Université de Poitiers
Master Informatique M1
Architectures Client/Serveur

TP 5. Apprentissage par Problème et Projet : Communication asynchrone en RMI

Ce TP se déroule sur 4h, mais inclut seulement 2h d'encadrement. Il sera donc effectué en partie en autonomie. Il s'agit de trouver une solution pour éviter qu'un client ne reste en attente d'une réponse d'un serveur lorsque le traitement demandé à ce dernier s'avère trop long. En effet, s'il est possible en TCP de base d'éviter d'attendre un message, les retours d'appel de procédure distante sont TOUJOURS bloquants en RMI.

Vous travaillerez par groupe de 4 étudiants. Il vous faut trouver une solution au problème et l'implanter.

Le déroulement de la séance est le suivant :

- T0 à T0+15mn : exposé du sujet, constitution des groupes. Dans chaque groupe il est proposé d'avoir un animateur qui gère le débat, un scribe qui prend des notes des discussions et un maître du temps qui veille à ne pas dépasser le temps imparti ; ;
- T0+15mn à T0+1h45 : étude du problème et recherche de solution(s) par Internet, synthèse des solutions, choix de la solution retenue ;
- *Autour de T0+1h : Discussion intermédiaire avec le tuteur. Vérification des pistes et des questions que vous vous posez ;*
- Entre T0+1h45 et T0+2h15, suivant votre avancement : exposé de la solution au tuteur ;
  Un rapport manuscrit décrivant la solution sous forme de diagrammes de séquence sera demandé pour un exposé clair de votre solution ;
- T0+2h15 à T0+2h30 : Définition du/des contrat(s) RMI. Distribution des codes à écrire à chaque membre du groupe ;
- T0+2h30 à T0+3h30 : écriture des divers programmes ;
- T0+3h30 à T0+4h15 : synthèse des codes, tests, etc. Projet à rendre à la fin du créneau.

Vous serez évalué à la fois sur le déroulement de l'APP (=avancement + rapport intermédiaire) et sur le code final. Le déroulement contribuera au moins à 50 % de la note finale. Au besoin, les notes pourront être individuelles, si le groupe n'a pas travaillé de façon homogène.

Exposé du sujet

On suppose qu'un serveur offre un service de type Horloge. Le client le sollicite pour qu'il puisse recevoir *n* tops d'horloge toutes les *x* secondes, *x* et *n* étant connus du client dès le départ. A chaque top d'horloge, un entier ou un réel aléatoire est généré par le serveur et envoyé au client. Pendant ce temps, le client continue d'effectuer sa tâche principale (un calcul, un affichage long, peu importe). On parle ici de communication asynchrone lorsque le serveur a la possibilité de transmettre une donnée sans que le client interrompe sa tâche courante. Il n'est donc pas question ici d'adresser une requête classique, puisque les méthodes distantes sont bloquantes en RMI. Régulièrement, la tâche principale du client veut exploiter l'un des nombres aléatoires générés par le serveur, en les récupérant dans l'ordre d'arrivée. Si aucun nombre n'est disponible, le client attend le prochain, mais il faut lui donner la possibilité de vérifier la disponibilité des nombres afin de ne pas le bloquer inutilement. Aussi, il ne s'agit pas pour le client d'attendre que le serveur génère les *n* nombres en un coup, mais bien qu'il puisse les obtenir l'un après l'autre (à chaque top d'horloge).

Le but de cet exercice est de fournir une solution d'implantation d'un tel service. Le serveur se contentera de faire un ***Thread***.*sleep(x*1000)* pour simuler les tops d'horloge. Bien sûr, le serveur peut servir plusieurs clients en même temps et un même client peut vouloir solliciter en même temps plusieurs fois le serveur. Il faut également s'assurer que tout objet distant créé et devenu inutile soit effectivement désactivé et supprimé.

Critères d'évaluation de l'APP

Les aspects suivants seront pris en compte pour évaluer votre solution (JAVA) :

Respect des consignes (readme, commentaires bien choisis)	
Compilation sans warning (-Xlint:all -Xdiags:verbose)	
Exécution, tests concluants	
Respect des principes de mise en œuvre RMI (classes&interfaces, exceptions, méthodes)	
Définition du/des contrat(s) des objets distants	
Tâche principale du client effectivement exécutée en parallèle de la requête au serveur	
Assurance que le client reçoit bien *n* valeurs aléatoires toutes les x secondes	
Client pouvant lancer plusieurs requêtes au même serveur	
Limitation du nombre d'échanges	
Gestion de la fin de communication asynchrone	
Gestion du parallélisme et protection des données partagées	
Gestion de la charge du serveur (le cas échéant)	
Synchronisation souple entre le client et le serveur (client bloqué, à sa demande, si valeur non dispo)	