SCIENCES SORBONNE UNIVERSITÉ

SRCS : Systèmes Répartis Client/Serveur

TD 6 : Java Remote Method Invocation

Objectifs pédagogiques :

- **Transformation d'un protocole message vers des invocations distantes**
- **Service RMI**
- **Répartition de charge**
- **Interaction entre services RMI**

Exercice 1 – Administration de réseaux

Le protocole *SNMP* (*Simple Network Management Protocol*) est un protocole de l'Internet (RFC 1157) pour l'administration de réseaux. Son objectif est de décrire le mode de communication d'applications qui acquièrent et qui configurent, à distance et en mode client/serveur, les paramètres (adresse, tables de routage, statistiques d'utilisation, etc.) des équipements (ordinateurs, imprimantes, routeurs, etc.) connectés au réseau. Deux entités sont présentes dans les environnements administrés avec SNMP : les stations d'administration et les agents administrés.

Dans cet exercice, on souhaite étudier la transformation de ce protocole, conçu selon un mode message, en un mode invocations de méthodes distantes. On souhaite également définir des interfaces Java RMI pour ces invocations.

La version 1 du protocole *SNMP* comprend les 5 messages suivants :

- `GetRequest` : ce message, émis par une station d'administration vers un agent, permet à la station de demander à l'agent la valeur d'un de ses paramètres (l'identificateur du paramètre est transmis) ;
- `GetNextRequest` : ce message, émis par une station d'administration vers un agent, permet à la station de demander à l'agent la valeur du paramètre suivant dans la liste des paramètres de l'agent (l'identificateur du paramètre initial est transmis) ;
- `SetRequest` : ce message, émis par une station d'administration vers un agent, permet à la station de modifier la valeur d'un paramètre de l'agent (l'identificateur du paramètre et la nouvelle valeur sont transmis) ;
- `GetResponse` : ce message, émis par un agent vers une station d'administration, est la réponse de l'agent à une requête `GetRequest` ou `GetNextRequest` (la valeur du paramètre demandé est transmise) ;
- `Trap` : ce message, émis par un agent vers une station d'administration, permet à l'agent de transmettre une alarme (une alarme est un message du type "il n'y a plus de papier dans l'imprimante" ou "liaison vers l'extérieur interrompue").

Question 1

Représenter sur un diagramme les différentes possibilités d'échanges de messages engendrées par ce protocole. Quelles sont celles qui correspondent à un mode client/serveur ? Quelles sont celles qui relèvent du mode message.

Question 2

Dans le cadre de la transformation de ces 5 messages en invocations de méthodes distantes, quels sont ceux que vous conservez ? Quels sont ceux que vous éliminez ? À quoi correspondent les messages éliminés ?

Question 3

Faut-il définir une interface RMI pour les agents ? Pour les serveurs d'administration ? Dans les deux cas, pourquoi ?

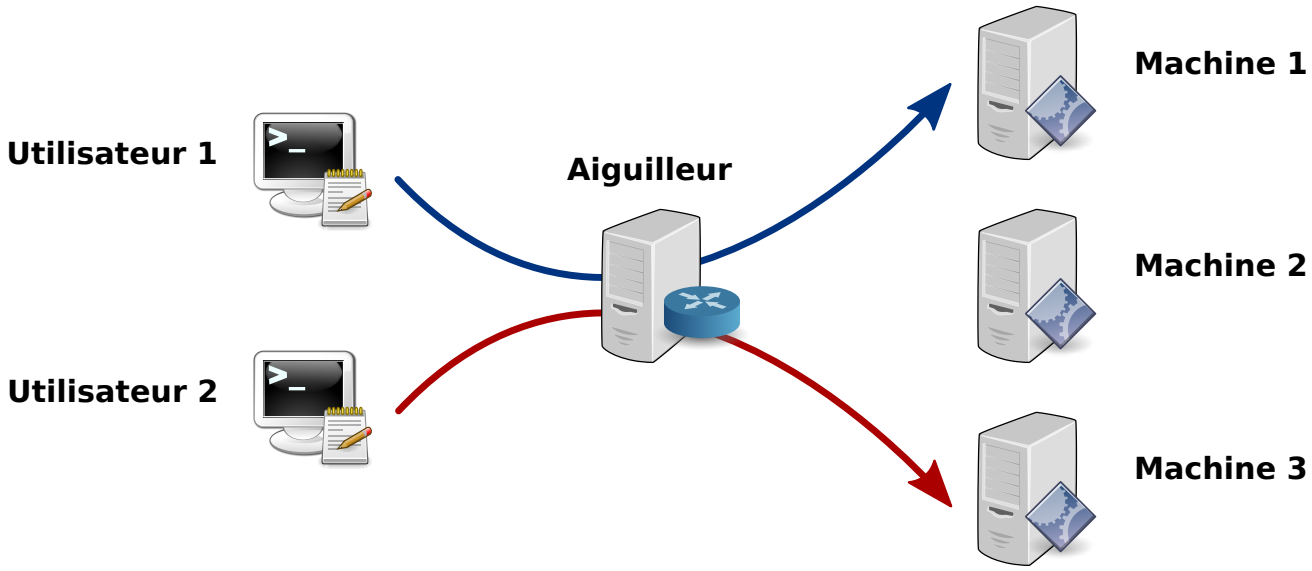
Question 4

En supposant que les identificateurs des paramètres des agents sont des tableaux d'entiers de taille quelconque, que les valeurs des paramètres sont des entiers, et que les messages d'alarme transmis par la primitive `Trap` sont des chaînes de caractères, donner la définition en Java RMI de la (ou des) interface(s) envisagée(s) à la question précédente.

Exercice 2 – Répartiteur de charges

Cet exercice a pour but d'étudier dans le cadre d'un environnement de programmation répartie un `Aiguilleur` de requêtes. Celui-ci permet de répartir la charge de traitements entre plusieurs `Machines`.

La figure ci-dessous présente l'architecture envisagée qui comprend deux `Utilisateurs`, un `Aiguilleur` et trois `Machines` identiques (i.e. fournissant les mêmes services). Les `Utilisateurs`, l'`Aiguilleur` et les `Machines` sont des objets RMI.



Les Utilisateurs soumettent des requêtes à l'`Aiguilleur`. Pour chaque requête, l'`Aiguilleur` choisit de la rediriger à une `Machine` et une seule. La `Machine` choisie traite la requête, fournit le résultat à l'`Aiguilleur` qui le retransmet à l'`Utilisateur`. Pendant le temps de traitement de la requête, l'`Aiguilleur` peut aiguiller une nouvelle requête vers une autre Machine ou vers la même. L'`Aiguilleur` est donc un objet concurrent. Le degré de concurrence des `Machines` est étudié à partir de la question 9.

Question 1

Pour chacun des trois types d'objets RMI (`Utilisateur`, `Aiguilleur`, `Machine`), dire s'ils sont client (de qui), serveur (pour qui) ou client/serveur (de qui et pour qui).

Question 2

Chaque objet `Machine` fournit deux services :

1. un service dit `lecture` qui à partir d'un nom de fichier (une chaîne de caractères) fourni en entrée, retourne les données (un tableau de taille non déterminée d'octets) correspondant au contenu du fichier,

2. un service dit `ecriture` qui à partir d'un nom de fichier (une chaîne de caractères) et de données (un tableau de taille non déterminée d'octets) fournis en entrée, écrit les données dans le fichier. Ce service retourne un booléen qui vaut vrai en cas de succès de l'opération d'écriture.

On suppose dans un premier temps que tous les objets `Machines` ont accès aux mêmes fichiers et que l'écriture dans un fichier est répercutée de façon immédiate et automatique sur toutes les `Machines`. Donner en Java RMI la définition de l'interface `Machine` contenant ces 2 services.

Question 3

Sans anticiper sur les questions suivantes et sans en donner la définition explicite, expliquer à partir des indications fournies jusqu'à présent, quelle doit être l'interface de l'`Aiguilleur`.

Question 4

On souhaite maintenant que des objets `Machines` puissent être ajoutés et retirés en cours d'exécution au "pool" d'objets géré par l'`Aiguilleur`.

Quelle(s) méthode(s) faut-il définir pour prendre en compte ces fonctionnalités ? Pour quel(s) objet(s) ? Définir en Java RMI, l'interface `Controle`, réunissant ces fonctionnalités. Donner en français la signification précise de chaque élément (méthodes, paramètres, etc.) défini.

Question 5

Proposer en français ou en pseudo code un algorithme, le plus simple possible, pour effectuer la répartition de charge. L'algorithme doit répartir équitablement les requêtes entre toutes les `Machines` sans tenir compte du fait que certaines requêtes peuvent être plus longues que d'autres.

Question 6

On souhaite maintenant prendre en compte les charges des `Machines`. Périodiquement les `Machines` informent l'`Aiguilleur` de leur niveau de charge. Celui-ci est représenté par un entier donnant le nombre de requêtes en attente sur la `Machine`. Quelle(s) méthode(s) faut-il définir pour prendre en compte cette fonctionnalité ? Pour quel(s) objet(s) ? Définir en Java RMI, l'interface `Notification` comprenant cette fonctionnalité.

Question 7

Finalement, comment construit-on l'interface complète de l'`Aiguilleur` prenant en compte l'ensemble des fonctionnalités le concernant ?

Question 8

Expliquer quel peut être le degré de concurrence acceptable pour une Machine en fonction des deux types de requêtes lecture et écriture.

Question 9

On considère maintenant que lors d'une opération d'écriture, la mise à jour du fichier sur toutes les `Machines` est à la charge de la `Machine` qui traite la requête d'écriture. Faut-il ajouter des définitions aux interfaces précédentes pour prendre en compte ce cas ? Si oui, le(s)quelle(s) ? Si non, pourquoi ?

Question 10

On s'intéresse maintenant aux incohérences qui peuvent survenir lorsque l'`Aiguilleur` répartit successivement deux requêtes d'écriture pour un même fichier sur deux `Machines` différentes. Expliquer en quoi consiste cette incohérence. Proposer pour gérer ce problème au niveau de l'`Aiguilleur`, une solution simple qui ne modifie aucune interface.

Question 11

L'`Aiguilleur` étant un point de passage obligé et de ce fait un goulet d'étranglement pour les requêtes et les réponses, proposer une solution n'introduisant pas de nouvelles entités pour alléger sa charge, notamment en ce qui concerne la gestion des réponses.

Question 12

Expliquer sans forcément les définir explicitement quelle(s) conséquence(s) cela entraîne sur les interfaces.

Question 13

Faire un diagramme de séquence entre les différentes entités du système lorsque l'on veut faire une écriture avec la solution de la question précédente. Vous représenterez pour chaque JVM le flux d'exécution de chaque thread concerné.