Examen Construction d'Applications Réparties

Maîtrise Informatique

Université des Sciences et Technologies de Lille

2003-2004

3 heures – Tous documents autorisés

1. Serveur de consultation Java RMI (6 points)

On souhaite accéder par Java RMI à des données stockées sur un serveur distant. Les données sont stockées dans une liste. Chaque élément, de type `java.io.Serializable`, de la liste est associé à une clé unique de type `java.lang.String`.
Le serveur souhaite savoir qui consulte quelles données à quels instants. Il définit pour cela une notion de connexion. Avant d'accéder à une donnée, un client potentiel doit invoquer la méthode `connexion` du serveur. Cette méthode prend comme paramètre un nom d'utilisateur (`String`) et retourne un entier représentant un identifiant de connexion. Le serveur fournit également les méthodes suivantes :

- `consulter` : à partir d'un identifiant de connexion (entier) et d'une clé (`String`), retourne l'élément associé à la clé, ou `null` s'il n'y en a pas.
- `fermer` : à partir d'un identifiant de connexion (entier), ferme la connexion. Cette méthode ne retourne rien.

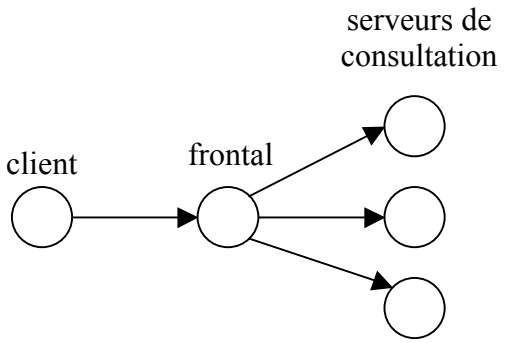
Par exemple, un scénario possible d'interrogation du serveur s va consister à invoquer les méthodes suivantes :

```
int cx = s.connexion("Bob");
Serializable obj = s.consulter(cx,"chauffagiste");
obj = s.consulter(cx,"électricien");
s.fermer(cx);
```

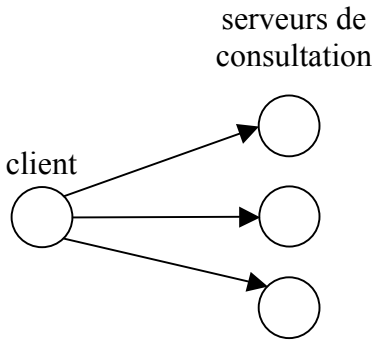
1 Définir l'interface Java RMI `ServeurItf` correspondant aux méthodes `connexion`, `consulter` et `fermer`. (1 point)

2 Rappeler les règles que doit respecter une classe implantant une interface Java RMI. (1 point)

Le volume de données augmentant, il ne devient plus possible de les stocker sur un seul serveur. On décide de les répartir sur plusieurs serveurs, et on envisage les deux architectures suivantes.



Architecture n° 1



Architecture n° 2

Dans la première, un frontal gère l'accès aux serveurs de consultation. Les clients ne connaissent que lui. Le frontal se charge de déléguer la requête aux serveurs de consultation. Le frontal connaît donc tous les serveurs de consultation.
Dans la seconde, les clients connaissent tous les serveurs de consultation et s'adressent directement au serveur dont ils souhaitent consulter les données.

3 En cas d'ajout ou de retrait d'un serveur de consultation, que faut-il faire avec la première architecture ? Avec la seconde ? Quelle est celle qui pose le moins de problème ? Pourquoi ? (1 point)

4 Lorsqu'il y a beaucoup de clients, quelle solution est la plus efficace ? Pourquoi ? (1 point)

Architecture n° 1

On se place dans le cas de la première architecture.

5 L'interface `ServeurItf` définie précédemment peut-elle être utilisée pour le frontal ? Pour les serveurs de consultation ? (0,5 point)

Soit `ConsultationItf` l'interface des serveurs de consultations. On souhaite définir une interface Java RMI `GestionFrontalItf` de gestion du frontal offrant les méthodes suivantes :
- `ajouter` : permet d'ajouter un serveur de consultation à la liste des serveurs gérée par le frontal. Prend un paramètre de type `ConsultationItf`. Ne retourne rien.
- `retirer` : permet de retirer un serveur de consultation de la liste des serveurs gérée par le frontal. Prend un paramètre de type `ConsultationItf`. Ne retourne rien.

6 Définir l'interface Java RMI `GestionFrontalItf`. Comment définir simplement l'interface complète du frontal ? (0,5 point)

7 Le frontal doit être capable d'aiguiller de façon simple les requêtes de consultation vers le bon serveur. Proposer un mécanisme de répartition des données sur les serveurs qui permettent de faire cela efficacement (on ne vous demande pas de l'implanter). (1 point)

2. Patron Observateur/Sujet avec CORBA (6 points)

Le patron de conception *Observateur/Sujet* permet de faire prendre en compte les changements d'un objet (le sujet) par d'autres objets (les observateurs). Cela assure, par exemple, la mise à jour des données sur les objets observateurs. Cette relation entre deux objets peut être explicitement codée dans la classe représentant le sujet, mais cela demande une connaissance sur la façon dont les observateurs doivent être mis à jour. Ce type de réalisation fait que les objets deviennent fortement couplés et ne peuvent pas être réutilisés.

Nous proposons donc une approche plus faiblement couplée par l'intermédiaire de deux classes `Observer` et `Subject` qu'il sera possible d'utiliser ou d'hériter. Un changement dans un objet devra être signalé aux autres par une notification, leur permettant ainsi de se tenir à jour.

L'utilisation de ce patron se fait donc en deux temps :

- Un observateur s'abonne et se désabonne d'un sujet par l'intermédiaire de deux méthodes offertes par l'objet sujet (`attach` et `detach`),
- Une fois l'observateur abonné au sujet, ce dernier le prévient lors de la modification d'un événement dans son environnement par l'appel de la méthode `notify` offerte par l'objet observer.

Interfaces

1. Les deux objets `Observer` et `Subject` ont des relations client/serveur. Décrire à quels moments ces objets ont une fonction de client et à quels moments ils ont une fonction de serveur. (1 point)

2. Donner l'interface IDL des deux objets `Observer` et `Subject`. Justifiez les méthodes et les paramètres employés. (1 point)

Plusieurs observateurs

3. Dans un second temps, il est demandé de modifier les interfaces des deux objets pour permettre l'abonnement d'un ensemble d'objets de type `Observer` à l'objet `Subject` et de proposer une notification sur l'ensemble des objets abonnés.(1 point)

Réalisation

4. Réaliser en Java/CORBA le code des classes associées aux interfaces `Subject` et `Observer` permettant la mise en place du patron observer/subject. Une notification sera émise toutes les secondes.(1,5 point)

5. Ecrire en Java/CORBA une infrastructure de tests permettant de positionner sur deux machines distinctes deux objets de type `Subject` et `Observer`. Cette infrastructure permettra de tester la connexion/déconnexion des composants de type `Subject` et `Observer` ainsi que la notification d'événement du composant de type `Subject` au composant de type `Observer`.(1,5 point)

3. Serveur Web (8 points)

Le but de cet exercice est d'implanter une partie du protocole HTTP/1.0 avec l'API *socket* du langage Java.

API *socket* Java

1 Quels sont en Java les différents constructeurs permettant la création d'une socket TCP? Expliquez les paramètres et les différents cas d'utilisation.(0,5 point)

Protocole HTTP

2 Ecrire le pseudo-code d'un serveur HTTP gérant la commande GET. On rappelle que la commande GET a la syntaxe suivante : `GET url HTTP/1.0|HTTP/1.1`. L'url à la forme suivante : `protocole"://"adresse[:port]["/"chemin]["/"nomdefichier]`. Les éléments entre crochets sont facultatifs. Les éléments entre guillemets sont des chaînes de caractères finales. (1 point)

Cette commande GET peut être suivie de lignes d'entêtes ayant la forme suivante et, finalement d'un retour chariot qui indique la fin de la commande.

```
User-agent: Mozilla/4.0
Accept: text/html, image/gif,image/jpeg
Accept-language:fr
```

La réponse à cette commande GET a la syntaxe suivante qui indique le protocole utilisé par le serveur `HTTP/1.0 ou HTTP/1.1`, un code réponse sous la forme d'un entier à trois positions et une chaîne de caractères qui donne un sens explicite au code, soit la forme suivante :

```
HTTP/1.0 code chaîne_de_caractères
```

Cette réponse est suivie par exemple des lignes d'entête suivantes, d'une ligne blanche et des données, soit la forme suivante :

```
Date: Thu, 06 jan 2004 12:00:15 GMT
Server: Apache/1.4.0 (Unix)
Last-Modified: Mon, 22 Dec 2004 …...
Content-Length: 6821
Content-Type: text/html

Data data data ….
```

Les deux codes de réponse que nous vous demandons de prendre sont 200 (tout est OK) et 404 (non trouvé).

Implantation en Java

3 Ecrire une classe `Serveur` avec une méthode `main`
- écoutant les demandes de connexion sur un port TCP > 1023
- délégant à l'aide d'un *thread* le traitement d'une requête entrante à un objet de la classe `HttpRequest`(1,5 point).

4 Ecrire en Java une classe `HttpRequest` comportant

- une méthode `processRequest` effectuant des traitements généraux concernant une requête entrante et déléguant le traitement de la commande GET à la méthode `processGet`
- une méthode `processGet` se chargeant de traiter la commande GET (1 point)

Les servlets

Nous souhaitons maintenant étendre le serveur http afin qu'il accepte le traitement de servlets. Une servlet est désignée par une URL de la forme `http://www.lifl.fr/servlet/Prog` où `servlet` désigne un répertoire où sont stockées les classes java (ici `Prog`) correspondant aux servlets disponibles.

5 Suite à la reconnaissance par le serveur web que cette url est une demande d'exécution d'une servlet, quelles sont les méthodes de la classe `HttpServlet` qui vont permettre l'appel et l'exécution de la servlet `Prog` ?(0,5 point)

6 Quel est le mécanisme mis en place par le serveur web qui rend les servlets persistantes ? (1 point)

7 Quel est le mécanisme mis en place par le serveur web qui permet l'exécution concurrente des servlets ? (0,5 point)

8 Décrire les étapes à mettre en place dans un serveur web permettant l'exécution d'une servlet. (1 point)

9 Ecrire les quelques lignes de code qui permettront à votre serveur web écrit avec les questions 3 et 4 de gérer l'exécution d'une servlet. Indiquer l'endroit où ce code sera ajouté. (1 point).