole basé sur UDP entre un objet client et un objet serveur. Au démarrage, l'objet client connaît l'adresse IP et le numéro de port UDP utilisés par l'objet serveur. Le protocole débute par l'envoi par l'objet client à l'objet serveur du message `start` accompagné éventuellement de paramètres que vous ajouterez au besoin. Quatre modes d'interaction sont définis dans le protocole : (1) `notification`, un seul message envoyé par le client au serveur, (2) `requête-réponse`, un message de requête envoyé par le client, suivi d'une réponse du serveur, (3) `sollicite-réponse`, un message de sollicitation envoyé par le serveur, suivi d'une réponse du client, (4) `rappel`, un seul message envoyé par le serveur au client.

On considère un objet client qui envoie successivement les messages `start`, puis `notification`, puis `requête`. Simultanément, l'objet serveur envoie successivement un message `sollicite`, puis `rappel`. Chaque message contient un tableau de quatre octets. On considère que le message `réponse` renvoie le même tableau de quatre octets que celui qui a été reçu.

Écrire en Java ou dans un autre langage de programmation de votre choix le code de l'objet client.

3 Annuaire étendu (6 points)

Soient trois types d'objets Java RMI : les consommateurs, les producteurs et un annuaire, dont les interfaces sont respectivement `ConsItf`, `ProdItf` et `AnnuaireItf`.
Les producteurs fournissent la méthode `produire` qui prend en paramètre un message de type chaîne de caractères, qui ne retourne rien, et qui affiche le message à l'écran. Les producteurs possèdent une variable interne `annuaire` qui contient la référence de l'annuaire. On suppose cette valeur connue et vous n'avez pas à vous préoccuper de son initialisation.
L'annuaire fournit les quatre fonctionnalités suivantes.

1. Les producteurs s'enregistrent auprès de l'annuaire en fournissant un nom.
2. Les consommateurs recherchent un ou plusieurs producteurs auprès de l'annuaire. La recherche se fait en transmettant un nom pouvant inclure le caractère `*` pour représenter n'importe quelle séquence, éventuellement vide, de caractères. Par exemple, `deb*fin` correspond à la recherche de tous les producteurs dont le nom commence par `deb` et fini par `fin`.
3. L'annuaire doit pouvoir détecter la panne des producteurs qui se sont enregistrés auprès de lui, afin, lorsque cela survient, de transmettre une notification aux consommateurs qui ont obtenus précédemment la référence du ou des producteurs tombés en panne. La détection de panne doit pouvoir se faire selon les deux modes évoqués à la question 1.1.
4. Lorsqu'un producteur s'enregistre auprès de l'annuaire avec un nom qui satisfait une ou plusieurs requêtes précédemment transmises à l'annuaire par un ou plusieurs consommateurs, l'annuaire transmet la référence de ce nouveau producteur à ces consommateurs.

3.1 Proposer une solution pour les interfaces `ConsItf`, `ProdItf` et `AnnuaireItf`. (2 points)

3.2 Proposer le code Java ou le pseudo-code de la classe qui implante un producteur. (2 points)

3.3 Proposer le code Java ou le pseudo-code d'un consommateur qui recherche les producteurs de nom `deb*fin`, puis, toutes les cinq secondes, invoque la méthode `produire` sur successivement chacun des producteurs avec la chaîne de caractères `"Hello World"`. (2 points)

4 Gestion de magasin avec Java EE et REST (4 points)

On considère une application Java EE qui permet de gérer des clients et des factures. Chaque client a un identifiant (entier) et un nom (chaîne de caractères). Chaque facture a un identifiant (entier), un montant (réel double) et fait référence au client devant régler cette facture. On souhaite pouvoir créer, supprimer et consulter des clients et des factures. On souhaite également pouvoir calculer le montant total des factures réglées par un client. L'interface d'accès à l'application doit pouvoir se faire avec REST. On ne peut pas supprimer la ou les factures d'un client existant.

4.1 Quelle(s) ressource(s) REST proposez-vous de définir pour cette application ?Quel(s) type(s) de *bean*(s) session et/ou entité proposez-vous de définir ? (1 point)

4.2 Écrire le code Java des interfaces et des classes correspondantes. (3 points)

5 Élection sur un anneau (4 points)

L'algorithme « Élection sur un anneau d'objets RMI » vu TD fait l'hypothèse que chaque objet de l'anneau a un identifiant unique. Cet exercice supprime cette hypothèse : chaque objet a maintenant un identifiant quelconque, par exemple choisi au hasard, et dont la valeur change à chaque nouveau lancement d'une élection. Comme dans l'exercice vu en TD, l'algorithme est réparti et aucun objet n'a une connaissance globale de tous les autres objets de l'anneau.

5.1 Proposer en français une solution pour le test d'arrêt de l'élection, i.e. pour savoir que l'élection a fait le tour de l'anneau. (1 point)

5.2 Proposer en français une solution pour détecter lors d'une élection, les cas où deux ou plusieurs objets choisissent le même identifiant. (1 point)

5.3 Proposer en français une solution pour l'algorithme d'élection. (2 points)