Objets Distribués : TP1 Java/RMI

Partie 1 : Hello World Reparti Exercice 1 Ecrire le Hello Word réparti sur 2 machines. Fichier interface RMI : interface Hello ... { string sayHello(); }; Faire en sorte que le serveur renvoie une chaîne "Hello World n°X" ou X est le nombre d'appels que le serveur a reçu. Aide : · les interfaces RMI doivent dériver de java.rmi.Remote et · les interfac RMI doivent avoir déclarer dans leur méthodes pouvoir lever une exception java.rmi.RemoteException (il faut donc écrire complètement « String sayHello() throws RemoteException ») · les classes d’implémentation doivent dériver de « java.rmi.server.UnicastRemoteObject » · n’oubliez pas d’utiliser un fichier policy (sécurité). Vous pouvez créer un fichier nommé par ex « secu_minimale ». le contenu sera : grant { // donne toute les permissions permission java.security.AllPermission; }; · l’appel se fait comme suit : java -Djava.security.policy=<fichier policy> <nom_de_classe> · le fichier policy est à utiliser coté client et coté serveur. Exercice 2 : appel d’un objet serveur tier Après l’avoir fait fonctionner sur votre machine, Essayer de faire fonctionner votre client sur le serveur d’un de vos co-etudiants et vice versa. Les fichiers que vous devez échanger sont : · le fichier d’interface bien sur ! (soit le .java soit le .class) · le fichier stub (un client doit avoir le fichier stub du serveur qu’il veut appeler). Exercice 3 : Enrichir la partie serveur - Modifier la classe d’implémentation pour que celle-ci reçoive un nom en paramètre du constructeur. Elle restituera ce nom à chaque appel de sayHello()-dans la chaine retournée. - Modifier le programme principal du serveur pour que celui crée plusieurs instances de l’OD avec des noms différents (3 objets différents par exemple) - Modifier le client pour que celui-ci puisse recevoir en paramètre le nom de l’instance de l’OD à appeler (en plus de l’IP). - Faites des appels au choix sur les 3 objets serveurs Exercice 4 : Client interactif Modifier le client pour qu'il boucle sur une attente clavier permettant de taper de commande permettant à l’utilisateur : - de faire l'appel à "sayHello" (par exemple avec la touche « h ») - demander au serveur de se terminer (par exemple avec la touche « x ») - de se terminer lui-même (par exemple avec la touche « s ») Aide : il y a des choses à modifier dans l’interface et dans l’implémentation ! Exercice 5 : chargement dynamique du stub Le but de cet exercice est de créer un client d’un objet RMI uniquement à partir du fichier d’interface (Hello.java) : le stub sera téléchargé.

En effet, dans le concept initial (en Objets distribués) c’est uniquement ce fichier d’interface qui doit être échangé comme un contrat entre le client et le serveur. Les fichiers que l’on aura de part et d’autre seront : Serveur Client Hello.java HelloImpl.java Helloserveur.java Policy Hello.class HelloImpl.class HelloImpl_Skel.class HelloImpl_Stub.class Hello.java HelloClient.java policy HelloClient.class Hello.class Le client devra pouvoir exécuter l’implémentation de Hello sur le serveur sans savoir son nom, en téléchargeant le stub associé depuis le serveur pendant l’exécution.

La configuration sera comme suit : L’essentiel se passe au moment du lancement du serveur, qui transmet via son environnement « java.rmi.server.codebase » au rmiregistry l’URL d’accès au stub. Marche à suivre coté serveur: récupérer le code d’un serveur Web (http) ICI (il provient du site de SUN)faire la même chose que précédemment pour la compilation du serveurATTENTION : lancer rmiregistry dans un autre répertoire que le serveur (rmiregistry ne doit pas avoir un accès fichier à la classe de stub)lancer le serveur web : start java WebServer 80lancer le serveur avec la nouvelle clause codebase. Par ex : java -Djava.security.policy=policy -Djava.rmi.server.codebase=http://<ip> HelloServervérifier que rmiregistry fait à ce moment là un accès au serveur Web Notes : ne pas mettre 127.0.0.1 en codebase (l’url est transmise telle quelle !)mettre un ‘/’ à la fin de l’url du codebase