Université Lille 1 Juin 2012
Master mention Informatique – M1
Module : Conception d'applications réparties (CAR)

Examen

Conception d'applications réparties (CAR)

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

1 Question de cours (3 points)

1.1 Dans les mécanismes d'invocation de méthodes à distance, à quoi sert une souche cliente ?
1.2 Comment partager simplement de l'information entre plusieurs EJB *session bean* sans passer par un *entity bean* ?
1.3 Quelle est à votre avis la différence conceptuelle fondamentale entre RMI et REST ?

2 Vidéo-club en ligne (8 points)

Cet exercice s'intéresse à la conception d'une application client/serveur de diffusion de films à base d'EJB et d'objets RMI. Elle met en relation un utilisateur et un vidéo-club en ligne. L'utilisateur se connecte au vidéo-club, demande un film et visualise en temps réel le film sur sa machine. Avant cela, il faut que l'utilisateur se soit abonné auprès du vidéo-club.

Gestion des abonnés et du catalogue de films

La gestion des abonnés et du catalogue de films est réalisée à l'aide de composants EJB.
Un abonné possède un numéro, un nom et une adresse.
Un film possède un titre, un genre (policier, SF, comédie) et une année de sortie.
On veut pouvoir créer un abonné et rechercher un abonné à partir de son nom.
On veut pouvoir créer un film et rechercher tous les films d'un genre donné.

Visualisation d'un film

La visualisation d'un film se déroule de la façon suivante :
1. l'utilisateur appelle la méthode `visualiser` en fournissant son numéro d'abonné et le titre du film qu'il souhaite visualiser,
2. si le numéro d'abonné et le titre du film existent, la méthode `visualiser` :
   - crée un objet RMI `Projecteur` qui va envoyer les images du film à l'utilisateur,
   - retourne à l'abonné la référence de cet objet.
3. l'utilisateur :
   - récupère la référence de l'objet `Projecteur`,

- crée un objet RMI `Ecran`,
- appelle la méthode `run` de l'objet `Ecran` lorsqu'il souhaite commencer la visualisation.

L'interface `ProjecteurItf` de l'objet RMI `Projecteur` possède les méthodes suivantes :
- `setEcran` : prend en paramètre un objet RMI de type `EcranItf`. Ne retourne rien.
- `play` : aucun paramètre, ne retourne rien. Permet de commencer la visualisation d'un film.

L'interface `EcranItf` de l'objet RMI `Ecran` possède les méthodes suivantes :
- `frame` : prend en paramètre un tableau de 1024 octets correspondant à une *frame* du film à visualiser. La visualisation complète d'un film correspond à plusieurs invocations de la méthode `frame`.

2.1 Quel type de composants EJB faut-il utiliser pour les abonnés ? Pourquoi ? Mêmes questions pour les films ? (1 point)

2.2 On souhaite mettre en place le *design pattern* Façade pour accéder à l'application. Donner le code Java de l'interface `VideoClubFacade` correspondant à cette façade. Pour cela, vous pourrez être amené à définir de nouvelles interfaces ou classes dont vous donnerez la signification (sans donner leur code). (1 point)

2.3 Proposer un diagramme d'échange de messages entre l'utilisateur, l'objet `Ecran` et l'objet `Projecteur` correspondant à la visualisation d'un film de 3*1024 octets. Proposer une solution, que vous expliquerez en français, pour que l'objet `Ecran` sache que le film est terminé. (1 point)

2.4 Ecrire le code Java des interfaces `ProjecteurItf` et `EcranItf`. (1 point)

2.5 Ecrire le code de la classe `Projecteur`. Le contenu du film à visualiser est fourni via le constructeur. (1,5 point)

2.6 On souhaite mettre en place une méthode `pause` pour arrêter temporairement la visualisation d'un film. La reprise de la visualisation se fait en appelant de nouveau la méthode `pause`. Expliquer en français quelles modifications au(x) interface(s) et au(x) classe(s) précédente(s) vous proposez pour mettre en place cette fonctionnalité. (0,5 point)

2.7 Plutôt que RMI, on propose maintenant de programmer la méthode `frame` de la classe `Ecran` à l'aide de *sockets* UDP. Expliquer les avantages et les inconvénients de cette solution par rapport à RMI. Donner le code Java de cette méthode (on ne demande pas le code correspondant à l'affichage que vous remplacerez par un simple commentaire). (2 points)

3 Service de diffusions d'informations (9 points)

Cet exercice s'intéresse à la conception d'un service de diffusion d'informations (de type `String`). Les trois entités suivantes sont présentes dans l'application : les producteurs qui produisent l'information, les consommateurs qui la consomment et les canaux de diffusion qui servent à mettre en relation les producteurs et les consommateurs. Un consommateur s'abonne auprès d'un canal en mentionnant qu'il est intéressé par un type d'information représenté par un identifiant (de type `String`). Les consommateurs peuvent s'abonner à plusieurs types d'informations et un producteur peut produire plusieurs types d'informations.

Les canaux fonctionnent selon deux modes : *push* et *pull*. Dans le mode *push*, un producteur ayant une information à produire la transmet au canal qui la retransmet à tous les consommateurs abonnés à ce type d'information. Dans le mode *pull*, un consommateur interroge le canal pour savoir si une information

d'un type donné est disponible, le canal interroge les producteurs qui, si ils disposent d'une information de ce type, la retourne au canal, qui la ou les (si plusieurs producteurs ont une information à produire) retransmet à tous les consommateurs abonnés.

3.1 Définir une ou plusieurs interfaces CORBA IDL pour gérer les abonnements et les désabonnements des consommateurs. Indiquer quelles entités implémentent quelles interfaces. Préciser le type que vous utilisez pour représenter les consommateurs. (1 point)

3.2 Définir une ou plusieurs interfaces CORBA IDL pour gérer le mode *push*. Indiquer quelles entités implémentent quelles interfaces. (1 point)

3.3 Définir une ou plusieurs interfaces CORBA IDL pour gérer le mode *pull*. Indiquer quelles entités implémentent quelles interfaces. (1 point)

3.4 On suppose maintenant que les canaux sont accessibles via REST. Décrire en français la façon dont vous mettriez en place une telle solution. (1 point)

En plus de *push* et de *pull*, un troisième mode de fonctionnement correspondant à une hybridation de *push* et de *pull* est envisageable.

3.5 Proposer une description en français du fonctionnement de ce troisième mode. (1 point)

On considère maintenant que les producteurs sont organisés selon une topologie en anneau et que un seul producteur, qui joue le rôle de point d'entrée dans l'anneau, est relié au canal. La recherche d'information se fait en parcourant l'anneau. On suppose que l'anneau existe et que chaque producteur sait s'il est un point d'entrée ou non. On suppose également que chaque producteur dispose d'une méthode privée `poll` qui retourne vrai si une nouvelle information est disponible pour la production et d'une méthode privée `get` qui retourne l'information à produire.

3.6 Définir une ou plusieurs interfaces CORBA IDL pour gérer le mode *pull*. Justifier en quoi la solution est similaire ou différente de la solution proposée pour la question 3.3. Écrire en Java l'implémentation d'un producteur membre de l'anneau (les méthodes privées `poll` et `get` peuvent être utilisées, mais on ne demande pas d'écrire leur code). (2 points)

Pour les anneaux comportant un très grand nombre de producteurs, le tour complet de l'anneau peut prendre un temps élevé. On introduit donc la notion de raccourci : en plus de son voisin dans l'anneau, un nœud peut connaître un raccourci, c'est-à-dire la référence d'un nœud situé plus loin dans l'anneau. Tous les nœuds n'ont pas forcément à leur disposition un raccourci. On fait les mêmes hypothèses qu'à la question précédente, et on ajoute le fait que les raccourcis sont connus.

3.7 Écrire en Java l'implémentation d'un producteur membre de l'anneau en exploitant cette notion de raccourci. (2 points)